日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

制御構造を導入したアクセス制御ポリシー記述言語の提案

田原 聖悟　長谷部 浩二　加藤 和彦

これまでアクセス制御のポリシー記述言語が数多く提案されており，その例として数理論理学を基にした言語や XACML などが挙げられる．これらの言語では，ポリシーは論理式などで記述したルールの集合として表現されるが，ポリシーが複雑な条件を伴う場合には，単純なルールの集合のみによる表現では記述が煩雑になるなどの問題があると考えられる．そこで本研究では，既存の論理言語を用いたポリシー記述言語に制御構造を導入し，複雑な条件を簡潔に表現し得る言語を提案する．さらに，この言語により記述されたポリシーからアクセス可否を判定するアルゴリズムを示す．

1　はじめに

アクセス制御は，システムのリソースに対してアクセスを行う能動的な主体が，そのリソースに対する可能な操作を管理する技術であり，コンピュータセキュリティを保つための重要な技術の一つである．このアクセス制御において，各主体に対する許可・禁止を決定する方針をポリシーと呼ぶ．今日，アクセス制御は多くのコンピュータシステムにおいて実装されているが，その制御方法が基にしているモデルはシステムにより異なり，例えば Access Control List（ACL）[11] やグループなどの概念を使って記述されるものや，組織の構成に着目した Role-based Access Control（RBAC）[12] などの様々なアクセス制御のモデルが存在する．また，RBAC を用いてポリシーを記述する SELinux [13] という Linux のモジュールなどが開発されている．

アクセス制御のポリシーを記述する言語には，可読性の高さ，記述の容易さ，さらには目的とするポリシーを記述するために必要な表現力などが求められる．また，その言語によって記述されたポリシーは，解釈の多様性や暗黙の了解のような曖昧さを排したものでなければならない．これらの条件を満たすために，近年，数理論理学の言語を基にしたアクセス制御のポリシー記述言語の提案が数多くなされている．こうした研究における基本的なアイデアは，まずポリシーを論理式により記述し，また，そのポリシーを論理推論の公理とみなすことで，アクセスクエリ（すなわち,「あるサブジェクトがあるオブジェクトに対してあるアクセスをすることが可能であるか」を問うクエリ）が与えられた際に，アクセスの許可・禁止の判定を論理推論により実現するというものである．このような言語の代表例としては，Halpern らによる [2] が挙げられる．この研究では，論理言語として基本的な表現力を持つ一階述語論理の部分体系を基に，論理式で記述したルールの集合によってポリシーを表現する方法が示されている．例えば,「課長というロールが割当てられた全てのユーザはファイル 1 に書き込みができる」というルールは，

$$\forall x(Manager(x) \Rightarrow may\_access(x, file1, write))$$

のような論理式で記述することができる．また，この論理体系を用いると，アクセスクエリが実時間で決定

A Policy Specification Language with Control Flow for Access Control

Shogo Tahara, Koji Hasebe, Kazuhiko Kato, 筑波大学大学院システム情報工学研究科, Graduate School of Systems and Information Engineering, University of Tsukuba.

可能であることも示されている．

しかしながら，このような形の論理式を列挙することによってポリシーを記述する方法では，時刻などのコンテクストの変化に合わせて異なるポリシーを柔軟に適用しようとした場合，ポリシーの記述が煩雑になるという問題が生じる．例えば，ポリシーの中であるパーミッションが時刻に合わせて異なるユーザに与えられる場合を考える．このとき，全ての時刻についてのユーザとパーミッションを列挙すると，同じパーミッションを何度も記述する必要があるためポリシーが煩雑になる．また，別の例として，あるユーザ A について 21:00 以降を除いてパーミッションが与えられている場合を考える．このとき，パーミッションが与えられている全ての時刻を列挙すると，ポリシーが煩雑となる．あるいは，この A に対してパーミッションが全ての時刻において与えられ，例外として 21:00 以降はパーミッションが与えられない，とポリシーを記述することもできるが，こうした記述は論理式の列挙だけでは表現できない．

そこで本研究では，既存の論理言語を用いたポリシー記述言語に制御構造を導入し，複雑な条件を簡潔に表現し得る形式言語を提案する．さらに，この言語により記述されたポリシーからアクセス可否を判定するアルゴリズムを与える．

本論文の概要は以下の通りである．第 2 章では本研究の関連研究について述べる．次に，第 3 章で提案するポリシー記述言語について，特に文法とアルゴリズムについて説明する．さらに，第 4 章では提案するポリシー記述言語を用いたポリシー記述例を紹介し，最後に第 5 章で結論と今後の課題について述べる．

2 関連研究

論理言語を基にしたポリシー記述言語は，Abadi らによる研究 [1] によって初めて提案された．この研究では，分散システムにおけるポリシー記述言語として様相論理を基にし，「誰が主張しているのか」という内容を記述するという方法が示されている．これ以降，様々な論理言語によりポリシーを簡潔に表現する試みが数多くなされて来た．

例えば，先に紹介した Halpern らの研究 [2] では，一階述語論理を基にした言語を提案している．この研究では，ポリシー記述言語として一階述語論理の部分体系を基にすることで，一般的なアクセス制御におけるポリシーを十分に表現できることが示されている．また，アクセスクエリの判定を論理式の証明可能性の決定問題に帰着できることも示されている．さらに，論理学の知識を有さないユーザであってもポリシーを容易に記述できるようにするために，ポリシー記述のための論理記号を含まないインタフェースを実装している．

Benferhat ら [3] は，一階述語論理を基にした言語により記述されたポリシーについて，一般的にどのような記述をするときにポリシーに矛盾が生じるかをいくつかのパターンによって分類し，また矛盾を含むポリシーから矛盾を含まないポリシーを生成するためのアルゴリズムを提案している．

Jajodia ら [4] は，特定のシステムについてのポリシーを記述するのではなく，複数のシステムについて記述するための柔軟なポリシー記述言語を提案している．この言語では，1 つのポリシーの中に様々な記述方法（肯定文の論理式のみにより記述する，否定文の論理式のみにより記述する，肯定文と否定文の両方を含む記述をする，矛盾を含む記述はできない，矛盾が起きた場合は肯定文を優先させる，矛盾が起きた場合は否定文を優先させる，等）が提案されている．これにより，システム毎に記述し易い方法で柔軟にポリシーを記述することができると主張している．

また，近年 XACML [14] という XML 形式のポリシー記述言語が提案されている．これは，セキュリティに関する情報を交換するための言語仕様である SAML を基にした言語である．この XACML や論理言語を基にした既存のポリシー記述言語の共通点として，アクセス権とその権限が付与される条件を全て列挙することでポリシーを記述する点が挙げられる．

これらの研究に対し，本研究では条件の組み合わせを制御構造で記述する事によりアクセス権の付与に対して複雑な条件を伴うポリシーを簡潔に記述する言語を目指している．

3 制御構造を導入したポリシー記述言語

3.1 基本的なアイデア

先に紹介した Halpern らの研究 [2] では，一階述語論理の部分体系によりポリシーを表現していた．しかしながら，アクセス権が付与される条件がコンテクストなどに応じて複雑に変化するポリシーを記述する場合，その記述は一般的に煩雑となる．例えば，時刻毎にポリシーが変化する場合には以下のような記述が必要となる．

$\forall x(time < 12:00 \wedge Manager(x)$
$\Rightarrow may\_access(x, file1, read))$
$\forall x(12:00 \leq time < 17:00 \wedge Manager(x)$
$\Rightarrow may\_access(x, file2, read))$
$\forall x(time \geq 17:00 \wedge Manager(x)$
$\Rightarrow may\_access(x, file1, read))$

この記述では，特に $Manager$ や may_access という記述を何度もする必要があるため，ポリシーが煩雑となる．

そこで，本研究では 2 つのアイデアを導入し，ポリシーを簡潔に記述する事を目指す．1 つ目は，ルールをまとめて簡潔に記述する方法である．これは，空欄を含むルール（例えば，「課長はファイル○を read 可能である.」のようなルール）を用意し，条件に応じてその空欄に適切な要素を入れる（例えば，ルールの○には 12:00 までは 1 を，12:00〜17:00 は 2 を，17:00 以降は 1 を入れる）表現方法である．2 つ目は，ルールの集合をまとめて簡潔に記述する方法である．これは，ポリシー全体に共通するルールの集合（例えば，「課長は常にファイル 1 を read 可能である.」）を用意し，条件毎に差分のルールの集合（例えば，「12:00〜17:00 の時，課長はファイル 1 を read できなくなり，ファイル 2 を read 可能となる.」）を足したり引いたりする表現方法である．

そのために，本研究で提案するポリシー記述言語では，Halpern らの一階述語論理を基にしたポリシー記述言語に新たにメタ言語を導入する．このメタ言語の導入により，条件に応じて異なるポリシーを適用することを可能とし，簡潔なポリシー記述を目指す．

なお，以下において新たに導入するメタ言語をローマン体で記述することにより，一階述語論理を基にした言語との区別を計ることとする．

3.2 提案言語の文法

提案するポリシー記述言語の文法を図 1 に示す．
この文法についての説明は以下の通りである．

- <policy>は<*rule*>，- <*rule*>，<if>，<for>，<substitution>の 5 つの記述から成る．
- <*rule*>には，アクセスの可否に関わるルールを記述する．<*rule*>は $\forall x1 \ldots xn$(<*rule_cond*> $\Rightarrow P(x1, \cdots, xn)$) または $\forall x1 \ldots xn$(<*rule_cond*> $\Rightarrow$"$\neg$"$P(x1, \cdots, xn)$) の形をしていて，<*rule_cond*>には $P(x1, \cdots, xn)$，その否定形，あるいはその連言から成る．
  例えば，次のような記述が可能である．
  $\forall x(Manager(x) \Rightarrow may\_access(x, file1, read))$
  なお，この記述は「課長はファイル 1 に read 可能である.」というルールを表している．
- - <*rule*>には，ルールに関する削除の命令を記述する．例えば，次のような記述が可能である．
  - $\forall x(Manager(x)$
  $\Rightarrow may\_access(x, file1, read))$
  なお，この記述はこの論理式を削除するという命令を表している．
- <if>は，if(<if_cond>){<policy>}の形をしている．else 文が付加されるときは，if 文の後ろに else{<policy>}の形がつく．<if_cond>は<var><<constant>，<var>><constant>，<var>==<constant>，<var>!=<constant>，<constant><<var><<constant>のいずれかの形をしている．なお，不等号の後ろに=(イコール)がつくこともある．
  例えば，次のような記述が可能である．
  if(time<17:00){
  $\forall x(may\_access(x, file1, read))$}
  else{$\forall x(may\_access(x, file2, read))$}
  なお，この記述は「17:00 までは全ての人はファイル 1 に read 可能である．そうでなければ（す

<policy> ::= <*rule*> | "-" <*rule*> | <if> | <for> | <substitution> | <policy> <policy>
<*rule*> ::= $\forall x1 \ldots xn$(<*rule_cond*> "⇒" $P(x1, \cdots, xn)$) | $\forall x1 \ldots xn$(<*rule_cond*> "⇒" "¬"$P(x1, \cdots, xn)$)
<*rule_cond*> ::= $P(x1, \cdots, xn)$ | "¬" <*rule_cond*> | <*rule_cond*> "∧" <*rule_cond*>
<if> ::= "if" "(" <if_cond> ")" "{" <policy> "}" | "if" "(" <if_cond> ")" "{" <policy> "}" "else" "{" <policy>"}"
<if_cond> ::= <cond> | <if_cond> "&&" <if_cond> | <if_cond> "||" <if_cond>
<cond> ::= <var> <rel> <constant> | <constant> <rel> <var> <rel> <constant>
<rel> ::= "<" | "<=" | ">" | ">=" | "==" | "!="
<for> ::= "for" "(" <for_cond> ")" "{" <policy> "}"
<for_cond> ::= <var> "∈" P | <for_cond> "," <for_cond>
<substitution> ::= <var> "=" <constant> | <var> "=" "{" <set_of_constants> "}"
<set_of_constants> ::= <constant> | <set_of_constants> "," <set_of_constants>

ただし，<constant>はユーザが自由に決められる文字列，P は述語記号，$x1, \cdots, xn$ は項とする．

図 1 ポリシー記述言語の文法

なわち 17:00 以降であれば）全ての人はファイル 2 に read 可能である.」というルールを表している．

- <for>は，for(<for_cond>){<policy>}の形をしている．<for_cond>は<var>∈P を並べたものである．
  例えば，次のような記述が可能である．

  for(X ∈Group){
  $\forall x(X(x) \Rightarrow may\_access(x, file1, read))$
  }

  なお，この記述は「Group に属する集合に含まれているユーザならばファイル 1 に read 可能である.」というルールを表している．
- <substitution>は，<var>=<constant>の形をしている．代入するものが<constant>の列である場合もある．
  例えば，次のような記述が可能である．

  Group={Manager,GeneralManager}

  なお，この記述は「Group に課長の集合と部長の集合を代入する」というルールを表している．

3.3 提案言語の表現力に関する利点

提案言語では，3.2 節で述べた文法により，<substitution>においてロールなどのユーザの集合のクラスについて表現可能である．また，- <*rule*>によりポリシー内ルールの削除について表現可能である．これらの新たな表現により，提案言語では以下の 2 つのポリシー表現方法が可能となる．

ルールをまとめる表現方法

条件に応じて異なるポリシーを適用する場合に，ルールに空欄を設け，条件に応じてその空欄に文を入れるという表現方法である．例えば,「17:00 までは課長のみファイル 1 に read 可能であり，17:00 以降は課長と部長がファイル 1 に read 可能である」というポリシーを次のように表現する．

```
if(time<17:00){Group=GeneralManager}
else{Group={Manager,GeneralManager}}
```

for(X ∈Group){
$\forall x(X(x) \Rightarrow may\_access(x, file1, read))$}

ここで，4 行目がルールをまとめる表現である．すなわち，この表現内の X は一種の空欄となっていて，時刻によってこの空欄に入る集合が変化するようになっている．この表現方法により，変化するポリシーを簡潔に記述できる．

ルールの集合をまとめる表現方法

条件に応じて異なるポリシーを適用する場合に，単純に条件毎のルールの集合を列挙するのではなく，全体を通して共通するルールの集合をまず記述し，その後で条件毎に差分がある場合にはその分を足したり引いたりするという表現方法である．例えば，「0:00〜17:00 は課長のみファイル 1 に read 可能であり，17:00〜21:00 は課長と部長がファイル 1 に read 可能であり，21:00 以降は誰もファイル 1 に read 可能でない」というポリシーを次のように表現する．

$\forall x(Manager(x) \Rightarrow may\_access(x, file1, read))$
if(17:00<=time<21:00){
$\forall x(GeneralManager(x)$
$\Rightarrow may\_access(x, file1, read))$}
if(time>=21:00){
- $\forall x(Manager(x)$
$\Rightarrow may\_access(x, file1, read))$}

ここで，1 行目が共通するルールの集合，2〜6 行目が差分のルールの集合となっている．すなわち，ルールの集合を時間毎に全て列挙するのではなく，「常に課長はファイル 1 に read 可能である．ただし，17:00〜21:00 は部長も read 可能で，21:00 以降は課長のファイル 1 に read 可能という権限を削除する」と，ポリシーを共通するルールの集合と差分にあたるルールの集合に分けて記述することが可能である．この表現により，共通するルールの集合をまとめられるためポリシーを簡潔に記述できる．

以上の 2 つの表現方法により，条件に伴い異なるポリシーを適用する場合に，文の集合を列挙する場合に比べてより簡潔に記述できる．この記述が有効な例としては，ワークフローにおいてタスクの進捗に合わせて異なるポリシーを適用する場合や，ユビキタスネットワークにおいて位置情報毎に異なるポリシーを適用する場合などが挙げられる．

なお，ワークフローのポリシー記述に関しては第 4 章にて具体例を挙げて考察する．

Algorithm 1 提案言語によるポリシーから条件に合ったポリシーを生成するためのアルゴリズム

C : 成立するコンテクスト
Policy : ポリシー
NewPolicy : Policy から生成されるポリシー
for i such that $i \in$ Policy **do**
  if i が $rule$ の形である **then**
    NewPolicy = NewPolicy ∪ rule
    Policy = Policy − i
  else if i が - $rule$ の形である **then**
    NewPolicy = NewPolicy − rule
    Policy = Policy − i
  else if i が if(φ){ψ_1}else{ψ_2} の形である **then**
    if $\varphi \in$ C **then**
      Policy = Policy − $i \cup \psi_1$
    else
      Policy = Policy − $i \cup \psi_2$
    end if
  else if i が if(φ){ψ} の形である **then**
    if $\varphi \in$ C **then**
      Policy = Policy − $i \cup \psi$
    end if
  else if i が for($var \in \varphi$){ψ} の形である **then**
    for j such that $j \in \varphi$ **do**
      $var \leftarrow j$
      Policy = Policy ∪ ψ
    end for Policy = Policy − i
  else if i が $var = \varphi$ の形である **then**
    $var \leftarrow \varphi$
    Policy = Policy − i
  end if
end for
return NewPolicy

ロール＼タスク	1		2		3		4	
申請者	(filei,read)	(filei,write)	(filei,read)	(filei,write)	(filei,read)	(filei,write)		
課長	(filei,read)		(filei,read)	(filei,write)	(filei,read)	(filei,write)		
部長	(filei,read)		(filei,read)		(filei,read)	(filei,write)		
総務部門			(filei,read)		(filei,read)		(filei,read)	(filei,write)

ただし，i={1,2,3} とする．

表 1　タスク毎のアクセス権

3.4　アクセスクエリ判定アルゴリズム

提案する言語で記述されたポリシーから，アクセスクエリを判定するためのアルゴリズムは以下の通りである．まず，Algorithm1 によって，提案言語によるポリシーからコンテクストに合ったポリシーを生成する．

このアルゴリズムにより元のポリシーから生成されたポリシーは，必ず Halpern ら [2] の提案言語である一階述語論理の部分体系による記述となっている．すなわち，ポリシーが生成したポリシーはアクセスクエリの判定は実時間で決定可能である．

4　ワークフローへの応用

ワークフローとは，作業プロセスをタスクの流れによって表したものである．このワークフローにおけるポリシーでは，通常タスク毎にポリシーが異なることが有り得るが，ポリシー毎の差異は大幅なものではなく，わずかな差異が積み重なることが実システムでの例として考えられる．このように，条件に伴い少しずつ異なるポリシーを適用する場合に，本研究の提案言語により，ルールをまとめる表現方法，およびルールの集合をまとめる表現方法が有効である．

ここで，本研究の提案言語により以下のポリシーを記述することについて考える．

物品購入申請におけるポリシーの例

- 申請者が申請書を記入する（タスク 1）．
- 課長が申請書をチェックする（タスク 2）．
- 物品の金額が 100 万円以上の場合のみ，部長が申請書をチェックする（タスク 3）．
- 総務部門が申請書を審査する（タスク 4）．
- このワークフローの概要は図 2 の通りである．
- それぞれのタスク中のアクセス権は表 1 の通りであるとする．

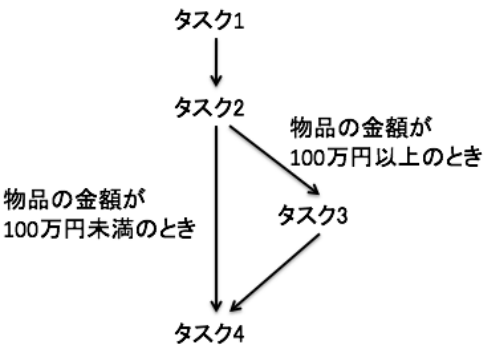

図 2　ワークフローの図

このポリシーを提案言語により記述すると，図 3 のようになる． 図 3 のポリシー記述についての説明は以下の通りである．

- 1〜7 行目はタスクの流れを表している．
- read 権限については，それぞれのロールにほぼ全てのタスクにおいてファイル 1，ファイル 2，ファイル 3 について与えられているため，ルールの集合をまとめる表現方法により記述している．すなわち，11 行目が共通するルールの集合，14 行目および 20 行目が差分のルールの集合となっ

```
 1:  do=task1
 2:  if(finish==task1){do=task2}
 3:  if(finish==task2){
 4:    if(price>=1,000,000){do=task3}
 5:    else{do=task4}
 6:  }
 7:  if(finish==task3){do=task4}
 8:
 9:  Files={file1,file2,file3}
10:  GroupR={Applicant,Manager,GeneralManager,GeneralAffairs}
11:  for(X ∈GroupR,F ∈Files){∀x(X(x)⇒may_access(x,F,read))}
12:
13:  if(do==task1){GroupW=Applicant
14:    for(F ∈Files){- ∀x(GeneralAffairs(x)⇒may_access(x,F,read))}
15:  }
16:  if(do==task2){GroupW={Applicant,Manager}}
17:  if(do==task3){GroupW={Manager,GeneralManager}}
18:  if(do==task4){GroupW=GeneralAffairs
19:    GroupNotR={Applicant,Manager,GeneralManager}
20:    for(X ∈GroupNotR,F ∈Files){- ∀x(X(x)⇒may_access(x,F,read))}
21:  }
22:  for(X ∈GroupW,F ∈Files){∀x(X(x)⇒may_access(x,F,write))}
```

図 3　ポリシー記述例

ている．さらに，11，14，20 行目のそれぞれにおいて，ルールをまとめる表現方法により，ポリシーを簡潔に記述している．この表現内での変数 X および F に関わるクラスについては，9〜10，19 行目にて GroupR，GroupNotR および Files を定めている．以上により，read 権限については共通するルールの集合として全てのロールについて全てのタスクにおいて 3 つのファイルに対するパーミッションを与え，差分のルールの集合として申請者・課長・部長の 3 つのロールについてはタスク 4 の read 権限を削除し，また総務部門のロースについてはタスク 1 の read 権限を削除している．

- write 権限については，ルールをまとめる表現方法により記述している．すなわち，22 行目の write 権限に関するルールの中で，変数 X がタスク毎に切り替わる．この変数 X については，13，16〜18 行目にてタスク毎の GroupW の場合分けを行っているため，タスクにより X が変化する．これにより，タスク 1 では申請者が，タスク 2 では申請者と課長が，タスク 3 では課長と部長が，タスク 4 では総務部門が 3 つのファイルに対する write 権限を持つようになる．

なお，このポリシーについて既存の一階述語論理を基にした言語により記述すると，

- タスクの流れについては，図 3 の 1〜7 行目とほぼ同様の記述をする．
- タスク毎にパーミッションについては，全てを列挙する必要がある．この場合，

$$\forall x(do(task1) \wedge Applicant(x) \Rightarrow may\_access(x, file1, read))$$

のような記述をパーミッション全てについてする

必要がある．つまりパーミッションに関して合計 54 行の記述が必要となる．

このように，既存の言語ではポリシー記述が煩雑となる．その理由は 2 つある．1 つ目の理由は，集合に対するクラスを表現できないので，ルールをまとめる表現方法ができないためである．例えば先程の例において，既存言語では「申請者と課長をまとめて GroupR とする」のような，集合に対するクラスを表現することができない．しかし，本研究の言語ではこの表現が可能である．また，もう 1 つの理由は，ルールの削除を表現できないので，ルールの集合をまとめる表現ができないためである．例えば先程の例において，既存言語により「申請者は全てのタスクにおいてファイル 1 に read 可能である．ただしタスク 4 を除く.」という表現をすると，ポリシーの矛盾が生じてしまう．しかし，本研究の言語ではルールの削除が可能であるため，ポリシーに矛盾が無いまま差分のルールの集合を表現することができる．

以上により，本研究の提案言語ではこのワークフローの例を簡潔に記述可能である．

5 結論と今後の課題

既存の論理言語を用いたポリシー記述言語に制御構造を導入し，複雑な条件を簡潔に表現し得る形式言語を提案した．特に，ルールをまとめる表現方法，およびルールの集合をまとめる表現方法を記述可能とした．さらに，この言語により記述されたポリシーからアクセス可否を判定するアルゴリズムを示した．

今後の課題として，提案言語により記述されたポリシーからアクセス可否を判定する際の計算量について考察，ならびに同事項を実験により評価することが挙げられる．また，本研究ではポリシーの矛盾に関する問題について扱っていないため，今後考察する必要がある．

参 考 文 献

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin.:A calculus for access control in distributed systems, ACM Transactions on Programming Languages and Systems, 1993, 15(4):706-734.

[2] J. Y. Halpern, and V. Weissman.:Using first-order logic to reason about policies, SACMAT'03, 2003, 187-201.

[3] S. Benferhat, R. E. Baida, and F. Cuppens.:A stratification-based approach for handling conflicts in access control, SACMAT'03, 2003, 189-195.

[4] S, Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian.:Flexible support for multiple access control policies, ACM Transactions on Database Systems, 2001, Vol.26, No.2, 214-260.

[5] S, Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino.:A unified framework for enforcing multiple access control policies, In Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, 1997, Vol.26, 474-485.

[6] S, Jajodia, P. Samarati, and V. S. Subrahmanian.:A logical language for expressing authorizations, In Proceedings of the IEEE Symposium on Security and Privacy, 1997, 94-107.

[7] L. Wang, D. Wijesekera, and S. Jajodia.:A logic-based framework for attributed based access control, ACM FMSE'04, 2004, 45-55.

[8] E. Yuan, and J. Tong.:Attributed based access control (ABAC) for webServices, In Proceedings of the IEEE International Conference on Web Services(ICWS'05), 2005, 561-569.

[9] V. Atluri, and J. Warner.:Supporting conditional delegation in secure workflow management systems, ACM SACMAT'05, 2005, 49-58.

[10] J. Crampton, and H. Khambhammettu.:Delegation and satisfiability in workflow systems, ACM SACMAT'08, 2008, 31-40.

[11] B. W. Lampson:Protection, Proc. 5th princeton Conf. on Information Sciences and Systems, 1971.

[12] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman.:Role-based access control models, IEEE Computer, 1996, 29(2):38-47.

[13] National Security Agency: Security-Enhanced Linux, http://www.nsa.gov/research/selinux/

[14] T. Moses:The extensible access control markup language, version2.0, http://www.xacml.org, 2005.

[15] M. Abadi:Logic in access control, Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science, 2003, 228-233.

[16] N. Li, B. N. Grosof, and J. Feigenbaum.:Delegation Logic: A logic-based approack to distributed authorization, 12th IEEE Computer Security Foundations Workshop, 1999.