A Temporal Description Logic Based Access Control Model for Expressing History Constrained Policies in Semantic Web

Fathieh Faghih, Morteza Amini, and Rasool Jalili
Sharif Network Security Center (NSC)
Department of Computer Engineering, Sharif University of Technology
Tehran, Iran
Tel: 98-21-66164020, Fax: 98-21-66054096
e-mail: {faghih@ce., m_amini@ce., jalili@}sharif.edu

Abstract

An access control model for Semantic Web should be compatible with the corresponding semantic model. The access control procedure(s) should also take the semantic relationships between the entities (specified as ontologies) into account. Considering the benefits of logic-based models and the description logic foundation of Semantic Web, in this paper, we propose an access control model based on a temporal variant of description logics ($\mathcal{TL}$-$\mathcal{ALCF}$). This logical schema enables us to express history constrained policies to enrich the policy-base with dynamic properties based on previous accesses. The specification of each component of the model as well as the approach to define history constrained policies along with some clarifying examples are presented. The access control procedure of the model is proposed over the inference services of $\mathcal{TL}$-$\mathcal{ALCF}$.

1. Introduction

With the growth of Semantic Web, security and access control for this environment have been extensively investigated. A security model for a semantic-aware environment should consider the defined semantic relationships between entities besides the security policies to infer the authorized accesses. On the other hand, one of the requirements in modern applications is the temporal aspect of authorization. In some applications, such as e-banking environments, policy definition with the constraint(s) based on previous users' accesses might be crucial. To consider both aspects in an access control model, this paper proposes a security model based on a temporal extension of description logics (DL) for Semantic Web.

SBAC (Semantic Based Access Control) is introduced as an access control model for Semantic Web, which takes semantic relationships into account [1]. In [2], we extended the SBAC model to express policies based on previous users' accesses. The limitation of this extension is that it is restricted to the policies at the level of individuals. Therefore, it confines security authorities to state policies at the concept level. Moreover, the extension utilizes a formal language without proof theory that is only used for stating policy constraints. This model is improved through a logical framework in this paper to state policies with a logic-based language. Using logic to state security policies has different advantages, such as non-ambiguity, abstraction from implementation, expressiveness, and verifiability [3], [4], [5]. Considering the Semantic Web properties, employing logic into the security model seems very promising. It is due to the applicability of logic in inferring the implicit policies from the explicit ones based on the semantic relationships in domains of subjects (access requesters), objects (resources), and actions. In Semantic Web, relationships are defined in ontologies . Considering DL superiority for definition, integration, and maintenance of ontologies, DL has always been applied as one of the supporting tools for Semantic Web [6]. Accordingly, DL is chosen as the basis for our proposed model in this paper.

Using description logic as a modeling language for different purposes has resulted in different extensions, including modal, epistemic, and temporal [7]. Temporal extension of DL has been extensively considered, mostly in Artificial Intelligence field [8]. Therefore, different temporal extensions to DL have been introduced [8], [9]. Artale and Franconi [9] proposed $\mathcal{TL}$-$\mathcal{ALCF}$ as a decidable interval based temporal description logic. They also introduced an approach for using $\mathcal{TL}$-$\mathcal{ALCF}$ in representing and reasoning about *actions* and *plans*, which is used on a robotic domain in [10]. A similar approach in definition of actions and plans (of actions and other plans) is employed in this paper to state history constrained policies.

This research is partially supported by Iran Telecommunication Research Center (IRTC).

978-0-7695-3742-9/09 $25.00 © 2009 IEEE
DOI 10.1109/POLICY.2009.15

This paper proposes an access control model, called TDLBAC, based on $\mathcal{TL}$-$\mathcal{ALCF}$. The model allows the security authorities to define history constrained policies at the concept level. TDLBAC components include two building blocks of DL; TBox and ABox. The model terminologies, including the semantic relationships of entities, as well as the specified security policies are included in the TBox. The knowledge related to the individuals and the history of accesses, are stored in the ABox. The proposed algorithm for access control in TDLBAC utilizes the inference services of DL to decide the received access requests based on the defined policies along with the history of accesses.

The rest of this paper is as follows: Section 2 discusses the related work. Section 3 is an overall description of the proposed model, along with an overview on necessary concepts of $\mathcal{TL}$-$\mathcal{ALCF}$. It also describes the model components with clarifying examples on policy definition. In Section 4, the algorithm for registering users' accesses, and the procedure for access control based on policies and history of users' accesses are introduced. A brief discussion of the algorithm time complexity is presented in Section 5. Finally, Section 6 concludes the paper.

2. Related Work

Since late 90's, logical ideas and tools have attracted many researchers to specify different kinds of security policies such as role-based policies [4], [5], information flow control policies [11], and dynamic and rule-based policies [12]. The temporal aspects of temporal logics motivated Bertino [12] to express dynamic policies. Furthermore, Bertino et al. [13] proposed TRBAC model, which considers temporal constraints in triggering roles. This model includes formal syntax and semantics. However, it is not based on a known logic, and inference is not possible in it. Therefore, the model can not be included among logic-based access control models.

Several efforts have been put into designing access control models for semantic-aware environments such as Semantic Web [1], [14]. Javanmardi *et al.* introduced SBAC that considers ontologies in three domains of access control [1]. The important novelty of the model is reducing semantic relationships to subsumption. This method reduces the temporal and spatial complexities, and simplifies the authorization propagation.

Chinese Wall Security Policy [15] can be mentioned as the first security policy based on the events in the past. Moreover, the security automata in [16], and Deeds system in [17] focus on collecting a selective history of sensitive access requests. This information is used in the procedure of access control. History-based access control for a semantic aware environment is taken into account by Noorollahi *et al.* in [2]. They extended the SBAC model, by adding two constraints; first, by limiting the authorization policies to special time intervals, and second, by adding constraints based on previous users' accesses to policy rules. This model is restricted to the policies in the level of individuals. Moreover, its formal language lacks proof theory, and is only used for stating policy constraints. This paper aims to improve this model with a logical foundation to take the advantages of a logic-based model in stating and inferring historical constrained access policies in conceptual (ontology) level.

Table 1. Syntax rules for temporal part of $\mathcal{TL}$-$\mathcal{ALCF}$

$\mathcal{TL}$	C, D	$\longrightarrow$	$E\vert$	(non-temporal concept)
			$C \sqcap D\vert$	(conjunction)
			$C@X\vert$	(qualifier)
			$C[Y]@X\vert$	(substitutive qualifier)
			$\Diamond(\overline{X})\overline{Tc}.C$	(existential qualifier)
	Tc	$\longrightarrow$	$(X(R)Y)\vert$	(temporal constraint)
			$(X(R)\#)\vert$	
			$(\#(R)Y)$	
	$\overline{Tc}$	$\longrightarrow$	$Tc\vert Tc\overline{Tc}$	
	R, S	$\longrightarrow$	$R, S\vert$	(disjunction)
			s|mi|f|...	(Allen's relations)
	X, Y	$\longrightarrow$	x|y|z|...	(temporal variables)
	$\overline{X}$	$\longrightarrow$	$X\vert X\overline{X}$	

3. TDLBAC

3.1. Preliminaries

In this section, the employed logic in this paper, named $\mathcal{TL}$-$\mathcal{ALCF}$ [9] is introduced briefly. $\mathcal{TL}$-$\mathcal{ALCF}$ is a composition of temporal logic $\mathcal{TL}$ and non-temporal description logic $\mathcal{ALCF}$. It is used for representing and reasoning about *actions* and *plans*. The syntax rules for the temporal part of $\mathcal{TL}$-$\mathcal{ALCF}$ are presented in Table 1. For the temporal part of the logic, the interval relations introduced by Allen [18] are used. The # variable refers to the current evaluation interval, or the time of occurrence in the case of actions. To evaluate a concept in another interval, for example $\mathbf{X}$, the $@\mathbf{X}$ expression is applied. The semantics of $\mathcal{TL}$-$\mathcal{ALCF}$ includes the interpretation $\mathcal{I} \doteq \langle \mathcal{T}^*_<,\ \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$, consisting of an interval set of the temporal structure ($\mathcal{T}$), domain of $\mathcal{I}$ ($\Delta^{\mathcal{I}}$), and interpretation function of $\mathcal{I}$ ($.^{\mathcal{I}}$). The function $.^{\mathcal{I}}$ gives meaning to concepts, features, and parametric features, as follows:

- $\mathcal{A}^{\mathcal{I}} \subseteq \mathcal{T}^* \times \Delta^{\mathcal{I}}$
- $f^{\mathcal{I}} = (\mathcal{T}^* \times \Delta^{\mathcal{I}}) \xrightarrow{partial} \Delta^{\mathcal{I}}$
- $*g^{\mathcal{I}} = \Delta^{\mathcal{I}} \xrightarrow{partial} \Delta^{\mathcal{I}}$

$*g$ is called atomic parametric feature in the syntax of $\mathcal{TL}$-$\mathcal{ALCF}$. Its difference from atomic feature in $\mathcal{TL}$-$\mathcal{ALCF}$ is being independent from time. More detailed information about syntax and semantics of $\mathcal{TL}$-$\mathcal{ALCF}$ can be found in [9].

3.2. Overall TDLBAC Model Description

TDLBAC (Temporal Description Logic Based Access Control) is our proposed access control model for expressing policies in Semantic Web. The policies in TDLBAC are defined at the concept level. Moreover, security authorities can define constraints based on previous users' accesses for their policies. The logic chosen for this purpose is $\mathcal{TL}$-$\mathcal{ALCF}$ [9]. The proposed approach in [9], [10] to express *actions* and *plans* is leveraged in this paper to state policies with constraint(s) based on previous users' accesses.

TDLBAC is composed of two DL components:

- **TBox** includes statements about a domain and relationships between the entities in the domain.
- **ABox** contains assertions related to individuals.

TDLBAC has two components related to history-based access control; History-Base (HB) and History-based Policy-Base (HPB). They are defined as follows:

- **HB** contains the history of previous users' accesses. HB is stored as a part of the ABox.
- **HPB** is a set of policies with history-based constraint(s). They are specified by security authorities, and stored in the TBox.

The model components (i.e. TBox and ABox) are discussed more formally in the following sections.

3.3. Model Terminologies

An access control model for Semantic Web should be compatible with the semantic model of this environment in the sense that decisions should be made based on the defined relationships and the resulting inferences. In TDLBAC, the axioms related to the domain are kept in the TBox. The axioms include the definitions of different concepts, as well as their logical relations. Besides semantic relationships, specifications of policy rules based on logic are also kept in the TBox. TBox includes five hierarchies of concepts. Accordingly, we have five concepts on top of each hierarchy, as follows:

- *Subject*: This is the concept on the top of the hierarchy of concepts that their individuals can have a role of active subject.
- *Object*: This is the concept on the top of the hierarchy of concepts that their individuals can be accessed by subjects.
- *Action*: This is the concept on the top of the hierarchy of concepts that their individuals specify access types.
- *Policy-Rule*: This is the concept on the top of the hierarchy of concepts specifying policy rules.
- *Access*: This is the concept on the top of the hierarchy of concepts that their individuals specify users' previous accesses in the system.

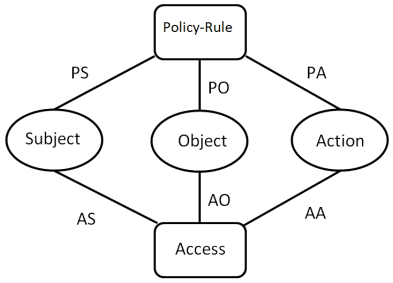

Figure 1. Main concepts and their relations

Note that in the rest of this paper, by subjects, we refer to the whole hierarchy, while *Subject* denotes the concept on top of this hierarchy. The logic used to define the concepts in the subjects, objects, and actions hierarchies is $\mathcal{ALCF}$. $\mathcal{ALCF}$, the non-temporal part of $\mathcal{TL}$-$\mathcal{ALCF}$, is an expressive, but still decidable DL.

In an access control system, the policy rules, as well as the requests and accesses are interpreted as triples, i.e. ⟨subject, object, action⟩. The first element (subject) is the entity requesting an access. The second one (object) is the entity on which the operation is performed, and the third element (action) indicates the type of the access. However, in a description logic system, we have only binary relations, expressed as roles or features (functional roles). In order to describe an access control model with DL, we represent each triplet with three binary relations. For example, an access denoted by $\langle s, o, a\rangle$, is represented by three pairs, $\langle x, s\rangle$,$\langle x, o\rangle$, and $\langle x, a\rangle$. To this end, in TDLBAC, we define three types of relations between the concepts of policy rules, and the concepts of subjects, objects, and actions, named as *PS*, *PO*, and *PA*, respectively. There are three similar relations between the concepts in the hierarchy of accesses, and the hierarchy of subjects, objects, and actions, named as *AS*, *AO*, and *AA* respectively. These concepts and their relations are shown in Figure 1 schematically. Relations are defined as parametric features in the syntax of $\mathcal{TL}$-$\mathcal{ALCF}$. Note that instances of policy rules have functional time-independent relations with instances of three concepts in subjects, objects, and actions hierarchies. Similar relations hold for access instances. To specify the type of the three parameters of an access or a permitted access in a policy rule, the selection operator in the syntax of $\mathcal{ALCF}$ is leveraged. The semantics of the selection operator in $\mathcal{ALCF}$ is as follows:

$$(p : C)^{\mathcal{I}} = \{a \in dom\ p^{\mathcal{I}} | p^{\mathcal{I}}(a) \in C^{\mathcal{I}}\}$$

To clarify, we first give the formal definition of the *Access* concept:

$$Access \equiv (*AS : Subject) \sqcap (*AO : Object) \sqcap (*AA : Action) \quad (1)$$

Intuitively, the *Access* concept can be interpreted as instances that have three parametric features: an *AS* with an individual of type *Subject*, an *AO* with an individual of type *Object*, and an *AA* with an individual of type *Action*. Therefore, a three parametric access can be defined under the *Access* concept in the hierarchy, similar to (1). In an analogous way, the *Policy-Rule* concept is defined as follows:

$$\begin{aligned} \textit{Policy-Rule} \equiv & (*PS : \textit{Subject}) \sqcap (*PO : \textit{Object}) \\ & \sqcap (*PA : \textit{Action}) \end{aligned}$$

Policy rules are defined under the *Policy-Rule* concept.

3.4. Ground Assertions

In a description logic system, ground assertions are stored in the ABox. These facts in TDLBAC are classified into three groups:

1) Assertions that state correspondence of individuals to the concepts defined in five aforementioned hierarchies in the TBox.
2) Assertions related to users' accesses in HB. They are used to check the history-based constraints in the access control algorithm as described in Section 4. They are added to the ABox after granting each request.
3) Assertions that state the type of subject individuals in each request. These are added to the ABox before starting the access control procedure as discussed in Section 4.

3.5. Clarifying Example

Now, we are ready to state a clarifying example of a history-based policy rule in TDLBAC. Consider an education system, where a security policy rule gives the read permission of the university website pages to all the students, in the case that an administrator had a grant update in the config-file in the past. The corresponding policy rule is expressed in this model as follows:

$$\begin{aligned} \textit{read-rule} \equiv & \Diamond x(x \ \ b \ \ \#)(*PS : \textit{student}) \\ & \sqcap (*PO : \textit{University-Website-Page}) \\ & \sqcap (*PA : \textit{read}) \sqcap \textit{grant-access}@x \end{aligned}$$

The defined policy rule is in the hierarchy of policy rules under the *Policy-Rule* concept, if and only if:

$$\begin{aligned} & \textit{student} \sqsubseteq \textit{Subject} \wedge \\ & \textit{University-Website-Page} \sqsubseteq \textit{Object} \wedge \textit{read} \sqsubseteq \textit{Action} \end{aligned}$$

In our example, the definition of the concept *grant-access* can be stated as follows:

$$\begin{aligned} & \textit{grant-access} \equiv (*AS : \textit{administrator}) \\ & \sqcap (*AO : \textit{config-file}) \sqcap (*AA : \textit{grant-update}) \end{aligned}$$

$\textit{grant-access}$ is subsumed by *Access*, if and only if:

$$\begin{aligned} & \textit{administrator} \sqsubseteq \textit{Subject} \wedge \textit{config-file} \sqsubseteq \textit{Object} \wedge \\ & \textit{grant-update} \sqsubseteq \textit{Action} \end{aligned}$$

Regarding the semantics of $\mathcal{TL}$-$\mathcal{ALCF}$, a concept is interpreted over pairs of temporal intervals and individuals. The interpretation of the concept *read-rule* includes pairs of interval # and individuals, for example, $\langle a, \# \rangle$. # is the current time interval (the time of the evaluation of *read-rule*). $\textit{grant-access}@x$ in the definition of $\textit{read-rule}$ indicates that if $\textit{read-rule}(a, \#)$ holds, then $\textit{grant-access}(a, x)$ is true, where x is an interval before # (i.e. $b(x, \#)$ holds with respect to the Allen's interval relations). Note that in order to define a policy rule based on previous users' access(es), a temporal variable (*x*) is defined and its relation with $\#$ is declared within the policy definition.

In many systems with history-based policies, security authorities prefer to give permission to the individuals who themselves have had special accesses. Suppose, for example, a banking system in which a policy is to give secured typed loans to business account holders who have repaid an unsecured loan before. The constraint that refers to giving permission to the same individual who has had an access is called *subject similarity* condition (i.e. similarity between the subject of the request and the subject of the logged access). To state *subject similarity* condition, we use the agreement operator ($\downarrow$) between features, defined in the syntax of $\mathcal{TL}$-$\mathcal{ALCF}$. To clarify, let us state the policy rule in banking example:

$$\begin{aligned} \textit{loan-policy-rule} \equiv & \Diamond x(x \ \ b \ \ \#) \\ & \sqcap (*PS : \textit{Business-Account-Holder}) \\ & \sqcap (*PO : \textit{secured-loan}) \\ & \sqcap (*PA : \textit{get}) \\ & \sqcap \textit{repay-unsecured-loan}@x \\ & \sqcap (*PS \downarrow *AS) \end{aligned} \tag{2}$$

To explain the policy definition in (2), we take the semantics of $\downarrow$ operator into account [9]:

$$(p \downarrow q)_t^{\mathcal{I}} = \{a \in \textit{dom } p_t^{\mathcal{I}} \cap \textit{dom } q_t^{\mathcal{I}} | p_t^{\mathcal{I}}(a) = q_t^{\mathcal{I}}(a)\}$$

The agreement operator between features selects the individuals from the domains of the features that are mapping to the same individuals. We leverage such an approach in the definition of the policy to state *subject similarity* condition.

4. Access Control Procedure

Here, we present a formal approach to deal with an access request in a system using TDLBAC. Every access request is interpreted as a triple $\langle s_r, o_r, a_r \rangle$ where the first element is the subject, requesting the

access. The second element is the object, which is requested to be accessed, and the last element is the type of the requested access. In TDLBAC, three conditions must be satisfied for a policy from HPB to be applicable to a request:

1) The history based constraint of the policy should be satisfied against HB.
2) *Subject similarity* condition (if existed in the policy definition) should be checked as well.
3) The type of the three parameters in the request should be compatible with the type of the corresponding ones in the definition of the policy. Formally, if r is a typical request in form $\langle sc_r, oc_r, ac_r \rangle$, and p is a typical policy rule in form $\langle s_p, o_p, a_p \rangle$, this condition can be formulated as:

$$sc_r \sqsubseteq s_p \wedge oc_r \sqsubseteq o_p \wedge ac_r \sqsubseteq a_p$$

In other words, the parameters of the request should be subsumed by the policy parameters. Note that sc_r, oc_r, ac_r are the corresponding concepts of the request parameters.

We propose an algorithm to decide on an access request by checking the existence of a policy rule in HPB, applicable to the request with respect to these three conditions. The algorithm utilizes the inference services of $\mathcal{TL}$-$\mathcal{ALCF}$.

Example: Consider an example based on controlling an election-system in a typical country. The election-system is a type, named ES, in the hierarchy of objects in the TBox. It has instances in each city to be accessed by people through the web. Suppose that the system is used in a presidential election in the second round, and the policy is to limit the voters to the residents who have participated in the first round. *resident* is a defined class in the hierarchy of subjects. Some examples of action types in the system are $register$, $vote$, $result\text{-}checking$. The policy for voting can be defined as follows in HPB:

$$\begin{aligned} vote\text{-}policy\text{-}2nd\text{-}round \equiv \Diamond x(x\ b\ \#)(*PS : resident) \\ \sqcap (*PO : ES) \sqcap (*PA : vote) \\ \sqcap vote\text{-}1st\text{-}round@x \sqcap \\ (*PS \downarrow *AS) \end{aligned} \tag{3}$$

The $vote\text{-}1st\text{-}round$ concept is defined in the TBox as follows:

$$\begin{aligned} vote\text{-}1st\text{-}round \equiv & (*AS : resident) \\ & \sqcap (*AO : ES) \sqcap (*AA : vote) \end{aligned}$$

In the rest, at first, we discuss how to store the history of accesses, named HB, in the ABox. Then we describe the algorithm that determines whether a policy is applicable to a request, or not.

4.1. Access History Base (HB)

The procedure for storing the history in HB is described using the election-system example. For each request for voting in the first round, if it is granted by the access control system, the corresponding assertions for storing the access are inserted into the ABox. Suppose a resident, named $John\ King$, requests to vote in the first round of presidential election in an election-system instance named $election\text{-}sub20$. The request is analyzed by the access control system using the defined policies in HPB, and the request is granted. The next step is registering the access with its three parameters in HB. The noticeable point in storing the access is that we have *unique name assumption*, or *UNA* in short, for subjects in TDLBAC. In a logic with *UNA*, different names refer to different entities of the world. *UNA* is needed for verification of *subject similarity* condition. The assumption can be easily satisfied by inserting the facts related to subjects using a unique certificate, for instance, national identification number in the election example. Granting the request, the system generates a name by a mechanism to be assigned to the access, for example, $v\text{-}1st\text{-}101$, and the following facts are inserted into the ABox:

1) $vote\text{-}1st\text{-}round(v\text{-}1st\text{-}101)$
2) $*AS(v\text{-}1st\text{-}101, 12345)$
3) $*AO(v\text{-}1st\text{-}101, election\text{-}sub20)$
4) $*AA(v\text{-}1st\text{-}101, v1)$

Note that $v1$ is a typical instance of $vote$ action, released as an element of the request, and 12345 is the national identification number of $John\ King$.

Generally, if $ac1$ is the assigned access name, acc is the access type, and s_r , o_r , and a_r are subject, object, and action individual parameters of the granted request, the inserted facts to the ABox are as follows:

1) $acc(ac1)$
2) $*AS(ac1, s_r)$
3) $*AO(ac1, o_r)$
4) $*AA(ac1, a_r)$

Using the approach, each access is identified in the ABox with its three parameters.

4.2. Algorithm of Access Control

Let us follow the election-system example to simplify the description of our access control procedure. Suppose that $John\ King$ wants to participate in the second round of the presidential election. The object and action individuals are known in server side, since different accesses and objects to be accessed, are already defined in the ABox. However, all individuals who are the potential users of the system, may not be known by the access control system. The approach that can be used is employing credential-based identification of subject individuals. Each user introduces

himself/herself to the system using the credentials that he/she attaches to his/her request. The system can map the requester to the concepts in the hierarchy of subjects based on the received credentials. For example, $John\ King$ may be mapped to the *resident* concept, by one of his credentials, and to the *graduate* concept by another one. Note that the unique certificate to satisfy the *UNA*, should be one of the credentials. The corresponding assertions are inserted to the ABox to be used in access control procedure. For example, assuming 12345 as the national identification number of $John\ King$, the assertions $resident(12345)$, and $graduate(12345)$ are inserted to the ABox, before the access control procedure starts.

For a policy to be applicable to the request, its history constraint is evaluated first. This is accomplished by *instance retrieval* task, which refers to finding all instances of a particular concept [19]. For a policy to be checked for the request, the *instance retrieval* task is used to find the instances of the history constraint of the policy. For example, for the policy defined in (2), the task is to find the instances of the concept $vote\text{-}1st\text{-}round$ from HB, which is a part of the ABox. If no instance is retrieved, the policy is not applicable, and the process should be repeated for the next policy. Otherwise, the instances which are found for the history constraint concept, should be checked for the two other conditions; *subject similarity* and *type compatibility*. Note that the retrieved instances are the potential instances of the defined policy that might grant the request.

In the second step of the algorithm, three assertions are added to the ABox temporarily. If s_r , o_r , and a_r are the three individual parameters of the request, and $ac1$ is the instance retrieved for the history constraint concept, the process continues by adding the following assertions to the ABox temporarily:

1) $*PS(ac1, s_r)$
2) $*PO(ac1, o_r)$
3) $*PA(ac1, a_r)$

The next step is checking the truth value of $p(ac1, \#)$, where p is the concept related to the policy, and $\#$ is the current interval. Checking that $\langle ac1, \# \rangle$ is the instance of concept p is accomplished by *instance checking* inference service. *Instance checking* service verifies whether a given individual is an instance of (belongs to) a specified concept [7]. The three temporarily added assertions are removed from the ABox after the *instance-checking* procedure. It is obvious that if the procedure can validate $p(ac1, \#)$, subject similarity, as well as type compatibility conditions are also held.

Subject Similarity Condition: If the policy includes $(*PS \downarrow *AS)$ (i.e. *subject similarity* condition), and HB contains $*AS(ac1, s_a)$, the *instance-checking* service returns true, if $s_r \equiv s_a$. Note that, the instance of $*AS$ already exists, added to HB (or the ABox) after the access, and the instance of $*PS$ is inserted as one of the three temporarily added assertions with the requester as its parameter.

Type Compatibility Condition: The assertions related to the types of the requester (i.e. subject parameter) have been inserted to the ABox, based on his/her credentials. The types of objects, and actions are already included in the ABox. These assertions in addition to the three assertions added temporarily are used collectively to check the types of the three request parameters against the types specified in the policy definition. For example, in policy definition in (2), we have $(PS : resident)$, $(*PO : ES)$, and $(*PA : vote)$ clauses. The *instance-checking* returns true, if the types of the three individual parameters of the request are subsumed by the types specified in the policy definition. For example, $John\ King$ is allowed to vote based on policy in (2), if he is mapped to the *resident* concept, or any concept subsumed by it, regarding $(*PS : resident)$.

There are two conditions for termination of the access control procedure:

1) A policy passes through all the condition checking steps. The request is granted, and the corresponding assertions required for storing the access are inserted into HB.
2) No policy passes through the algorithm steps, and the request is rejected.

4.3. Access Control Optimization

The procedure requires $O(k.n)$ times of *instance checking* execution, where k is the size of HPB, and n is the size of HB. Since n is usually larger than k considerably, we propose an optimization step, called *pre-filtering*. The step reduces the policies for evaluation in the main procedure significantly.

Pre-filtering detects unusable policies based on type compatibility condition. Suppose a typical request denoted by $\langle s_r, o_r, a_r \rangle$. To employ type compatibility checking in order to filter the inapplicable policies, the individual parameters of the request should be mapped to the corresponding concepts in the hierarchy of subjects, objects, and actions. The most well-known approach to this end, is the *most-specific-concept* (*msc*). The *msc* of an individual *b* is a concept description that has *b* as an instance and is the least concept description (w.r.t. subsumption) with this property. Note that the *msc* of an individual is a set of concepts in the ontology. In this procedure, an individual is mapped to a concept description with map(a), where a is an individual and $map(a) = \bigsqcap_{c_i \in msc(a)} c_i$. The *msc* set of the object and action parameters of the request can be computed using the related assertions already defined

in the ABox. The *msc* set of the subject individual is achieved using the assertions inserted to the ABox based on the attached credentials.

Suppose that s_r, o_r, and a_r are mapped to the concepts sc_r, oc_r, and ac_r in the hierarchy of subjects, objects, and actions respectively. After mapping the individuals to the concepts, a concept named $Request\text{-}Permission$ is formulated for the request as follows:

$$Request\text{-}Permission \equiv (*PS : sc_r) \sqcap (*PO : oc_r) \sqcap (*PA : ac_r)$$

For performing the *pre-filtering* step for each policy, the satisfiability of the concept resulted from its conjunction with the $Request\text{-}Permission$ concept is checked. *Concept satisfiability* is one of the main inference services in DL. It specifies whether a concept definition can have individual(s) or not. The service is used here to check the policy applicability with respect to a specified request. Unsatisfiability of the conjunction shows the fact that one of the three types of the parameters in the request, returned by the *map* operator, can not have individuals of the corresponding type in the policy. To make it more clear, let us see a simple example based on the election-system example.

Suppose $Mary\ Green$ wants to participate in the election using an instance of ES. She attaches her credentials to her request, and the system defines her as an individual of $nonresident$ and $female$ concepts. Then, the corresponding assertions are inserted to the ABox. The *map* operator maps her to $nonresident \sqcap female$, for example. Therefore, the $Request\text{-}Permission$ concept for her request is formed as follows:

$$Request\text{-}Permission \equiv (*PS : nonresident \sqcap female) \sqcap (*PO : ES) \sqcap (*PA : vote)$$

To evaluate $vote\text{-}policy\text{-}2nd\text{-}round$ specified in (2) for the request released by $Mary\ Green$, the satisfiability of the following conjunction is checked:

$$\begin{aligned} &vote\text{-}policy\text{-}2nd\text{-}round \sqcap Request\text{-}Permission \equiv \\ &\Diamond x(x\ b\ \#)(*PS : resident) \sqcap (*PO : ES) \\ &\sqcap (*PA : vote) \sqcap vote\text{-}1st\text{-}round@x \\ &\sqcap (*PS : nonresident \sqcap female) \\ &\sqcap (*PO : ES) \sqcap (*PA : vote) \end{aligned}$$

Assuming that $resident \sqcap nonresident \equiv \bot$ can be inferred from the TBox, the two clauses, $(*PS : resident)$ and $(*PS : nonresident \sqcap female)$, makes the conjunction unsatisfiable.

However, the *pre-filtering* procedure is not sufficient for checking type consistency between a policy and a request. Consider the case in which the type returned by the *map* operator subsumes the corresponding type in policy. In this case, the *concept satisfiability* task for the related conjunction returns true, but *type compatibility* condition is not satisfied. Note that the type in the policy is more specific than the type in the request. Moreover, the type in the request is computed by the *map* operator, and is the most specific concept according to the knowledge-base.

To clarify the case, let us describe it using a simple example in the election-system. Suppose a person who requests for voting in the election, and he/she attaches no credential to specify his/her nationality. Therefore, the system maps the subject to a more general concept, for example the *people* concept. The conjunction computed for the request is satisfiable, since it is possible for an individual to be of types, *people* and *resident* at the same type. Therefore, if the concept of the request parameter is above the concept of the policy parameter in the hierarchy, the task returns true, in spite of the policy inapplicability. We refer to these policies as *unfiltered policy rules*. They will be filtered out in the access control procedure.

The proposed optimization method acts as a filter for policies to separate the ones which are consistent with respect to the types of the request parameters, although the result set may be larger than the real one. The step can reduce the set of policies for checking in the access control procedure considerably.

5. Time Complexity Discussion

In this section, we assume the size of HPB as k, and the size of HB as n. The two most time-consuming steps of the algorithm in the main procedure are:

- *Instance Retrieval*
- *Instance Checking*

The optimization method adds the two following tasks:

- *Most Specific Concept*
- *Concept Satisfiability*

First, we discuss about time complexity of inference services in the employed logic. All inference services for DL can be reduced to *consistency checking* task for ABox [7]. *ABox consistency* is proved to be PSPACE-complete in $\mathcal{ALCF}$ [20]. Therefore, all the inference services in $\mathcal{ALCF}$ are PSPACE-complete as well. The *msc* operator in the proposed access control algorithm is performed on the concepts of subjects, objects, and actions hierarchies. As mentioned in Section 3, the concepts in these three hierarchies are defined based on $\mathcal{ALCF}$. Therefore, *most specific concept* is PSPACE-complete in our algorithm. *Concept satisfiability* is proved to be NEXPTIME-HARD in $\mathcal{TL}$-$\mathcal{ALCF}$ with a simple or acyclic TBox [21]. However, the temporal part of the concept resulted from the conjunction can be eliminated for the *Concept satisfiability* task. Therefore, this task can also be considered to be PSPACE-complete. The time complexity of *instance retrieval*

and *instance checking* tasks in $\mathcal{TL}$-$\mathcal{ALCF}$ are not investigated yet. However, it seems that in our proposed algorithm, these two tasks are not considerably effected by the temporal part of $\mathcal{TL}$-$\mathcal{ALCF}$. It can be intuitively concluded that the time complexity of these two tasks in our algorithm does not exceed their time complexity in $\mathcal{ALCF}$, or PSPACE-complete.

The *instance retrieval* task, in the worst case, should be performed for all policies in HPB (i.e. k times). Moreover, for each retrieved instance, we should run the *instance checking* task as well. As a result, in the worst case, the number of *instance checking* task execution has the order of $O(k.n)$. Obviously, after optimization, the number of *instance retrieval* tasks reduces from k to k', where k' is the number of policies that pass through the *pre-filtering* step. Hence, the number of *instance checking* task execution reduces to $O(k'.n)$. Therefore, although the *pre-filtering* step adds three *msc* operator execution, as well as k *concept satisfiability* checking, it reduces the overall execution time of the algorithm considerably.

The noticeable point is that we have investigated just an upper bound for the access control algorithm based on the general inference services for DLs. Of course, in practice the algorithm can be implemented more efficiently. The proof of the algorithm decidability was more important to us. In future, algorithms with better time complexity may be proposed for some fragments of $\mathcal{TL}$-$\mathcal{ALCF}$ that are enough for our proposed approach, and good reasoners for working on the knowledge-bases of this logic may be developed.

6. Conclusion

In this paper, we proposed an access control model based on a temporal extension of description logics. The employed logic allows security authorities to define policy rules at the concept level. Moreover, they can define constrains based on users' accesses in the past for the policy rules. This can be helpful in different applications such as e-banking environments. The model components, the algorithm of access control, as well as the suggested optimization method were described, and a brief discussion on the time complexity of the algorithm was presented.

References

[1] S. Javanmardi, M. Amini, R. Jalili, and Y. Ganjisaffar, "SBAC: Semantic Based Access Control", in *Proceedings of the 11th Nordic Workshop on Secure IT-systems, Linkping, Sweden*, 2006, pp. 157–168.

[2] A.N. Ravari, M. Amini, and R. Jalili, "A Semantic Aware Access Control Model with Real Time Constraints on History of Accesses", *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, pp. 827–836, 2008.

[3] Z. Iranmanesh, M. Amini, and R. Jalili, "A Logic for Multi-domain Authorization Considering Administrators", in *IEEE Workshop on Policies for Distributed Systems and Networks, 2008. POLICY 2008*, 2008, pp. 189–196.

[4] G. Kołaczek, "Application of Deontic Logic in Role–Based Access Control", *Int. J. Appl. Math. Comput. Sci*, vol. 12, no. 2, pp. 269–275, 2002.

[5] J. Chae, "Towards Modal Logic Formalization of Role-Based Access Control with Object Classes", *Lecture Notes in Computer Science*, vol. 4574, pp. 97, 2007.

[6] F. Baader, I. Horrocks, and U. Sattler, "Description logics as ontology languages for the semantic web", *Festschrift in honor of Jorg Siekmann, Lecture Notes in Artificial Intelligence. Springer-Verlag*, 2003.

[7] F. Baader, *The description logic handbook: theory, implementation, and applications*, Cambridge University Press, 2003.

[8] A. Artale and E. Franconi, "A survey of temporal extensions of description logics", *Annals of Mathematics and Artificial Intelligence*, vol. 30, no. 1, pp. 171–210, 2000.

[9] A. Artale and E. Franconi, "A temporal description logic for reasoning about actions and plans", *Journal of Artificial Intelligence Research*, vol. 9, no. 2, pp. 463–506, 1998.

[10] A. Artale and E. Franconi, "Representing a robotic domain using temporal description logics", *AI EDAM*, vol. 13, no. 02, pp. 105–117, 1999.

[11] F. Cuppens and R. Demolombe, "A deontic logic for reasoning about confidentiality", *Deontic Logic, Agency and Normative Systems, Workshops in Computing. Springer*, 1996.

[12] E. Bertino, "Temporal authorization bases: From specification to integration", *Journal of Computer Security*, vol. 8, no. 4, pp. 309–353, 2000.

[13] E. Bertino, P.A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model", *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 191–233, 2001.

[14] L. Qin and V. Atluri, "Concept-level access control for the semantic web", in *Proceedings of the 2003 ACM workshop on XML security*. ACM New York, NY, USA, 2003, pp. 94–103.

[15] D.F.C. Brewer and M.J. Nash, "The Chinese wall security policy", in *Proceedings of the 1989 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1989, pp. 206–214.

[16] P. Dias, C. Ribeiro, and P. Ferreira, "Enforcing history-based security policies in mobile agent systems", in *IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003*, 2003, pp. 231–234.

[17] G. Edjlali, A. Acharya, and V. Chaudhary, "History-based access control for mobile code", in *Proceedings of the 5th ACM Conference on Computer and Communications Security*. ACM New York, NY, USA, 1998, pp. 38–48.

[18] J.F. Allen, "Temporal reasoning and planning", *Morgan-Kaufmann Series In Representation And Reasoning*, pp. 1–67, 1991.

[19] L. Serafini and A. Tamilin, "Distributed instance retrieval in heterogeneous ontologies", in *Proceedings of SWAP 2005, CEUR Workshop Vol*, 2005, vol. 166.

[20] C. Lutz, "PSpace reasoning with the description logic ALCF (D)", *Subscription Information*, p. 535.

[21] A. Artale and C. Lutz, "A correspondence between temporal description logics", *JOURNAL OF APPLIED NONCLASSICAL LOGICS*, vol. 14, no. 1/2, pp. 209–233, 2004.