A Logic-Based Authorization Framework and Implementation

Mingsheng Zhang¹, Xinqiang Ma², and Mingyi Zhang³ †

¹ College of Economics & Management Guizhou University for Nationalities, Guiyang, P.R China
Email: gyzhangms@126.com
² Chongqing University of Arts and Sciences, Chongqing, P.R China
Email: mxq345@sohu.com
³ Dept of Mathematics Guizhou Academy of Sciences, Guiyang, P.R China
Email: zhangmingyi045@yahoo.com.cn

Abstract—In access control, it is a reasonable requirement that authorization mechanism can be implemented intelligently by logic programs. We propose an authorization framework based on logic programs, ranging from design, analysis, and implementation. Our proposed framework is powerful and useful, with RBAC features, flexible authorization, logic-based formalization and integration of policies. It can specify complex security requirements in real world, and facilitate many advantage aspects held by the some prevailing authorization theories.

Index Terms—authorization framework, logic program, flexible authorization, implementation

I. INTRODUCTION

Over the years, many researchers have proposed a variety of access control policies, models and mechanisms. But there is no any approach that can realize all security requirements, and every approach only fits to a special situation. In fact, the definition of an access control policy /model is far from being a trivial process. The most major difficulty lies in the interpretation of real-world security policies (often complex and sometimes ambiguous) and their translation in well defined and unambiguous rules, which are easily implemented by a computer system [1]. Hence, it is a feasible means to realize complex security requirements via integrating policies in a way.

Designing a flexible authorization framework based on logic programs is our main task. There are the several literatures’ achievements affecting deeply our study motivation in this field. Woo and Lam [2] proposed that a number of different policies can be represented by default logic. It is an abstract and expressible approach. E. Bertino et al [3] proposed a logic formalism (based on order logic programs) that naturally support the complex security specifications. The formalism depends on a hierarchically structured domain made of subjects, objects and privileges. Jajodia et al [4] worked on a proposal for a logic specification allowing the representation of different policies in an expressively powerful, clear semantics, bearable data complexity and flexible manner.

It is well known that the driving motivation of RBAC model is to simplify security policy administration while facilitating the definition of flexible customized policies. Nowadays, the number of practical access control systems that offer RBAC features is coming to be expected and growing rapidly in large organizations [5]. Sandhu et al. [6] informally described the key features of RBAC models, which are summarized: users acquire permissions via roles; many-to-many user-role assignment; many-to-many permission-role assignment; simultaneous use of permissions on multiple roles; role hierarchy; separation of duty; session and constraints. Barker and Stuckey [7] showed how a range of RBAC models can be represented as constraint logic programs in terms of the functional capabilities of RBAC. They extended the standard RBAC, and enable security administrators to define a range of access polices with the features like denials of access and temporal authorizations. These features are often useful in practice, but are not widely supported in existing access control models. It is more important that representing access policies as constraint logic programs makes it possible to support constraint checks and administrator queries, and also enables access requests and constraint checks to be efficiently evaluated. Therefore, we think that RBAC model is valuable for exploring on implementing flexible access control policies.

Moreover, there is the public topic how to combine the advantages of the authorization models FAF and HTAM [8]. In HTAM, a general solution to inheritance and overriding is given by means of non-stratifiable programs, while it is impossible in FAF to override an inherited authorization with a derived authorization. On the other hand, the conflict resolution and decision policies are fixed as the denials-take-precedence principle in HTAM, while multiple policies are supported in FAF. So far as we know, there is no study on this aspect specially by using logic programs. Thus we try to study the topic.

Based on the above statement, we will propose an authorization framework based on logic programs, ranging from design, analysis, and implementation. The rest of this paper is arranged as follows: in section 2, we describe the specifications of our authorization framework, ranging from architecture, PRAP module, PRAP module and UR-RP Authorization Policy in the framework respectively; the implementations of the

---

¹ Corresponding Author

© 2009 ACADEMY PUBLISHER
AP-PROC-CS-09CN003 322
framework are stated in section 3; some relevant works are summarized in section 4; at last, the conclusion is discussed.

II. AUTHORIZATION FRAMEWORK SPECIFICATION

A. Architecture

The architecture of our authorization framework is shown in Figure 1. There are the three main functional modules: URAP module, PRAP module, and UR-RP Authorization Policy, which are composed by the sets of logic programming rules.

--URAP module is constructed by stratified logic programs for assigning roles to users.

--PRAP module consists of ordered logic programs for assigning privileges to roles.

--UR-RP Authorization Policy enforces multi-policies by combining URAP module and PRAP module.

We should note that there is the data system supporting the three basic modules. The three basic modules constitute a policy program which materializes authorization requirements. The policy program determines authorizations in terms of its semantics. The semantics of the policy program is based on answer sets, but is obtained by several computations. That is, our authorization framework can be supported by powerful computing tools. When a user make a request for performing a particular access r on an object o, that is Request (u, p, o), Permitted(u, p, o) are checked whether it belongs the answer set of the policy program. If the result of the checking is “Yes”, the request Request (u, p, o) is granted, and denied otherwise. The completeness of our authorization framework is satisfied by using the closed policy or open closed policy.

Similar to other authorization frameworks, the data system of our authorization model refers to four basic elements: Subject (S), Object (O), Role (R), and Privilege (P). The three basic modules will be described in the following.

Our architecture is a flexible authorization framework based on logic programming. It has main advantages of RBAC model, FAF model, and ordered logic programs. The main advantages will be clearly shown through the following statements.

B. URAP module

(1) Syntax of URAP

URAP denote User Role Assignment Program. URAP module works for assigning roles to users. URAP includes some basic predicate expressions and auxiliary predicate expressions.

Definition 1(Basic Predicate Expressions)

--can-ura(u,r,g) The predicate expresses explicitly the specified role-user assignment authorization. For instance, can-ura(u, r, g) specifies that the grantor(i.e. an administrative role in our framework) has assigned the role to the user u.

--der-ura(u,r,g) The predicate specifies how to obtain new derived role-user assignment authorizations. These derived role-user assignment authorizations not only are obtained according to users hierarchies, but also come from other forms of derivations (authorizations of other users).

--do-ura(u,r,g) The predicate acquires role-user assignment authorizations by solving possible conflicts between role-user assignment authorizations (explicit and/or derived).

--ura(u,r,g) The predicate represents dynamic role-user assignment authorization specially designed for dynamic separation of duty.

Besides above three basic predicate components, there are some auxiliary predicates as follows:

Definition 2(Auxiliary Predicate Expressions)

--inh The predicate expresses inherited relations between users.

--dirinh The predicate expresses directly inherited relations between users.

--overUG The predicate is used in propagation of role-user assignment authorizations. It expresses overriding.

--rel The relationship predicates are defined in systems for acquirements.

The assignment of URAP is divided into four decision stages:

(a) Explicit Role-User Assignment Authorization

Any access policy has original authorization specified by an administrator. Here we use the rules with predicate can-ura(u, r, g) represent explicit role-user assignment authorization. The kind of rules means that in the case of what conditions a user u is assigned a role r via a grantor (an administrative role).

Definition 3 (Explicit Role-User Assignment Authorization)

An explicit role-user assignment authorization rule is the form:

\[
\text{can-ura}(u, r, g) \leftarrow A_1, \ldots, A_n, \text{not } B_1, \ldots, B_m
\]

where, u, r, and g are elements of U, Rgra, and Radm respectively; A_i and B_i are inh, dirinh, and rel predicates.

Example! the set of URAP rules with explicit assignment authorization:

can-ura(John, Research-staff, adm) ←

~can-ura(Jam, Research-staff, adm) ←
can-ura(John, Citizens, adm) ← inh(John, Adm-staff, ̸≤UG)
can-ura(Bell, Chair, adm) ← not can-ura (Bob, Chair, adm)

(b) Rules for Propagation Policies
In our framework, there are common propagation policies that are similar to other definitions, but the specific free propagation is our definition used specially.
Definition 4 (Specific Free Propagation)
The positive (resp. negative) role-user assignment authorizations are propagated down (resp. up) along a UG regardless of the presence of other contradicting authorizations.

\[
\neg \text{der-ura}(u, r, g) \leftarrow \neg \text{can-ura}(u, r, g), \text{inh}(u, u', \leq_{\text{UG}})
\]

\[
\text{der-ura}(u', r, g) \leftarrow \text{can-ura}(u, r, g), \text{inh}(u, u', \leq_{\text{UG}})
\]

(c) Rules for Conflict Revolution Policies
There are selectable conflict resolution policies for the role-user assignment authorizations, which are similar to the definitions in other references, but we should note that several points in the following:
--Some conflict resolutions may be no complete. For example, in the case of “Denials Take Precedence”, if there is no any authorization how we answer the request. This needs the decision policies: closed policy and open policy or other approaches.
--In “Most Specific Takes Precedence”, we assume that “Most Specific Overrides” propagation is adopted, and “most specific”(i.e. “prevail”) is regulated according to the hierarchy ̸≤UG relationship (i.e. lower users prevail higher ones).
--Moreover, we can introduce grantor priority for solving conflicts. For instance, the following rules may express a new strategy with grantor priority for Denials Take Precedence conflict policy:

\[
\neg \text{do-ura}(u, r, g) \leftarrow \neg \text{der-ura}(u, r, g), \neg \text{der-ura}(u, r, g'), \text{inh}(g, g', \leq R)
\]

\[
\text{do-ura}(u, r, g) \leftarrow \text{der-ura}(u, r, g), \neg \text{der-ura}(u, r, g'), \text{inh}(g, g', \leq R)
\]

\[
\neg \text{do-ura}(u, r, g') \leftarrow \neg \text{der-ura}(u, r, g), \neg \text{der-ura}(u, r, g'), \text{inh}(g', g, \leq R)
\]

\[
\text{do-ura}(u, r, g') \leftarrow \text{der-ura}(u, r, g), \neg \text{der-ura}(u, r, g'), \text{inh}(g', g, \leq R)
\]

d) As conflict resolution is a more complex matter, it does not usually have a unique answer [11, 12]. Different conflict decisions could be used, and they are applicable in specific situations.
--The rule definitions of the conflicts resolution above are different from ones in FAF [4]. Certainly, we need to modify a little semantics for the classical logic programs.
(d) Dynamic Role-User Assignment Authorizations
Similar to the concept of a user session [6], we introduce the form of rules as follows:

\[
\text{ura}(u, r, g) \leftarrow \text{do-ura}(u, r, g), \text{activate}(u, r)
\]

"activate(u,r)" in this rule expresses that a user activate role r on the basis of “do-ura(u, r, g)” when it logs on the system. “activate(u,r)” can be viewed as a fact rule added into the URAP module program by means of some methods, such as GUI. Therefore ura(u, r, g) can represent dynamic role-user assignment authorizations.

(2) Semantics of URAP

A URAP program is composed of the four rules, which are Explicit Role-User Assignment Authorization with can-ura, Rules for Propagation Policies with deri-ura, Rules for Conflict Revolution Policies with do-ura, and Dynamic Role-User Assignment Authorizations with ura. We can obviously know the program is stratified (i.e. the dependency graph of the program is no cyclic for negation as failure). Therefore the URAP programs are stratified ones.

According to the relevant results in [4] we can obtain that: A URAP program has a unique stable model (normal version of logic programs); the unique stable model can be computed in quadratic time data complexity; the stable model is coincided with well founded model semantics of normal version of logic program.

In addition, our structure of URAP programs is in some aspects different from [4]. We can summarize that:
--We introduce dynamic role-user assignment authorization rule with ura for the feature of BRAC model;
--There are no decision policies in conflicts resolution, and we put decision policies into UR-RP module;
--We have introduced other conflicts resolution approaches, such as priority level and preference. How to evaluate these is considered in next work.
--We introduced Specific-Free-Propagation propagation for implementing BRAC mechanisms.

C. PRAP module

(1) Syntax of PRAP
PRAP means intuitively Privilege Role Assignment Program. The program of PRAP module takes charge of assigning privileges to roles. Here, our syntax definitions of PRAP are similar to the reference [3].

(2) Semantics of PRAP
In the following, we give the outline of formalization and evaluation of PRAP program:

In Figure 2, it is shown how the syntax and semantics of a PRAP program are specified. There are four steps: firstly, the components are original; the knowledge base comes from the propagation of rules in the components according to the hierarchies related to role, privilege and object; the reduction is obtained by conflict resolution; at last, the reduction program produces stable model semantics as the semantics of the whole PRAP program.

It is also seen that a PRAP program P has two stable model semantics corresponding to the reductions of \(G_{\text{sem}}^u(P)\) and \(G_{\text{sem}}^m(P)\), respectively(see [3]). The former can induce the called well-founded semantics, while the latter stable one. Here, our semantics definitions are different from the reference[3], but they are more
intuitively and clear. On the other hand, our well-founded semantics can be no uniquely, as rules in $G_{well}(P)$ are not certainly stratified.

For stratified well-founded model semantics, we can deal simply with answering a request on privilege role assignment. If a positive (resp. negative) assignment authorization is in the well-founded model, the request is permitted (resp. denied). If there is no authorization according to the request, deciding whether to grant or deny the request depends on the decision policies of the UR-RP module.

For stable model semantics, the resulting $G_{st}(P)$ is adopted “most specific takes precedence” conflict resolution that is not complete. Multiple consistent stable models are produced. Therefore we use also naturally two approaches: pessimistic reasoning and optimistic reasoning. The pessimistic semantics means that only the request that is satisfied in every assignment authorization sets can be taken into account. Whereas the optimistic semantics specify that the request can be answered certainly if it is satisfied in one of assignment authorization sets.

D. UR-RP Authorization Policy

In this section, we specify how to whether to decide to grant an access request based on URAP and PRAP programs. Firstly, some related concepts are given below.

(1) Relevant Concepts

Definition 5(Access Request)

An access request is defined as a triple $(u, p, o)$, where $u \in U$, $p \in P$, and $o \in O$. For simplicity, we also denote an access request $(u, p, o)$ as the predicate $Req(u, p, o)$.

An access request $(u, p, o)$ means that the user $u$ requests the action $p$ to the object $o$. For an access request $(u, p, o)$, whether our system framework grants or denies the request depends on the authorization rules in the system. Therefore, we need to define the following some concepts.

Definition 6(Authorization Predicate)

Predicate $Permitted(u, p, o)$ specifies that a user $u$ has the privilege $p$ access to object $o$ for an access request $(u, p, o)$, i.e. $Req(u, p, o)$ is satisfied by $Permitted(u, p, o)$.

Definition 7(Session,)

Session, in our work represents the set of dynamic role-user assignment authorizations:

$Session_\alpha = \{\ura(u, r, g)|u \in U, r \in R, g \in G\}$

We should note that:

--$Session_\alpha$ expresses that a user $u$ holds the activated roles appearing in a rule $ura(u, r, g)$ in Session;

--$Session_\alpha$ is different from the relevant notation in Session [5,6]. $Session_\alpha$ is a set, while session is a mapping.

--A $Session_\alpha$ is maintained and managed in the implementations. We can assume that a when user $u$ logs on the system the $Session_\alpha$ is built; a $ura(u, r, g)$ is added into the $Session_\alpha$ when the user activate a role $r$; when the the $Session_\alpha$ is automatically deactivated user logs off the system. while the is $Session_\alpha$ active these $ura(u, r, g)$ can be appropriately added and deleted.

Definition 8($\Gamma_\alpha$)

The set $\Gamma_\alpha = \{(o, r).ura(p, r, g)|(o, r).ura(p, r, g) \in A\}$ is a PRAP program $P$ stable model.

Definition 9(Authorization Rule)

The authorization rules are defined according to the form of rules:

$\text{Permitted}(u, p, o) \leftarrow \ura(u, r, g), (o, r).pra(p, g')$

$\neg \text{Permitted}(u, p, o) \leftarrow \text{not} \ura(u, r, g)$

$\neg \text{Permitted}(u, p, o) \leftarrow (o, r) \neg \pra(p, g)$

(2) Authorization Policies

In an authorization rule, $Permitted(u, p, o)$ depends on $Session_\alpha$, $\Gamma_\alpha$ and decision policies in UR-RP. Therefore, we can divide authorization policies into the following classes.

Definition 10(Authorization Policy)

An authorization policy can be defined as a pair $(\text{pro}, \text{con}, \text{sem}, \text{dec})$, where

$\text{--pro} \in \text{PROPAGATION} = \{\text{No Propagation}, \text{No Overriding}, \text{Most Specific Overrides, Path Overrides, Specific Free Propagation}\}$;

$\text{--con} \in \text{CONFLICT-RESOLUTION} = \{\text{No Conflict, Denials Take Precedence, Permission Take Precedence, Nothing Take Precedence, Most Specific Takes Precedence, Most Specific Along A Path Takes Precedence, Priority Level}\}$;

$\text{--sem} \in \text{SEMANTICS} = \{Wf, \text{Pess}, \text{Opti}\}$, where $Wf$: Well-Founded semantics, $\text{Pess}$: Pessimistic semantics, $\text{Opti}$: Optimistic semantics;

$\text{--dec} \in \text{DECISION} = \{\text{closed policy, open policy}\}$.

III. IMPLEMENTATION

On the basis of statements above, we can obtain the algorithm for our framework program as follows:

INPUT: (1) An access request $\alpha = (u, p, o)$

(2) An authorization program $P$

OUTPUT: (1) AUTHORIZAITON, if the request $\alpha = (u, p, o)$ is satisfied,

(2) REJECT, otherwise.

METHOD:

if $\exists u \ura(u, r, g) \in \text{SESSION}$ then

If Pess semantics then

If $(o, r).pra(p, g') \in \forall \text{ Pess-Auth-Set}$ then

return AUTHORIZAITON

else

return REJECT

endif

else * Opti or Wf semantics*

Let Current-Access be the set of current accesses

If $(o, r).pra(p, g') \in \text{Current-Auth-Set}$ then

Current-Access : = Current-Access $\cup \{\alpha\}$

return AUTHORIZAITON

else

if $\exists a \text{ Auth-Set A s.t. Current-Access} \cup \{\alpha\}$ is satisfied in $\text{A}$ then

Current-Auth-Set : = $\text{A}$

Current-Access : = Current-Access $\cup \{\alpha\}$

return AUTHORIZAITON

endif

325
return REJECT
endif
else
    return REJECT
endif

About the complexity of our authorization framework, we make a simple statement: since URAP module programs are stratified, the programs have unique stable model semantics and quadratic time complexity. A UR-RP module program is also stratified. We note that in logic programming, under data complexity, computing the well-founded semantics is polynomial complexity, while certainty and possibility reasoning are CoNP-complete and NP-complete respectively[9]. Therefore, if we select well-founded model semantics in a PRAP module program, the complexity of our framework is quadratic time complexity, while the others are CoNP-complete or NP-complete. However we have many quickly computing tools to support [10] so that there are possible schemas to solve some special complex problems.

IV. RELEVANT WORK

In the above statements, the comparisons of some relevant works were done. To sum up, it is more important that our proposed framework is powerful and flexible on functionality. With comparison with other approaches the authorization framework has the following advantages:
--It can enforce the function of “denial-role assignments have precedence” conflict resolution strategy (resp. “denial-role assignments have precedence” conflict resolution strategy) in [7] through combining “Specific Free Propagation” and denials take precedence (resp. “permissions take precedence”) strategy.
--It can implement SSD (Static Separation of Duty) and DSD (Dynamic Separation of Duty) mechanism in RBAC via the URAP program.
--Because of introducing order logic program in PRAP, it can deal granularly with authorizations on objects. General RBAC models don’t support it.
--As we adopt in UR-PR the authorization mechanism of RBAC, the framework is organizational.
--Since grantors must not a central administrator role, it should be easily extended into the applications of distributed systems.
--It is certainly flexible and powerful for use the advantages of the frameworks in [3, 4].

V. CONCLUSION

Our proposed framework is powerful and flexible on the basis of logic programs. It can facilitate many advantage aspects held by other approaches, such as [7], [3] and [4]. As the construct feature is in reason it is extended and refined. In our future work, we prepare to study authorization decisions with preferences based on the framework for realizing flexible access control policies.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (NSFC) under grant No.90718009, the Stadholder Foundation of Guizhou Province (Grant No.2009-111), and the Science and Technology Funds of Guizhou Province(Grant No.[2009]2125).

REFERENCES