動的なセキュリティポリシのためのRBACシステムの設計

山崎 航[††] 平石広典[††] 溝口文雄[†]

多くの現実的なコンピュータソフトウェアは，動的な側面を備えているが，この問題に対して，既存のアクセスコントロールメカニズムでは，十分に扱えないという現状にある．本論文では，動的なセキュリティポリシを扱うためのアクセスコントロールシステムについて述べる．我々が提案する方法は，RBAC（Role-Based Access Control）を基本としており，あらかじめコンテクスト情報と抽象ロールを用いて静的に定義されたルールを用いて，動的に具体的なロールを決定する．ルールを論理型言語による宣言的な表現で記述することによって，ユーザ，ロール，パーミッションに対して，双方向の問合せが可能となる．本論文では，プロジェクトマネジメントシステムのシンプルな例を用いることによって，提案する方法の有効性について議論する．

Designing a Rule-Based RBAC System for Dynamic Security Policy

Wataru Yamazaki,[††] Hironori Hiraishi[††] and Fumio Mizoguchi[†]

Most practical applications have dynamic attributes, but conventional access control mechanisms have not addressed the problem sufficiently. In this paper, we discuss how to realize an access control system that enables us to manage dynamic security policies. Our proposed method is based on Role-Based Access Control (RBAC), and the rule-based agent decides access rights dynamically for the abstract role, which is defined by the role administrator statically using context-enabled rules and an inference engine. By defining rules using declarative representation (logic programming style), bidirectional queries can be realized for USER-ROLE-PERMISSION relationships. In this paper, we will demonstrate the usefulness of our proposed system by presenting our project management application and its access control system.

1. はじめに

アクセス制御はすべての計算機プログラムに重要であり，その目的は悪意のある攻撃を防ぐこと，および正規のユーザや管理者のミスによる意図しない動作や正しくない動作を防ぐことである．しかしながら，正しくアクセス制御のポリシを制定し実行することは少なくとも以下のような2つの理由により，困難な問題である．第1に，今日のコンピュータプログラムが複雑になり，ネットワーク越しの制御が可能となったことから，プログラム中における制御対象のリソースとユーザの数が従来に比べて増加したことである．第2の理由は，多くのアプリケーションが多かれ少なかれ動的な側面を含んでおり，動的に決定されるすべての状況に応じた制御方法をあらかじめ決めておくことは，非常に困難であることである．

第1の問題に対する最近の手法は，役割に基づくアクセス制御（Role-Based Access Control: RBAC）を[1]適用することである．RBACでは，ユーザはロールと呼ばれる実際の対象リソースの集合に関連付けられる．つまり，ユーザはロールにマッピングされ，ロールが対象リソースにマッピングされる．これにより，RBACではMAC（Mandatory Access Control）やDAC（Discretionary Access Control）といった既存のアクセス制御システム[2]と比較して利用者やリソースの追加，削除といった場面において，シンプルなアクセス管理を実現しており，いくつかのモデル[3),4)]が提案され，ヘルスケア[5]やWeb[6]といった応用例が多く開発されている．

しかしながら，これまで上記の第2の問題については十分に議論されてこなかった．ここでいう動的な側面とは，ユーザの入力やGUI操作といった，アプリケーションの状態に応じたアクセス制御，ユーザの場所に応じたアクセス制御，時間に応じたアクセス制

† 東京理科大学理工学部
Faculty of Science and Technology, Tokyo University of Science
†† 東京理科大学総合研究所
Research Institute for Science and Technology, Tokyo University of Science

御，排他制御，ロールの委譲などを指す．ロールの委譲については近年活発に議論されているもの[7),8)]，一般的には動的なロールの割り振りについて，既存の RBAC では対応しないために，これまでのアプリケーションでは，プログラマは個別に設計・実装時に必要なすべての状況をあらかじめ記述しておく必要があった．しかしながら，これらのオペレーションの多くは，多くのアプリケーションで共通であったり類似しているものを含んだりしており，一般化が可能である．

本論文では，我々はこの第 2 の問題を解決するための拡張した RBAC システムを提案する．基本的なアイデアは，まず抽象ロールを定義し，具体的なロールつまりアクセス権の集合をユーザの位置，時間といった状況情報とルールを用いて動的に決定する．我々のモデルは，抽象ロール集合，状況ルール，そして状況情報の収集機能および推論機能を持つエージェントからなる．状況情報はエージェントによって動的に生成され，エージェントは，抽象ロール集合，状況ルール状況情報の 3 つの情報から具体的なロールを生成する．3 つの情報およびエージェント自体は一階述語論理プログラミング（prolog）の形式によって記述されるため，それらの関係において，双方向の問合せが容易に実現できる．

本論文では，我々が提案するモデルの特性と有効性を，プロジェクトマネジメントシステムの例を示しながら議論する．

本論文は，以下のように構成される．2 章では，基本的な RBAC について簡単に説明する．3 章では，動的な環境の RBAC システムについて，プロジェクトマネジメントシステムの例のシナリオを示すことによって，基本的なアイデアについて述べる．4 章では，我々の提案するモデルについてそのコンポーネントと役割について示し，5 章ではシステム実現のための実装について述べ，6 章では関連研究を示した後，本論文をまとめる．

2. Role-Based Access Control

RBAC では，ユーザが役割を果たすために必要最低限の権利を役割ごとに分類することによって，不正またはミスによるアクセスを防ぐとともに，管理の間違いを減らすことができるとされるが，それは RBAC がシンプルでありながら様々なセキュリティポリシに対応することができるためである．ここでは，Ferraiolo らによる，NIST's enhanced RBAC Model [1)]（以下基本的な RBAC モデルとする）に従って，そのモデルを示す．まず，ユーザ，ロール，パーミッションの関係は，以下のように定義される．

- $user$, 個々のユーザ
- $role$, 個々の role
- $RM(r : role) \rightarrow 2^{user}$
  この関数は role r が与えられたときに，r を演じることのできるユーザの集合を返す．
- $permission = 2^{(operation \times object)}$
- $RP(r : role) \rightarrow 2^{permissions}$
  role/permission のマッピングで，ロール（role）r に許されたパーミッションの集合を返す．

ここで，パーミッション（permission），または権利（privilege）は 1 つ以上のオブジェクト（object）に対してのオペレーション（operation）への許可であり，1 つのパーミッションは 1 つ以上のロールと結び付いており，1 つのロールは 1 つ以上のパーミッションと結び付いている．

基本的な RBAC モデルでは，階層と制約という 2 つの特徴を備えている．ロールの階層は 1 つのロールに属するすべてのアクセス権が，異なるもう 1 つのロールに属することによって実現する．ここで，ロールの階層を示すための，$A \succeq B$ という記法では，ロール A がロール B を含んでおり，そのため，A は B としても扱うことができることを表す．

$$(\forall i, j; role)(\forall u : user) i \succeq j \wedge u \in RM[i] \Rightarrow u \in RM[j]$$

基本的な RBAC モデルでは，2 つの静的な制約をユーザ/ロール間，およびロール/パーミッション間でサポートする．SSD（Static Separation of Duty）とロール濃度（Role Cardinality）である．

SSD はユーザが認可され，ロールが互いに排他的である場合の制約であり，ある 1 つのロールをユーザが得ている場合，もう一方のロールを得ることができない．

$$Ea : role \times role$$

両立できない role の組合せに対して Static Separation of Duty（SSD）は以下のように定義される．

$$(\forall u : user)(\forall i, j : role)$$
$$u \in RM[i] \wedge u \in RM[j] \Rightarrow (i, j) \notin Ea$$

つまり，あるユーザが i と j の両方のメンバであるなら，i と j は SSD にはならない．

また，もう一方の制約である，ロール濃度は 1 つのロールに対して同時に演じることができるユーザの最大数を定義する．

3. 動的な環境のための RBAC

前章で示した基本的な RBAC モデルは，ユーザ，

ロール，パーミッションの関係は静的であり，動的に決定する場合には，ロール管理者は，あらかじめその関係を定義しておく必要がある．本章では，まず，そのような既存の RBAC の限界点をサンプルシナリオを通して明らかにし，その後我々のアプローチを提案する．

3.1 サンプルシナリオ

本節では，既存の RBAC の限界点を示すために，プロジェクト管理システムにおけるアクセス制御システムについて述べる．プロジェクト管理システムの目的は，エンタープライズ分野における各種プロジェクトを効率良く管理するために，利用者やリソースのスケジューリング，タスクの結果の評価，進捗管理などを行うことを想定する．プロジェクトは，プロジェクト管理者，タスク実行者，タスクメンバ，などからなり，その操作のほぼすべては，アクセス制御の対象となる．ここでは簡単のために，以下のようなシナリオを例として示す．

(1) プロジェクトタスクの管理者は，タスクをサブタスクに分割し，サブプロジェクトに対して適切なリソースを割り振り，スケジューリングされた結果をすべてのプロジェクトメンバに通知する．
(2) タスクの実行者は割り当てられたタスクを実行し，実行結果をシステムにレポートする．
(3) すべてのプロジェクトメンバは，実行者によって実行されたレポートを閲覧する．
(4) タスクが終了すると，タスクの管理者は，評価を行う．タスクが期限までに実行されなかった場合．タスクの再スケジューリングを行う．

以上のオペレーションを行うためには，最低限以下の 3 つのロールが必要となる．

TaskManager：このロールは，上記 (1)〜(4) のすべてのオペレーションを行える．タスクの管理者であり所有者であり，タスクの生成，リソースアロケーション，削除，タスクの評価を行う．一般に RBAC では，すべてのリソースに対してアクセスできるロールを作成しないが，ここでは，担当するタスクに対してすべてのリソースにアクセスできるのであって，すべてのタスクに対してアクセスできるわけではないことに注意されたい．

TaskExecutant：このロールは上記の (2) のオペレーションを行い，タスクの実行後にはレポートの読み書きを行う．

TaskMember：このロールは上記 (3) のオペレーションを行い，このロールを演じるユーザはスケジュール，レポート，評価といった情報を読むことができる．

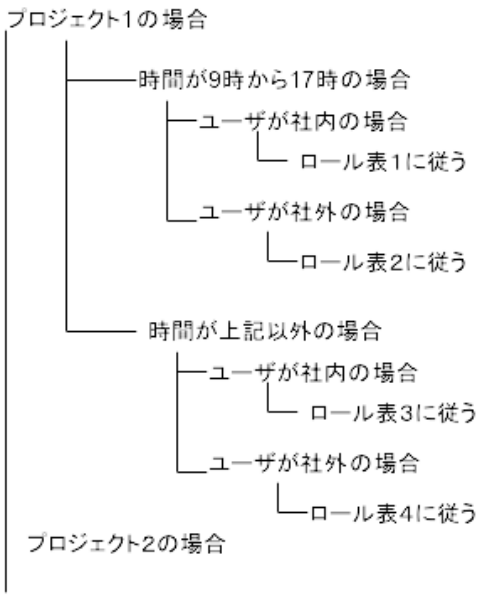


図 1 基本的な RBAC での動的な状況の記述例
Fig. 1 Example of a dynamic situation by the basic RBAC.

ここで，上位のロールは，下位のロールのオペレーションを含み，これは RBAC におけるロールの階層によって表現できる．つまり，以下の関係を満たす．

$$member \prec executant \prec manager$$

以上に示したシナリオでは，前章で示した基本的な RBAC をそのまま用いることは適当でない．主な理由はプロジェクトが複数存在することであり，たとえばあるプロジェクトのマネージャがほかのプロジェクトのマネージャであるとは限らず，さらに，ツールが対象とするプロジェクトは，ユーザの操作（アプリケーションの状態），時間，場所，といった制約により動的にアクティブになる．

たとえば，基本的な RBAC システムでは，プロジェクトが複数あり，時間，ユーザの場所という複数の状況によって，ロールを決定するためには，**図 1** のようにして，複数のロールの関係をあらかじめ記述しておく必要がある．ユーザ，ロール，プロジェクト，状況といった設定の対象が多くなるほど，それらの関係はより複雑になり，すべての状況についてあらかじめロールの関係を決定しておくことは，合理的な解決方法とはいえない．さらに，優先度に基づくロール濃度の変更や，ロールの委譲といった動的な性質も基本的な RBAC では扱うことはできない．したがって，こ

表 1 抽象ロール集合
Table 1 Abstract role set.

Project Name	Name
Target	T1,...
Role:Manager	T1.makeSchedule(); T1.deleteSchedule(); ...
Role:Executant	T1.setResult(); ...
Role:Member	T1.readSchedule(); ...

のような動的な属性を扱うためのアクセス制御システムが必要となる．

3.2 動的な環境のための RBAC システム

本節では，動的な環境のための RBAC を実現するための基本的な考え方について示す．

計算機システムにおいて，動的な動作を実現する一般的な方法は，まず，動作する対象を抽象的な形式によって表現し，具体的な詳細を実行時に決定する方式である．たとえば，オブジェクト指向プログラミングにおけるインタフェースや抽象クラスはそのような例の 1 つであり，ライブラリ設計者はライブラリを作成する際に，抽象メソッドを利用することで，ライブラリ利用者であるアプリケーション作成者が具体的な動作や実装方式を決めることを可能とする．また，プログラム中における変数という考え方も，最もシンプルな例といえるかもしれない．たとえば，prolog において，リスト処理を行うために，抽象的な論理変数を `[Top|Tail]` のように利用することで，リストの先頭とそれ以外のリストを区別することができる．

我々は，動的なアクセス制御のための RBAC において，同様なアプローチをとる．初めに，抽象ロール集合と呼ぶ抽象的なロールの集合を静的に定義しておき，アクセス権の関係を決定するためのルールを適用する．具体的なアクセス権は，抽象ロール集合とルールおよび状況を表す事実である，状況情報によって動的に決定する．つまり，アプリケーション（先に示したシナリオではプロジェクト管理システム）に対し 1 つの抽象ロール集合を定義し，各プロジェクトごとにルールを定義することによって，観測される状況情報を用いてアクセス権を決定する．たとえば，プロジェクト管理システムのシナリオにおける抽象ロール集合は，**表 1** のように記述できる．

プロジェクトの名前を示す変数 Name が決定すると，ターゲットのオブジェクトである変数 Target が決定し，各ロールに対するアクセス権が自動的に割り当てられる．表 1 では，ターゲットオブジェクトに対するアクセス権は，`T1.makeSchedule()` や `T1.setResult()` のようにオブジェクト指向プログラミングで用いられる形式によって記述されている．表 1 は，プロジェクト名の変数 Name が決定すると，ロール Manager は上記 2 つの作業を行うことができ，ロール Executant は，ターゲットオブジェクトに対して `setResult()` を実行でき，ロール Member は，ターゲットの `readSchedule()` を実行できるということを示している．表 1 では，ターゲットは T1 1 つだけであるが，提案するモデルは 1 つの抽象ロールに対してターゲットの数を制限しない．

表 2 ロールマッピングのルール例
Table 2 Example of a role mapping.

Name	project_task1
Target	task1,...
Manager	userA
Executant	userB, userC
Member	userA, userB, userC, userD,...

表 3 時間制約を含むルールの例
Table 3 Example of a rule with a time constraint.

Name	Peoject_task1
Target	task1
Manager	userA
Executant	Executant_Candidate and ($1000 \leq time \leq 1700$)
Member	userA, userB, userC, userD,...
Executant_Candidate	userB, userC

表 1 のプロジェクトに対するロールマッピングを決定するルールは，たとえば **表 2** のように記述される．

表 2 のルールが適用されると，ユーザ，userA はプロジェクト project_task1 に対して，ロール Manager およびロール Member を演じることができるようになる．たとえば，あるユーザがこのルールにおいて Manager のロールを持っているとすると，ターゲット “task1” に関連付けられたオブジェクトに対して，オペレーション makeScedule() および deleteScdule() へのアクセス権を持つことになる．表 2 は，最もシンプルなルールの例であり，すべてのロールは直接このルールのみから得ることができる．一般的には，我々が提案するモデルではより複雑なルールが適用される．たとえば，ロール管理者のポリシにより，ロール Executant が正しい（アクティブである）とされるのが時刻 10:00 から 17:00 までであるようにしたい場合，**表 3** のようなルールを定義する．

表 2 と表 3 の違いは，表 3 におけるロール Executant の定義している対象がユーザのリストではなく，変数 Executant_Candidate および制約 $1000 \leq 1700$ となっていることである．Executant_Candidate は，

対応するユーザが userB および userC であることが表の 6 行目で宣言されている．表 2 もしくは表 3 のようなルールを定義することによって，抽象ロール集合にマッピングするためのパーミッションの適切な関係をシステムが探索することになる．

4. ルールベース管理システム

本章では，前章で示した方式をルールベースの推論機構によって実現する方法について述べる．本章で示す方法では，具体的なアクセス権を抽象ロールおよび状況情報から推論する．前章で示したいくつかのルールは，情報収集を担当するプログラムであるエージェントによって実行時に自動的に生成される．それ以外のルールはロール管理者によって静的に定義される．

4.1 本システムのコンポーネント

図 2 は，システムの概要を示しており，推論機構を備えるエージェント，エージェントへの入力，エージェントからの出力の 3 つに大別される．エージェントの入力は，抽象ロール集合および状況情報，状況ルールから構成される．エージェントは上記の入力から，自身が備える推論エンジンを用いて，アクセス権を出力する．このモデルは非形式的に以下のように説明される．

- 抽象ロール集合
  抽象ロール集合は，前章で示したような変数を用いることによって，抽象的なパーミッションのマッピングを定義し，パーミッションは変数の束縛によって決定する．変数束縛は，エージェントの推論によって行われ，推論は，次に示す状況情報および状況ルールを用いて実行される．抽象ロール集合は，ロール管理時にロール管理者によって静的に定義される．
- 状況情報
  状況情報は，次に述べるエージェントによって観測される状態から構成される．本論文では，状況情報として，ユーザの操作によるアプリケーションの状態，時間，ユーザの場所，ロールの委譲，および排他制御を扱う．そのため，状況ルールは，アプリケーションとユーザ環境に基づく状況によって動的に変化する，事実とルールから構成される．
- 状況ルール
  状況ルールは静的なルールであり，エージェントの入力の一部である．状況ルールは上記状況情報が生成されることを前提としており，状況情報がエージェントによって生成された際のアクセス権を生成するルールを定義する．

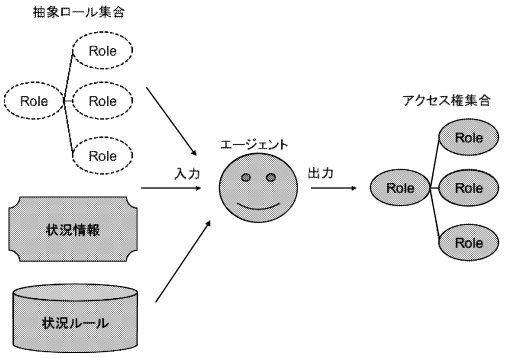


図 2 ルールベース管理システムの入出力関係
Fig. 2 Input/Output relation of the rule-based management system.

- エージェント
  エージェントは推論によって具体的なアクセス権を生成する．エージェントの入力は抽象ロール集合，状況情報，および状況ルールであり，出力は下記の具体的なロール集合である．ユーザの問合せも状況情報として扱われ，エージェントは問合せに対して推論を実行する．
- アクセス権集合（具体的なロール集合）
  アクセス権集合はエージェントによって動的に決定されたアクセス権の集合である．ロールはアクセス権の集合であるので，アクセス権集合は，具体的なロール集合と見ることができる．

4.2 システムへの入出力

前章で示したルールを用いて，前節で示したコンポーネントによる実現を可能とするために，我々は，論理型言語を用いる．抽象ロール集合，状況情報，状況ルール，はそれぞれルールおよび事実として表現でき，用いられる変数はユーザ名やロール名などであることから，一階述語論理のプログラミング言語（prolog）によって実現できる．また，エージェントは，それらの prolog プログラムを実行する推論エンジンとしても動作する．さらに，状況に関する情報である状況情報は，エージェントによってその時点で観測され，動的に生成される節として表現される．本節では，3 章で示した例を用いながら，エージェントとその入出力の関係について定義する．

4.2.1 抽象ロール集合

抽象ロール集合は，節の集合として定義される．ロール管理者は，抽象ロールの名前，ユーザがマッピングされるロールの名前，ロールの対象となるオブジェクト，アクセス権，のそれぞれを，prolog の形式によって記述する．表 1 に対応する，prolog 形式のルール

は，以下のように記述される．

```
makeSchedule(U,T) :-
  role(manager,U), target(T).      (Rule 1)
deleteSchedule(U,T) :-
  role(manager,U), target(T).      (Rule 2)
setResult(U,T) :-
  role(executant,U), target(T).    (Rule 3)
readSchedule(U,T) :-
  role(member,U), target(T).       (Rule 4)
```

ここでは，Rule1 は，ユーザ U がマネージャロールであり，ターゲットが T である際には，ユーザ U が T に対してオペレーション makeSchedule を実行可能であることを示している．Rule2 から Rule4 も同様である．一般的には，抽象ロール集合の定義は，すべてのアクセス権を述語 `role(R,U)` および述語 `target(T)` を利用して定義する．ここで，述語 `role(R,U)` と `target(T)` の連言によって，現在ユーザ U は，ターゲット T とロール R を選択していることを示しており，その際，アクセス権 *perm* を行えることを述語 `perm(U,T)` を定義することによって次のように宣言する．

```
perm(U,T):- role(R,U),target(T).
                                   (Rule 5)
```

なお，ここで，述語 role および target の定義はされていない．これらは，後に示す，状況情報，および状況ルールによって定義される．

4.2.2 状況情報

状況情報は，エージェントによって観測され，動的に生成される節の集合である．たとえば，表 3 では，現在時刻と現在のユーザ情報が，節として以下のように生成される．状況情報は，ロール管理者が抽象ロール集合および状況ルールを記述する際に組み込み述語のように利用される．

```
user(userA).                       (Rule 6)
target(task1).                     (Rule 7)
selected(manager).                 (Rule 8)
```

ここで，Rule6 の `user(U)` は，ユーザ U がシステムにログインしている状況を述べる組み込み述語であり，ユーザのログインと同時に，エージェントによってエージェントのルール節集合に加えられ，ログアウトと同時に削除される．Rule7 の，述語 `target(T)` は，ユーザによって選択されている．（もしくは選択されようとしている）対象オブジェクトが T であることを示している．Rule9 の `selected(R)` は，ロール R がユーザによって選択されたことを示している．

ここで，組み込み述語は，システムがあらかじめ定義する述語のほか，システムの組み込み述語を用いたユーザ独自の組み込み述語の定義がある．ユーザによる組み込み述語の定義は，アプリケーション動作のタイミングに大きく依存するため，アプリケーション作成のための API として，実装言語用に提供されているものとする．

4.2.3 状況ルール

状況ルールは，エージェントによって観測される状況によって生成される状況情報を生成するためのルールを定義する．また，ロールの階層に関するルールも状況ルールの一部となる．このルールを用いて，エージェントはアクセス権を決定することができるようになる．表 3 に対応するルールは，以下のように記述される．

```
rolemember(member,userD).          (Rule 9)
executant_candidate(userB).        (Rule 10)
executant_candidate(userC).        (Rule 11)
rolemember(executant, U) :-
  executant_candidate(U) ,sys_time(X),
  X>1000, X<1700.                  (Rule 12)
rolemember(manager, userA).        (Rule 13)
role(R,U) :-
  user(U), selected(R), rolemember(R,U).
                                   (Rule 14)
rolemember(member, U) :-
  executant_candidate(U).          (Rule 15)

rolemember(executant, U) :-
  rolemember(manager, U).          (Rule 16)
```

ここで，Rule9 は，userD は，ロール member を演じることができることを宣言している．実際に userD が member ロールになっているかどうかは，先に示した状況情報の問合せをすることによって調べることができる．Rule10 から Rule12 までの 3 つのルールによって，userB，および userC が，ロール executant を演じることができるのは，その時刻が 10:00 から 17:00 の間に限ることを定義している．ここでは，動的にロールになれるかどうかを決定するために，システムの時間を得る組み込み述語 `sys_time(T)` を，Rule14 は，あるユーザ U がロール R を得るという状況を定義している．ここで組み込み述語 `selected(R)` は，ユーザによって，UI 上でロール R が選択されたことを意味する．

なお，エージェントが観測する状況とは直接関係な

いものの，基本的な RBAC における SSD，ロール濃度，ロール階層などの制約は，状況ルールにおいて記述されているものとする．Rule15 および Rule16 は，Role の階層を定義するルールであり，Rule15 は，ユーザが executant_candidate ならば，ユーザはロール member のメンバでもあることを，つまり，すべての member のアクセス権を executant_candidate は実行できることを意味している．Rule16 では，manager ロールは，executant ロールの上位ロールであることを示している．

ここで，たとえば Rule12 に対してユーザの場所に応じたアクセス制御を追加したい場合には，ユーザ U の場所を知る組み込み述語を place(U) とし，ホスト host1 からのみアクセスを許すようにするならば，以下のように Rule 12 を変更する．

```
rolemember(executant, U) :-
  executant_candidate(U) ,sys_time(X),
  X>1000, X<1700,
  place(U), U=host1.
```

4.2.4 エージェント

エージェントの第 1 の機能は，上記の 2 つのルール集合を用いて，適切なアクセス権を動的に生成することである．また，そのために，状況情報を生成する機能も必要とする．上記の例では，抽象ルール集合と状況ルールが定義されており，ユーザがすでにタスクを選択し，マネージャロールを得ている際に，Rule6, Rule7, Rule8 を生成する．この状況では，ターゲット task1 への，オペレーション makeSchedule が userA が可能かどうかを調べるためには，以下のような質問を推論エンジンであるエージェントに対して行えばよい．

```
?- makeSchedule(userA,task1).
                                    (Query 1)
```

この質問によって，エージェントは，ユーザ A が対応するアクセス権があるかどうかを prolog エンジンによって推論する．ここで，双方向の問合せが可能となることは，論理型言語を用いる利点の 1 つである．たとえば，システムが，オペレーション setResult を行えるユーザのリストを得たい場合，動的に生成される状況情報，および，問合せとその答えは，以下のように得ることができる．ここで，(1) から (3) までが状況情報，(4) が問合せ，(5) 以降が答えに相当する．

```
user(_).                            (1)
target(task1).                      (2)
selected(_).                        (3)
?-setResult(X,task1).               (4)
X = userB;                          (5)
X = userC;                          (6)
X = userA                           (7)
yes                                 (8)
```

5. 実装に関する考察

本章では，前章までで示した拡張した RBAC を実装する際の方式について述べる．我々は，前章で示したシナリオおよび例において，時間に基づいたアクセスを見た．この例において，ユーザが RBAC サーバに問合せをした時点において成立していた制約が，実際のリソースアクセス時には，制約が満たされていない場合が考えられる．このような状況に対応するために，我々は実装上では，チケットを用いたシステムによって実現する．つまり，RBAC サーバへの問合せを行うとサーバはユーザにロールチケットを発行する．ロールチケットに，ロールの有効期限や優先度といった制約を設定し，ユーザはロールチケットに対応するデバイスを管理するコントロールマネージャに提出する．ここでコントロールマネージャは，アクセス対象のオブジェクトを管理するプログラムのことをいう．なお，有効期限や，優先度といった，チケットに含まれる情報についてはアプリケーション固有の問題となるため，本論文では扱わない．さらに，有効期限などが切れた場合に，そのことをどのようにユーザに知らせるかといった問題もアプリケーション固有の問題である．これらのアプリケーション固有の問題に対しては，チケットとの配布とその正しさを検証するコントロールマネージャという構成によって，制御時の制約を解決するものとする．

図 3 は，本システムを実現するためのシステム構成を示しており，基本的な動作は以下のように行われる．

(1) ユーザはロールサーバに対してユーザ名とパスワードおよび状況情報を提出することにより，利用可能なロールのリストを要求する．

(2) ロールサーバは提出された情報と状況情報お

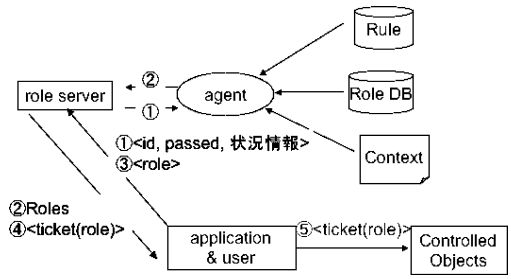


図 3 システム構成
Fig. 3 System architecture.

よびルールから，ユーザが演じることのできるロール集合を生成し，それをユーザに提示する．

(3) ユーザは提示されたリストの中から所望のロールのチケットをロールサーバに要求する．

(4) ユーザはチケットを得た後にはチケットをコントロールマネージャに提出する．

(5) コントロールマネージャはチケットが正しいことを検証し，正しい場合に限り制御を許可する．

チケットシステムと，本論文で示した動的なロール決定の仕組みを組み合わせて用いることにより，たとえば，ロール濃度は，ロールサーバが発行するチケットの枚数とし，コントロールマネージャがユーザからチケットを回収しロールサーバに返還することによって実現できる．また，排他制御はロールの濃度が 1 の場合と考えることができる．優先度を用いた制御を実現するためには，ロールサーバがチケットに付加した優先度情報を処理する機構がコントロールマネージャに必要となる．さらに，ロールの委譲はチケットの委譲と考えることができるので，一時的なロールの委譲も実現可能である．

6. 関連研究とまとめ

本論文で我々が提案した抽象ロール集合は構造的には Giuri らによる Role Template[10] と類似している．我々のモデルでは，抽象ロールは，管理者によって静的に定義される状況ルールと，エージェントによって動的に生成される状況情報を用いることを仮定し，状況を利用することで動的な情報を扱えることを強調しており，Giuri らがロールの容易な管理を目的としてる点と大きく異なる．さらに，我々のシステムは，抽象ロール，状況ルール，状況情報のすべてが，論理型言語によるルール構造をしていることから，双方向の問合せが可能となっている．

Zhang ら[8] は，prolog によるルール記述を主にロール委譲に用いている．ロールの委譲はロールの動的な一面と見ることもできる．我々の研究は，これに対して状況情報を利用した動的なアクセス権の生成に焦点を当てており，Zhang らのシステムは，動的な環境のための状況や抽象的なロールに重きを置いていない．細かな委譲の仕組みを提供する，より一般的なアプローチである Zhang らのアプローチとは我々のアプローチは異なるが，これらは互いに競合せず相補的なものであると考える．

Convington らは，基本的な RBAC を role の概念を subject のみでなく，object と environment といった概念に拡張し，より細かなセキュリティポリシを定義することを提案している[11]．この研究は，状況を利用するためのモデルを与えるが，そのためにロール管理者は，数多くのロールとその関係を定義する必要がある．これに対して我々の提案するモデルでは，ロールを抽象的に与え，ルールによってロールを生成するアプローチをとることによって，管理するロールとその関係を少なくすることを目的としている．

本論文では，我々は，動的な属性を持つアプリケーションのためのアクセス制御システムの構成とそのモデルを提案した．動的なアクセス権はルールと状況，およびそれらを入力とする推論エンジンからなるエージェントによって決定される．現在，我々は本論文で示したシステムの実装として，ロールサーバおよびロールマネージャを Web サーバ上に構築し，アプリケーション作成のための API を開発している．

参 考 文 献

1) Ferraiolo, D.F., Barkly, J.F. and Kuhn, D.R.: A role based access control model and reference implementation within a corporate internet, *ACM Trans.Information Systems Security*, Vol.2, No.1, pp.34–64 (1999).

2) Joshi, J.B.D., Aref, W.G., Ghafoor, A. and Spafford, E.H.: Security models for web-based applications, *Comm. ACM*, Vol.44, No.2, pp.38–44 (2001).

3) Sandhu, R.: Rational for the RBAC96 family of access control models, *Proc. 1st ACM Workshop on Role-based Access Control* (1996).

4) Sandhu, R., Bhamidipati, V. and Munawer, O.: The ARBAC97 model for role-based administration of roles, *ACM Trans. Information and System Security*, Vol.2, No.1, pp.105–135 (1999).

5) Chandramouli, R.: A Framework for Multiple Authorization Types in a Healthcare Application System, *17th Annual Computer Security Applications Conference* (*ACSAC*) (2001).

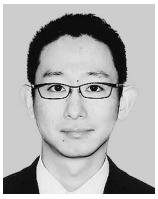
6) Ferraiolo, D.F. and Barkley, J.: Specifying and Managing Role-Based Access Control within a Corporate Intranet, *Proc. 2nd ACM Workshop on Role-Based Access Control*, pp.77–82 (1997).

7) Barka, E. and Aandhu, R.: A role-based delegation model and some extensions, *Proc. 16th Annual Computer Security Application Conference*, pp.101–114 (2000).

8) Zhang, L., Ahn, G.-J. and Chu, B.-T.: A Rule-Based Framework for Role-Based Delegation and Revocation, *ACM Trans. Information and System Security*, Vol.6, No.3, pp.404–441

(2003).
9) Sandhu, R.S., Coyne, E.J., Feinstein, H.L. and Youman, C.E.: Role-Based Access Control Models, *IEEE Computer*, Vol.29, No.2, pp.38–47 (1996).
10) Giuri, L. and Iglio, P.: Role templates for content-based access control, *Proc. 2nd ACM Workshop on Role Based Access Control*, pp.153–159 (1997).
11) Convington, M., et al.: Securing Context-Aware Applications Using Environment Roles, *Proc. SACMAT'01*, pp.10–20 (2001).

(平成 17 年 7 月 8 日受付)
(平成 18 年 3 月 2 日採録)

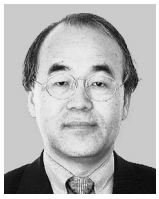


山崎　航（正会員）

1974 年生．2004 年東京理科大学大学院理工学研究科経営工学専攻博士後期課程修了，博士（工学）．同年同大学情報メディアセンターポストドクトラル研究員，2005 年東京理科大学総合研究所研究員，現在に至る．分散システム，マルチエージェント，グリッドコンピューティングに興味を持つ．日本ソフトウェア科学会，人工知能学会，AAAI，ACM 各会員．

平石　広典（正会員）

1971 年生．1997 年東京理科大学大学院理工学研究科経営工学専攻修士課程修了．2000 年同大学院博士後期課程修了．博士（工学）．2001 年より，株式会社ウィズダムテックの取締役技術担当に就任し現在に至る．東京理科大学総合研究所客員研究員．GRID コンピュータと人工知能を応用したアプリケーションの研究に興味を持つ．2003 年の人工知能革新応用の国際会議（IAAI2003）にて，優秀論文賞を受賞．



溝口　文雄（正会員）

1968 年東京理科大学大学院修士課程修了，同年同大学理工学部経営工学科助手．1987 年教授となる．工学博士．1994 年 Stanford 大学の Center for the Study of Language and Information（CSLI）の上級研究員となる．株式会社ウィズダムテック取締役．ネットワークセキュリティ，バイオインフォマティクスに関する研究に従事．Artificial Intelligence Journal の論文編集委員．日本ソフトウェア科学会，AAAI，IEEE 各会員．2003 年の人工知能革新応用の国際会議（IAAI2003）にて，優秀論文賞を受賞．