COMPUTERS & SECURITY 86 (2019) 183–205

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Managing attribute-based access control policies in a unified framework using data warehousing and in-memory database

Mahendra Pratap Singh[a], *Shamik Sural*[a,*], *Jaideep Vaidya*[b], *Vijayalakshmi Atluri*[b]

[a] *Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India*
[b] *Management Science and Information Systems Department, Rutgers University, USA*

ARTICLE INFO

Article history:
Received 29 December 2018
Revised 30 May 2019
Accepted 2 June 2019
Available online 12 June 2019

Keywords:
Attribute Based Access Control
Meta-policy
Unified security policy
Authorization
In-memory database
Data warehousing

ABSTRACT

Over the last few years, various types of access control models have been proposed for expressing the growing needs of organizations. Out of these, there is an increasing interest towards specification and enforcement of flexible and dynamic decision making security policies using Attribute Based Access Control (ABAC). However, it is not easy to migrate an existing security policy specified in a different model into ABAC. Furthermore, there exists no comprehensive approach that can specify, enforce and manage ABAC policies along with other policies potentially already existing in the organization as a unified security policy. In this article, we present a unique and flexible solution that enables concurrent specification and enforcement of such security policies through storing and querying data in a multi-dimensional and multi-granular data model. Specifically, we present a unified database schema, similar to that traditionally used in data warehouse design, that can represent different types of access control policies and store relevant policies as in-memory data, thereby significantly reducing the execution time of access request evaluation. We also present a novel approach for combining multiple access control policies through meta-policies. For ease of management, an administrative schema is presented that can specify different types of administrative policies. Extensive experiments on a wide range of data sets demonstrate the viability of the proposed approach.

1. Introduction

Organizations set up access control mechanisms to prevent unauthorized use of their resources, systems and data. In recent years, the requirements of new and emerging applications and a changing environment have led to significant advancements in the field of access control. This has broadened the scope of access control decision dimensions (e.g., security level, category, role, attribute, etc.) and has resulted in several access control models beyond the traditional models like Discretionary Access Control (DAC) (Graham and Denning, 1972) and Mandatory Access Control (MAC) (Bell and LaPadula, 1976). Among these, Role Based Access Control (RBAC) (Sandhu et al., 1996) is the most widely accepted access control model in commercial organizations that groups job functions into roles and provides a mechanism for easy administration of roles. RBAC has been extended in

* Corresponding author.
E-mail addresses: mahoo15@gmail.com (M.P. Singh), shamik@cse.iitkgp.ac.in (S. Sural), jsvaidya@business.rutgers.edu (J. Vaidya), atluri@rutgers.edu (V. Atluri).
https://doi.org/10.1016/j.cose.2019.06.001
0167-4048/

different dimensions (e.g., temporal, Bertino et al., 2001; Joshi et al., 2005; spatial, Damiani et al., 2005; Ray et al., 2006; and spatio-temporal, Aich et al., 2009; Aich et al., 2007; Ray and Toahchoodee, 2007) to constrain the availability of roles and permissions thus enabling fine-grained access to resources.

It may be noted that, not only have the newer access control models been designed to overcome the limitations which earlier models had regarding specification and enforcement of policies, but also to address the evolving access control needs of organizations. However, among those that work towards enhancement, it is mostly the expressive power that has been the target for improvement through specification of fine-grained policies (Committee, 2009), which allows organizations to specify more precisely any policy they want to enforce or accurately comply with regulations (e.g., HIPAA, PCI, etc.). Although highly granular access control seems desirable and their implementation exists in the literature, very few of them have found deployment in real life. This is primarily due to the complexity in specifying, enforcing and managing policies of different levels of granularity through them. Thus, to achieve high granularity, these factors need to be balanced.

ABAC (Hu et al., 2013) is a recent access control model that addresses some of these concerns. ABAC considers dimensions as attributes and controls user's access to resources through policies composed of attributes of various entities, e.g., users, objects and environment. However, administration of ABAC is quite complicated, since visualizing the effect of a change in policy or obtaining the permissions available to a user of a system requires evaluation of a large number of rules.

In essence, RBAC is easily manageable and is widely implemented, whereas ABAC is flexible and more granular but not yet adopted by many organizations like RBAC. Therefore, the ability to combine the evolving ABAC model with traditional RBAC model would overcome both of their drawbacks and enhance policy expressibility. Besides RBAC, being able to specify DAC and MAC in the proposed unified policy would allow organizations to move over from legacy systems to ABAC. It may be noted that, support for diverse types of policies could be implemented either by having two distinct access control models in place or by transforming one policy to the other if at all possible. Furthermore, there can be some resources for which the access needs to be controlled through more than one policy (say, RBAC and ABAC) or to be specified simultaneously at multiple levels of granularity, like DAC, RBAC, etc. Moreover, there may exist certain organization level policies that cannot be specified using a standard access control mechanism. These types of policies are usually implemented as application code and are embedded into the system. Thus, the aforementioned factors further complicate specification, enforcement and maintenance of security policies.

In a panel discussion, Ferraiolo and Atluri (2008) emphasized the necessity for a meta-model to resolve the issue of multiple access control models. In recent years, although several meta-models (Barker, 2009; Bertolissi and Fernandez, 2014; Damianou et al., 2001) have been proposed, they are mainly restricted to the specification of different types of access control policies and to some extent their enforcement. Moreover, those models do not consider the different levels of granularity with respect to specification, the complexity of analyzing the models and enforcement when different features and dimensions are supported. They also do not consider the issue of flexibility which provides freedom to decide the level of access control granularity according to different needs, such as user, resource, situation and application. Therefore, a comprehensive and flexible meta-model is required that allows for (1) specification, (2) evaluation and (3) maintenance of policies at different granularities and dimensions.

Our key insight is that, while multiple dimensions and granularities make effective access control complex when viewed from a security perspective, the issue of managing multi-dimensional, multi-granular (MDMG) data has been effectively addressed by the database community, through the notion of data warehousing schema. Thus, our key challenge is to explore the data warehousing model in this regard, and map the MDMG access control issue to the data warehouse environment for enabling specification and enforcement of flexible access control policies through data warehousing technology. Since data warehouses are general purpose software and not designed specifically for access control, performance may be an issue. Hence, we also use in-memory database (Oracle Technical Committee, 2015) to address those issues effectively.

As an illustrative example, consider a banking system that provides various financial services (such as debit and credit cards, foreign currency transaction, etc.) to its customers through different departments like Branch Banking (BB), Business Banking (BsB), Priority Banking (PB), Forex and Treasury Banking (TxB), etc. These services involve confidential and sensitive information of both the customers and well as the bank. A bank primarily has two types of stakeholders, namely, customer and employee. A branch is headed by an employee of Assistant Vice President (AVP) grade who is designated as the Branch Head (BrH). Each department of a bank is comprised of employees in the grades Assistant Manager (AM), Deputy Manager (DM), Manager (M) and Senior Manager (SM). The BB department is headed by an SM, designated as the Branch Operation Head (BoH). In this department, employees in the grades AM, DM and M are designated as Customer Service Officers (CSO), whereas in the BsB and PB departments, they are designated as Relationship Manager (RM). In the TxB department, employees of the AM, DM, M and SM grades are designated as TxB Customer Service Officer (TCSO). In any branch of the bank, the BsB, PB and TxB departments can be headed by employees of M or SM grades.

In the bank, employees are given access (e.g., read, write, debit, credit, approve, etc.) based on their grade, department and the branch of posting. To carry out daily banking activities, in addition to the access to resources which can belong to the same or another department, employees of the departments mentioned above often require approval or involvement of one or more employees of the same or different departments. Additionally, employees of all grades of the departments can initiate debit transactions which can only be approved by employees of the BB department of those grades who have not initiated them and these are within their allowed approval limits. Similarly, all employees of all grades of these departments can read customer details from the branch of posting, whose creation, updating or deletion can be initiated by employees of any grade and approved by SM grade employees of

the customer's home branch BB department. These requirements involve multiple attributes of various entities (e.g., user, object and environment) that can be MDMG in nature and need to be adequately addressed via meta-policies.

In this article, we present a unified framework which enables specification and enforcement of attribute-based policies (namely, ABAC) along with meta-policies as well as role-based policies (namely, RBAC) and legacy policies (namely, DAC). Additionally, we also present a unified solution for managing all of these security policies. Furthermore, we discuss efficient implementation of the proposed approach in terms of specification, maintenance, enforcement and evaluation of policies through data warehousing schemas, in-memory database and database query language. An initial version of the unified database schema appeared in Singh et al. (2015), where we presented an idea for the unification of access control policies and demonstrated the specification of DAC, MAC, RBAC and TRBAC policies. The novel contributions of the work presented in this article are summarized below:

- First, we develop an extension to the unified database schema presented in Singh et al. (2015) that is capable of specifying ABAC (Hu et al., 2013) policies.
- We also present a novel approach that enables specification of meta-policies comprising multiple policies (e.g., RBAC, ABAC, etc.) as sub-policies. These meta-policies can combine the benefits of different policies and can specify policies for resources that need to be protected under more than one security policy or simultaneously at multiple levels of granularity. Further, to determine the result of a meta-policy, different PRCAs have been proposed.
- Database capabilities, such as data warehousing for MDMG database schema and in-memory database, have been effectively utilized instead of any existing policy specification language to specify and enforce access control policies.
- For easy administration of security policies, an administrative schema is introduced that can specify administrative policies, such as role-based administration model for attributes (Jin et al., 2012a) and ARBAC97 role-based administration of roles (Sandhu et al., 1999).
- Finally, we have performed an extensive experimental study that shows the viability of the proposed approach.

The rest of the paper is organized as follows. Section 2 provides a brief overview of various access control models, their administrative models and data warehouse. The overall system architecture is presented in Section 3. Section 4 describes how ABAC and other different types of administrative policies can be specified. Section 5 explains how relevant access control policies can be stored as in-memory data while Section 6 presents an approach for specifying and enforcing meta-policies using PRCAs. Enforcement of policies for evaluating different types of user and administrative requests is discussed in Section 7. Section 8 presents detailed experimental evaluation of the proposed approach. Finally, Section 9 reviews related work while Section 10 concludes the article and presents directions for future research.

2. Preliminaries

This section presents a brief description of ABAC, RBAC and DAC models, and different types of administrative models, namely, role-based administration model for attributes and ARBAC97. It also gives an overview of data warehousing.

2.1. *ABAC and role-based administration model for attributes*

ABAC (Hu et al., 2013) is well known for flexible policy specification and dynamic decision-making capabilities because it specifies policies using attributes of users, objects and environment. In ABAC, each user and object can have one or more user attributes and object attributes, and each such attribute can have one or more values. Access to objects can be restricted by using environment attributes such as time, location, etc., and each environment attribute can have one or more values. Thus, ABAC grants permissions according to attributes of entities involved in a request.

The role-based administration model for attributes (Jin et al., 2012a) presents an approach for managing the user to user attribute value assignment relation. It includes two main components, namely *can_assign* and *can_delete*. These components capture a triple $(AR, C_{UAV}, 2^{UAV})$, where AR, C_{UAV} and 2^{UAV} represent the set of administrative roles, conditions based on user attribute values and user attribute value ranges, respectively. The relation *can_assign* allows a user of an administrative role to assign user attribute values in the specified set of user attribute values to a user if the user satisfies the condition. Similarly, *can_delete* allows a user of an administrative role to delete a user association in the specified set of user attribute values if the user satisfies the condition.

2.2. *RBAC and ARBAC97*

RBAC (Sandhu et al., 1996) grants access to users on objects through roles. A role represents the set of activities that users can perform in an organization. Each user can be associated with one or more roles either directly or through a role hierarchy. Role hierarchy is a partial order defined on the set of roles that specifies which roles are junior to which other roles. In RBAC, a right on an object is known as a permission. Permissions are assigned to roles, and roles are assigned to users. Thus, RBAC consists of U, R, P, RH, UA and PA representing the set of users, the set of roles, the set of permissions, role hierarchy, users to roles assignments and permissions to roles assignments, respectively.

ARBAC97 (Sandhu et al., 1999) is a role-based administrative model used to manage the user to role assignment (URA), permission to role assignment (PRA) and role to role assignment (RRA) relations in RBAC. ARBAC97 comprises three components, namely, URA97, PRA97 and RRA97. It also contains five types of relations, namely, *can_assign, can_assignp, can_revoke, can_revokep* and *can_modify*. The relations *can_assign* and *can_assignp* capture a triple $(AR, CR, 2^R)$, where AR, CR and 2^R represent the set of administrative roles, prerequisite conditions and role ranges, respectively. The relations *can_revoke, can_revokep* and *can_modify* capture a pair (AR,

2^R), where AR and 2^R represent the set of administrative roles and role ranges, respectively. In URA97, the relation *can_assign* allows a user of an administrative role to assign a user to roles in the role range if the user in regular roles satisfies the condition; and the relation *can_revoke* allows a user of an administrative role to revoke a user's association with the role range. In PRA97, the relation *can_assignp* allows a user of an administrative role to assign permission to roles in the role range if the permission in regular roles satisfies the condition; and the relation *can_revoke* allows a user of an administrative role to revoke a permission association with the role range. In RRA97, the relation *can_modify* allows a user of an administrative role to create a role in the role range, delete a role from the role range, and modify relationship between roles in the role range.

2.3. DAC

DAC (Graham and Denning, 1972) is a user-centric access control mechanism in which object owners decide the permissions that users can have on objects. Those permissions can be captured in a matrix which is known as the access control matrix (ACM). In an ACM, each row represents subjects and each column represents either a subject or an object. The corresponding row and column entries indicate a right that a subject has on an object. Thus, an access request can be directly verified through the ACM. If there exists an entry corresponding to the access request, then the access is granted; otherwise, access is denied.

2.4. Data warehouse

A data warehouse is a collection of integrated, subject-oriented data, where each unit of data is relevant at some moment in time (Chaudhuri and Dayal, 1997). It is a repository of information that is extracted, integrated, summarized and stored from multiple sources in order to support different types of analytical queries. To facilitate complex analysis and visualization, a data warehouse employs multi-dimensional models to capture all the dimensions of interest. Each dimension again may be described by a set of attributes.

The typical data models employed to specify a schema using which multi-dimensional data is stored are *star schema, snowflake schema,* and the *fact constellation schema*. In a star schema, the database consists of a single fact table and one table for each dimension. The dimension tables are usually de-normalized to provide symmetric access to the fact table. Each column of the fact table has a foreign key reference to the primary key of the corresponding dimension table that provides its multi-dimensional coordinates. Each row of the fact table, thus, stores the numeric measures for a combination of those coordinates. A dimension table consists of columns that correspond to all the meaningful attributes of that particular dimension. On the other hand, in a snowflake schema, dimension tables are normalized especially since de-normalization introduces scope for inconsistency, which could have a negative impact on the system integrity. It may, however, be noted that many of the data warehouse extensions/toolkits of commercial database management systems like Oracle, support multi-dimensional visualization of data from star schema only with a well-defined fact table.

In order to meet both these requirements and before going into the detailed discussion of how MDMG access control policies can be specified and evaluated using database technologies, we first present the system architecture in Section 3.

3. System architecture

Fig. 1 shows the overall system architecture of our proposed approach. It comprises a policy enforcement module, policy decision module, policy specification/maintenance module, policy administration module and a policy data warehouse which is named as the policy vault. A detailed description of the functionality of each module, the role they play, and their integration with other modules are given below.

3.1. Policy enforcement module (PEM)

Each access control system comprises a policy enforcement mechanism that can be either embedded in the user interface or at the application hosting server. In our system, PEM is a part of the user interface that prepares an access request in the prescribed format and embeds the information required for evaluating the request. It also forwards an access request to the policy decision module and conveys the outcome of evaluation to the user.

3.2. Policy decision module (PDM)

PDM acts as an interface between the PEM and the policy vault (PV). It is comprised of policy evaluation procedures, which are written in a standard database language (e.g., SQL, PL/SQL, etc.) and used to evaluate access requests. When the PDM receives an access request from the PEM, it first identifies a relevant procedure through meta-policy identification function given in Algorithm 7 and then obtains applicable policy and other entity-related data from the PV for evaluating the request. Finally, PDM conveys evaluation outcome of the request in the predefined format to the PEM.

3.3. Policy specification/maintenance module (PSM)

Modern organizations require multiple access control policies to protect resources which belong to different units. The PSM through unified database schema enables the specification of different types of security policies which are represented at the bottom of Fig. 1. It also enables the specification of constraints at multiple dimensions and granularity levels. In case of any change in policies or unified schema, administrative users reflect the changes in the policy vault by using the PSM.

3.4. Policy administration module (PAM)

PAM acts as an interface between administrative users and the authorization system through which an administrative user can access various modules, namely, PSM, PV and PDM. PAM is similar to PDM and contains administrative policy evaluation procedures which are used to verify administrative user access requests and are written in a suitable database language. Administrative users initiate these requests for performing

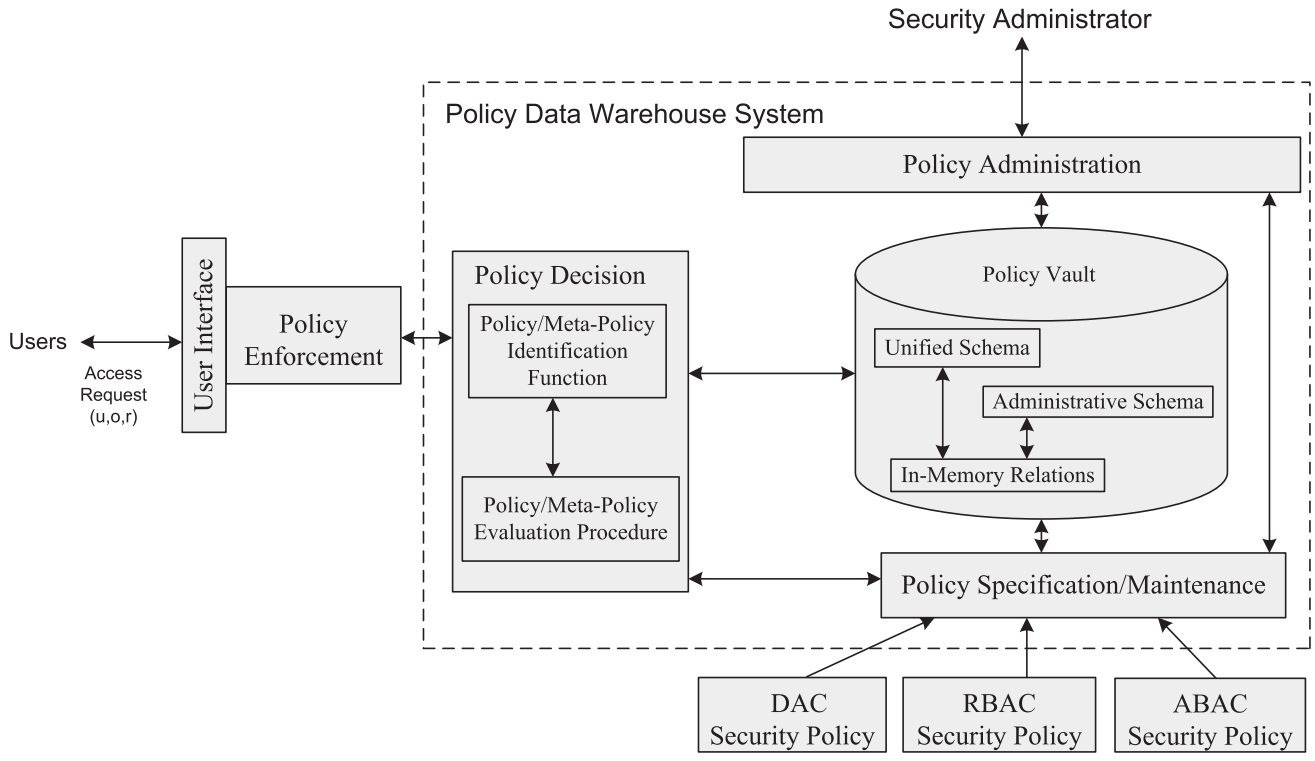

Fig. 1 – System architecture.

various activities, such as incorporating policy update in the unified database schema, deletion of a policy, etc.

3.5. *Policy vault (PV)*

PV acts as a central repository of the proposed architecture and stores different types of security policies. It comprises an administrative schema and a unified database schema that contain relations which are created in data warehouse and in-memory empowered database (e.g., Oracle 12c, etc.). These relations are made up of different types of attributes and capture data related to policies, constraints, and several entities (e.g., user, object, etc.). The relations which capture policy information are marked as in-memory enable and are stored in the main memory of the system along with disk-based storage.

The various components inside the box labeled as the "policy data warehouse system" show how database technologies such as data warehousing schemas, in-memory database and database languages can be meaningfully used to represent access control systems. We develop a suitable database schema for supporting various access control models as described in the following sections.

4. Policy specification

In this section, a unified database schema is presented for specifying ABAC policies along with RBAC and DAC policies. Furthermore, as access control models have notions of administrative models to enable decentralized administration and delegation of administrative responsibilities, a unified administrative schema is also presented that can specify different types of administrative policies.

4.1. *Unified database schema*

A unified database schema is developed for the specification of various access control policies in a consistent and integrated manner, as shown in Figs. 2 and 3. It may be noted that the entire integrated schema has been divided into separate figures for ease of understanding. A common set of notations is used to unify ABAC, RBAC and DAC policies. The relation *User* captures the set of users, and the term *user* is used to indicate a subject in DAC, as well as a user in ABAC and RBAC. The relations *Object* and *Right* capture the set of resources (that need to be protected) and access rights (e.g., read, write, delete, etc.), respectively. Similarly, rights on objects are defined as permissions in RBAC, whereas the term *right* denotes possible modes of access over an object in ABAC and DAC. The details regarding specification of various access control policies introduced in Sections 2.1–2.3 when using the unified database schema are provided below.

4.1.1. *Specification of ABAC policies*

In Fig. 2, we present the ABAC schema. The relation *User_attribute* captures the attributes of users, whereas the relation *User_attribute_value* captures the values of user attributes. Each user can have one or more user attribute values and the relation *User_UAV_assignment* captures the user to user attribute value assignments. Similarly, the relations *Object_attribute* and *Object_attribute_value*, respectively,

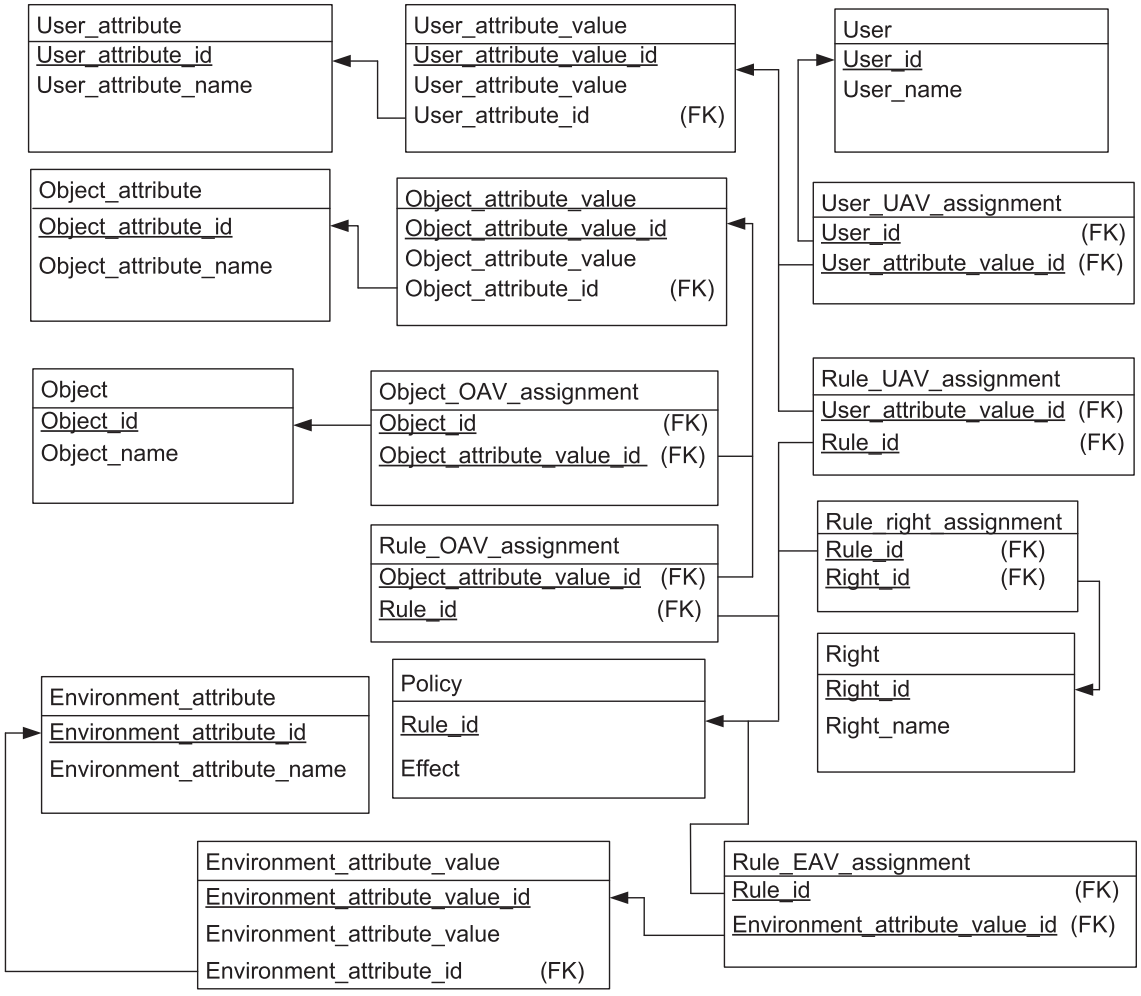

Fig. 2 – Schema for representing ABAC policies.

Table 1 – User attribute.

User_attribute_id	User_attribute_name
1	Grade
2	Designation
3	Department

Table 2 – Object attribute.

Object_attribute_id	User_attribute_name
1	Object type
2	Department

capture the attributes of objects and the values of object attributes, while the relation *Object_OAV_assignment* captures the object to object attribute value assignments. The *Environment_attribute* and *Environment_attribute_value* relations capture the environment attributes (such as time, location, etc.) and their values, respectively. The relations *Policy, Rule_UAV_assignment, Rule_OAV_assignment, Rule_EAV_assignment* and *Rule_right_assignment*, respectively, capture the set of rules, the association of rules with user attribute values, the association of rules with object attribute values, the association of rules with environment attribute values, the association of rules with rights, and are used for representing the ABAC policies. In ABAC, permissions are granted according to various attributes of entities involved in an access request. Thus, no separate fact table is required for modeling ABAC.

ABAC specification for the banking system described in Section 1 through the relations shown in Fig. 2 is as follows. Tables 1 and 2 capture user attributes and object attributes, whereas Tables 3 and 4 capture user attribute values and object attribute values, respectively. Environment attributes and their values are captured in Tables 5 and 6. As an example, consider one of the access control requirements of the banking system, which states that a *user #u*, who is in *Assistant Manager* grade in the *Branch Banking* department, through the role *Customer Service Officer* can *initiate* a transaction during *Working hours* from the *Branch of posting*. These requirements are captured as a policy in Tables 7–11.

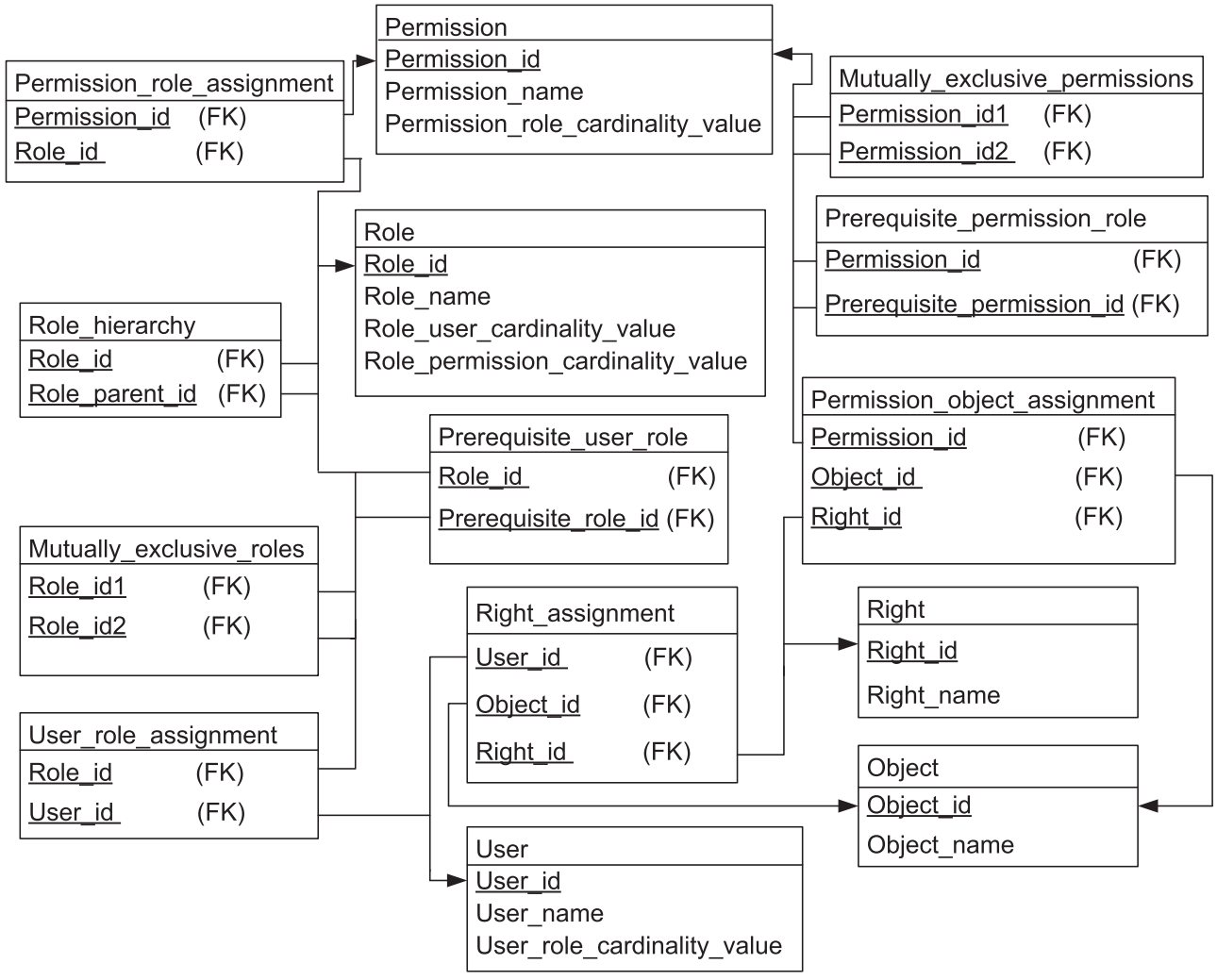

Fig. 3 – Schema for representing DAC and RBAC policies.

Table 3 – User attribute value assignments.

User_attribute_value_id	User_attribute_value	User_attribute_id
1	Assistant Manager	1
2	Deputy Manager	1
3	Manager	1
4	Senior Manager	1
5	Assistant Vice President	1
6	Customer Service Officer	2
7	Relationship Manager	2
8	TxB Customer Service Officer	2
9	Branch Operation Head	2
10	Branch Head	2
11	Branch Banking	3
12	Business Banking	3
13	Priority Banking	3
14	Forex and Treasury Banking	3

4.1.2. *Specification of RBAC policies*

The proposed unified schema is capable of specifying the access control policies supported by RBAC0, RBAC1 and RBAC2 models. In Fig. 3, the relations *User, Role, Permission, User_role_assignment, Permission_role_assignment* and *Permission_object_assignment* are used to specify RBAC policies. The relation *Permission_object_assignment* captures the permissions in the form of rights on the objects. The user role assignment (UA) and permission role assignment (PA) components of RBAC are represented by the *User_role_assignment*

Table 4 – Object attribute value assignments.

Object_attribute_value_id	Object_attribute_value	Object_attribute_id
1	Saving Account	1
2	Customer Detail	1
3	Transaction	1
4	Branch Banking	2
5	Business Banking	2
6	Priority Banking	2
7	Forex and Treasury Banking	2

Table 5 – Environment attribute.

Environment_attribute_id	Environment_attribute_name
1	Working hours
2	Branch of posting

and *Permission_role_assignment* relations, respectively. In RBAC, each user can be associated with one or more roles and each role can have one or more permissions. A single permission can correspond to one or more rights on different objects. The relations *User_role_assignment*, *Permission_role_assignment* and *Permission_object_assignment* act as the fact tables. The relation *Role_hierarchy* represents the partial order relation defined on the set of roles called the role hierarchy.

RBAC enforces a variety of constraints in the specification of various types of access control policies. The most important constraints are Separation of duty (SoD), cardinality and pre-requisites. SoD is enforced through mutually exclusives roles which are captured in the *Mutually_exclusive_roles* relation. The proposed schema supports four types of cardinality constraints. These are: *Role_user_cardinality_value* (maximum number of users that each role can have) in the role relation, *Role_permission_cardinality_value* (maximum number of permissions that each role can have) in the role relation, *User_role_cardinality_value* (maximum number of roles that each user can have) in the user relation and *Permission_role_cardinality_value* (maximum number of roles that each permission can have) in the permission relation. Finally, the pre-requisites of roles for users and permissions are captured in the *Prerequisite_user_role* and *Prerequisite_permission_role* relations, respectively.

For RBAC specification of the banking system described in Section 1, Tables 12,13,14 and 15 capture users, roles, objects and access rights, respectively. Tables 18 and 19 capture user to role assignments and permission to role assignments. The permissions are captured in Table 16.

Table 7 – Policy.

Rule_id	Effect
1	Permit
2	Permit

Table 8 – Rule UAV assignment.

Rule_id	User_attribute_value_id
1	1
1	6
1	11

Table 9 – Rule OAV assignment.

Rule_id	Object_attribute_value_id
1	3

Table 10 – Rule EAV assignment.

Rule_id	Environment_attribute_value_id
1	1
1	3

4.1.3. *Specification of DAC policies*

In Fig. 3, the relations *User*, *Object* and *Right* are used for representing the DAC policies. The relation *Right_assignment* captures DAC policies. All the basic rights supported by the system are captured in the relation *Right*. In the data warehousing terminology, the relations *User*, *Object* and *Right* serve as dimensions, whereas the relation *Right_assignment* acts as a fact table.

Table 6 – Environment attribute assignments.

Environment_attribute_value_id	Environment_attribute_name	Environment_attribute_id
1	09: 00 AM–07: 00 PM	1
2	NITK Campus	2
3	IIT KGP Campus	2

Table 11 – Rule right assignment.

Rule_id	Right_id
1	4

Table 12 – User.

User_id	User_name
1	U1
2	U2
3	U3
4	U4
5	U5
6	U6
7	U7

Table 13 – Role.

Role_id	Role_name
1	Customer Service Officer
2	Relationship Manager
3	TxB Customer Service Officer
4	Branch Head
5	Branch Operation Head

Table 14 – Object.

Object_id	Object_name
1	O1
2	O2
3	O3

Table 15 – Access right.

Right_id	Right_name
1	Read
2	Write
3	Approve
4	Initiate

Table 16 – Permission.

Permission_id	Object_id	Right_id
1	1	1
2	1	2
3	2	3
4	2	4
5	3	5

Table 17 – Right assignment.

User_id	Object_id	Right_id
1	1	1
2	1	2
3	2	3
4	2	4
5	3	5

Table 18 – User role assignments.

User id	Role id
1	1
2	1
3	1
4	2
5	3
6	4
7	5

Table 19 – Permission role assignments.

Role	Permission
1	1
1	2
1	3
1	4
2	3
2	5
3	3
3	5

Table 17 captures DAC specification of access requirements of the same banking system.

4.2. *Unified administrative schema*

In this subsection, we present a unified administrative schema for specifying administrative policies introduced in Sections 2.1 and 2.2. It may also be noted that, ABAC and RBAC support role-based administrative models, whereas no explicit administrative model is supported in DAC. Hence, Fig. 4 illustrates the role-based administrative schema.

The relations *User, Role* and *User_attribute_value* shown in Fig. 4 are the same as the corresponding ones shown in Figs. 2 and 3. However, another disjoint set of roles meant for administrating users of an RBAC system is specified using the relation *Adminrole*. The valid administrative permissions are contained in the relation *Adminpermission*. Assignment of administrative users to administrative roles is done using the relation *User_adminrole_assignment*, while permissions belonging to the various administrative roles are maintained in the relation *Adminpermission_adminrole_assignment*.

4.2.1. *Specification of administrative policies for ABAC*

In Fig. 4, the relations *Can_assign_UAV_user* and *Can_delete_UAV_user* capture policies for administrative roles that can assign various ranges of user attribute values to users and delete various ranges of user attribute values from users, respectively. Similarly, the relations *Can_assign_OAV_object* and *Can_delete_OAV_object* can be designed to capture

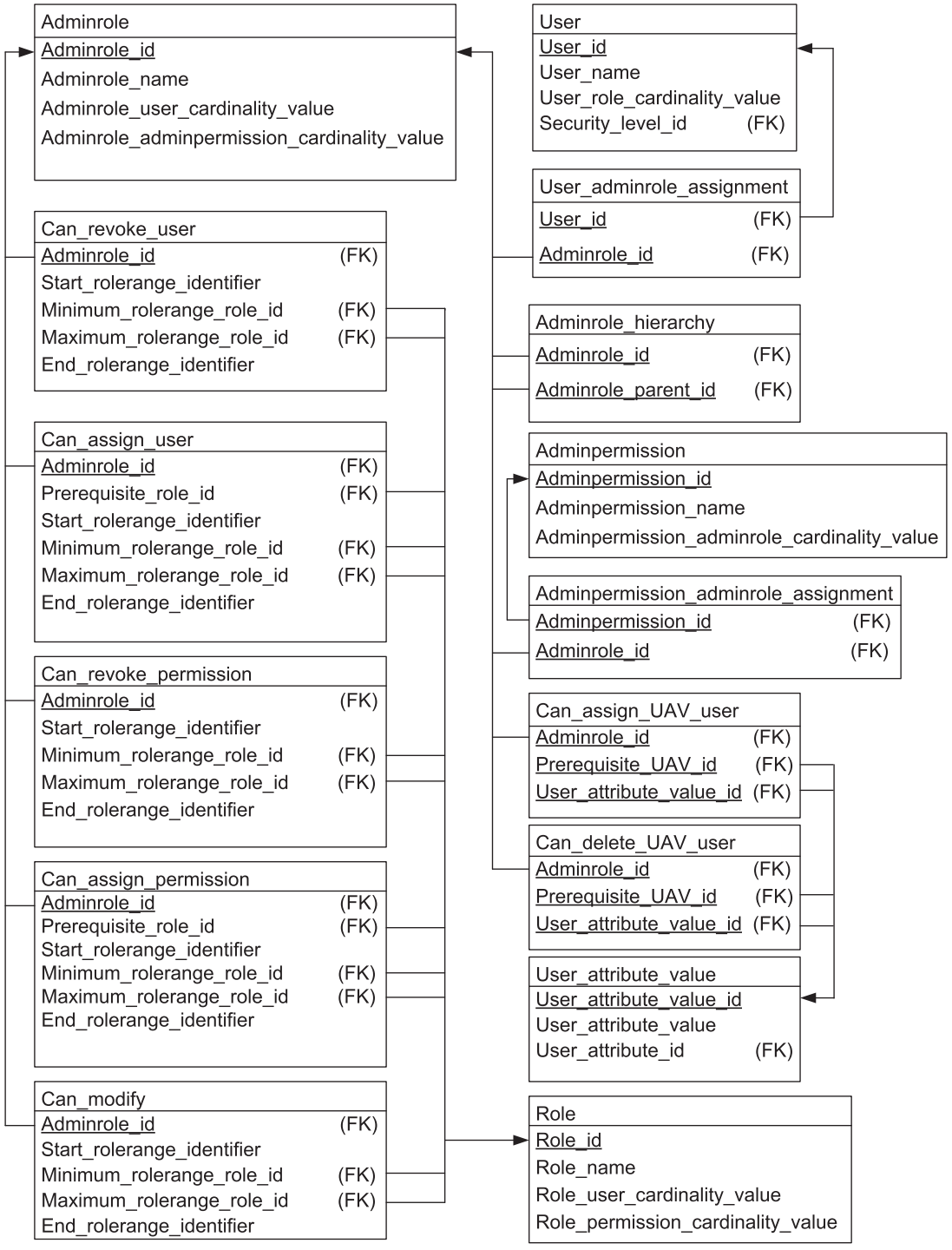

Fig. 4 – Schema for representing role-based administrative policies for ABAC and RBAC.

policies for administrative roles that can assign various ranges of object attribute values to objects and delete various ranges of object attribute values from objects, respectively.

4.2.2. *Specification of ARBAC97 policies*

The basic functions supported by the administrative RBAC model include *Can_assign*, *Can_assignp*, *Can_revoke*, *Can_revokep* and *Can_modify*. The relations *Can_assign_user* and *Can_assign_permission* capture relevant policies for which administrative roles can make changes to *User_role_assignment* and *Permission_role_assignment* relations, respectively, for various ranges of regular roles. The ability of various administrative roles to revoke user's membership from roles is specified in the relation *Can_revoke*. Similarly, the ability of various administrative roles to revoke permissions from roles is specified in the relation *Can_revokep*. Finally, the relation *Can_modify*

maintains the mapping between various administrative roles and user role ranges whose role hierarchy they can administer.

Using the administrative policies specified through the set of relations shown in Fig. 4, the relations relevant to the unified database schema can be suitably maintained. For example, the role hierarchy with the *Manager* being a role senior to the *Deputy Manager* can be specified using the relation *Role_hierarchy* of Fig. 3. If there is an administrative role, say the Senior Security Officer (SSO), which has the authority to assign or remove users over a role range covering the *Manager* and the *Deputy Manager*, then the same is specified in the relations *Can_assign_user* and *Can_revoke*. Once such a role-based administrative policy has been set up, any user belonging to the administrative role of SSO can perform the necessary addition and deletion of users to the roles of *Manager* and *Deputy Manager*.

Thus, the proposed unified database schema is extensible and can easily specify ABAC, RBAC and DAC policies. The unified administrative schema simplifies maintenance of the unified security policies. In the next section, we describe use of in-memory database storage and processing capabilities that enables quick access to policy data from the unified database schema.

5. In-memory specification of policies

In this section, we present an approach for grouping the relations into two broad categories and also demonstrate how in-memory storage of critical relations helps in reducing the execution time of access requests.

5.1. *Categorization of relations based on evaluation performance*

The schemas shown in Figs. 2 and 3 mainly consist of three different types of relations: (1) relations that store attributes of entities (e.g., user, object, role, etc.), (2) relations that capture constraints (e.g., mutually exclusive, cardinality, etc.), and (3) relations that capture policy data and are involved in evaluating access requests (e.g., *Right_Assignment*, etc.). These relations can be grouped into two broad categories, namely, critical and non-critical, according to access request evaluation performance. Relations that are involved in the evaluation of access requests are considered to be critical, whereas the relations which are not directly involved in the evaluation of access requests are considered to be non-critical. Thus, the aforementioned first and second types of relations are classified as non-critical, whereas the third type is classified as critical.

5.2. *Reduction in access-request evaluation time through in-memory storage of critical relations*

Execution time of access requests depends on the policy identification and enforcement mechanism. In most cases, access control mechanisms use an approach similar to XACML for identifying applicable policies. In XACML, a policy is defined as a set of rules, and a rule can be composed of one or more authorization conditions that a request must satisfy for performing a specific action on an object. Moreover, for determining whether there is a policy and if so, which rule in that policy applies to a request, XACML associates a target with each policy and rule that is composed of user attributes, object attributes and a right. In XACML, although all policies are not applicable to each request, still each policy target is compared with the request until the applicable policy is found or the last policy is compared. In contrast, we use database products for specifying, enforcing and managing security policies that are not designed by keeping access control terminologies in mind. Hence, storage/specification of policies as data in relations and retrieval/enforcement of policies for evaluating access requests through database language can have certain performance overhead. So, for addressing the aforementioned issues, we present a solution that not only helps in avoiding irrelevant policy comparison but also enables efficient utilization of database capabilities for identification and enforcement of policies.

The schemas shown in Figs. 2 and 3 are capable of specifying different types of access control policies that are composed of multiple attributes and are specified in one or more relations. Thus, enforcement of policies/evaluation of access requests through those relations is similar to executing database analytic queries. To ensure faster access to security policies, we also store the performance-critical relations in-memory that enables quick retrieval of a set of attributes or a particular attribute through the relation(s) as follows.

1. First, a memory chunk, which contains the relation, is located in-memory by using meta-data. A meta-data acts as an index for a memory chunk and captures various types of information, such as the names of the specified relation and its columns (attributes), mapping of the captured values in those columns (attributes) with the actual rows (row-ids) of the relation, and location (address) of the chunk in the memory.
2. Next, in the memory chunk, it is verified whether the required value falls within the index range of the concerned column (attribute). If the value is within the range, then the cell which stores the value is searched and its position (row-id) is noted. Otherwise, the memory chunk is discarded, and Steps 1 and 2 are repeated with other relevant memory chunks, if any exists.
3. To retrieve values from the other columns (attributes) of the same memory chunk, the position, which is obtained in Step 2, is directly referred in those columns because they store the values in the order of row-id.

It may be noted that, though several relations are stored in-memory, the first step helps in quickly locating the relevant relation (memory chunk). The second step assists in discarding the irrelevant comparisons that significantly reduces the search space. Finally, the third step enables the retrieval of the values of the remaining attributes of the relation in a single comparison.

Fig. 5 illustrates in-memory storage of the relation *right_assignment* which captures the DAC policies and comprises *user, object* and *right* attributes (columns). The relation is too large to fit into a single memory chunk. Therefore, the

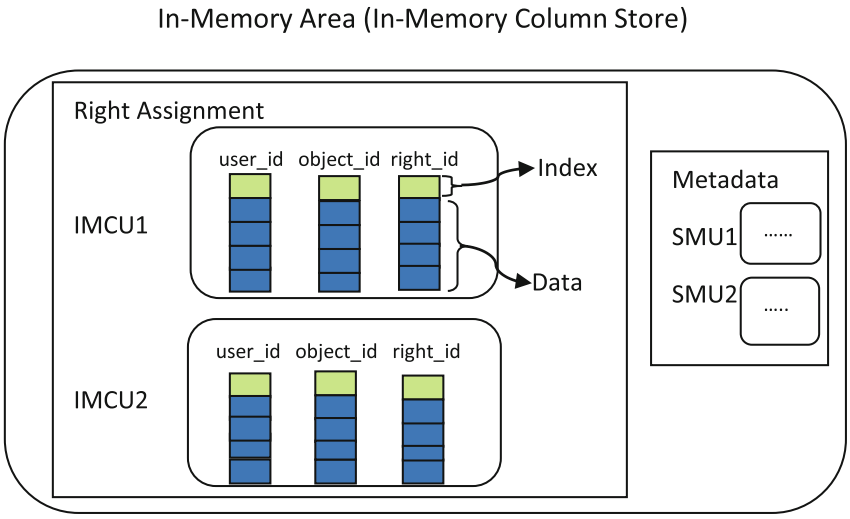

Fig. 5 – In-memory storage of the relation Right_Assignment.

relation is stored in two memory chunks named as IMCU1 and IMCU2, and for each memory chunk, a meta-data is also created named as SMU1 and SMU2. Thus, for evaluating a user (#*u*) specific type of access (#*r*) on an object (#*o*), the policy decision module of Fig. 1 executes the following enforcement query.

SELECT Count (*) FROM right_assignment
WHERE user_id = #*u* AND object_id = #*o* AND right_id = #*r*;

The enforcement query first refers to the SMU in meta-data and identifies the memory chunk IMCU1 which contains a part of the relation *right_assignment*. It then verifies whether the user id (#*u*) falls within the index range of the *user_id* column of IMCU1. If it is, then the position that stores the user (#*u*) is captured, and the object (#*o*) and right (#*r*) are, respectively, searched in the object and right columns at that position. If the object and the right exist, then the access is granted. Otherwise, IMCU1 (entire set of values) is discarded and the whole process is repeated with IMCU2.

Thus, in-memory storage of policies helps in reducing the execution time of access requests. In the next section, we introduce an approach for specifying and evaluating meta-policies.

6. Meta-policy: an approach for combining different types of access control policies

In recent years, combining the benefit of different types of access control policies has gained increasing importance as a topic of research (Fatima et al., 2016; Kuhn et al., 2010; Servos and Osborn, 2017). Several customized approaches have been proposed that either modify RBAC to incorporate ABAC (Huang et al., 2012; Jin et al., 2012b; Qi et al., 2016; Rajpoot et al., 2015a; 2015b) or attempt to infer the final result of policies by combining their common attributes (Li et al., 2009; Rao et al., 2009). Generally, customized approaches put efforts towards unification of access control policies, whereas general purpose languages (like FAF Jajodia et al., 2001, etc.) enable specification and enforcement of flexible policies. Most of these approaches either complicate specification, implementation, administration and security analysis of policies or do not have administrative models for managing them. In contrast, we present a unique approach which is simple yet powerful and can easily combine various types of access control policies through specification of meta-policies. The proposed approach makes use of databases and query processing languages for capturing and maintaining policies and hence assumes the availability of an underlying database management system, which is its potential limitation.

In this section, we first define the different types of meta-policies and present our approaches for obtaining the final results from them. Moreover, we also demonstrate specification and evaluation of different types of meta-policies through the unified database schema.

6.1. *Specification and evaluation of meta-policies*

A meta-policy is defined as a combination of security policies, which contains multiple security policies as sub-policies. Let us assume that there exist *n* policies which are represented as $sp_1, sp_2, \ldots, sp_n$. These security policies can be MDMG, which indicates that the policies can contain multiple dimensions and each dimension can have different levels of granularity. Depending on the need of the secure access to be ensured, these security policies can be broadly grouped into two categories, namely, liberal and strict, which grant access according to the outcome of any one policy and all policies, respectively. For evaluating the final results of these meta-policies, we present two types of PRCAs as follows.

6.1.1. *Allow access if any sub-policy allows access*

This approach is suitable for evaluating the final result of a liberal meta-policy that grants access according to the outcome of *any one* sub-policy. The approach "ORs" the results of individual sub-policies and permits access if the result of at least one sub-policy is true. Thus, the final result of the approach can be represented as (sp_1 OR sp_2 OR ... OR sp_n).

Algorithm 1: Evaluate_if_any_subpolicy ($Ar = (u, o, r)$, $P = \{sp_1, sp_2, \ldots, sp_n\}$).

```
result ← false
for each sub − policy (sp) ∈ P do
    result ← evaluate_subpolicy(sp, Ar)
    if (result == true) then
        return true
        break
    end
end
return false
```

Algorithm 1 evaluates an access request through the aforementioned approach. The algorithm takes a meta-policy, which consists of multiple sub-policies, and an access request as input. Each sub-policy and the access request are then passed to a function *evaluate_subpolicy*. If the function returns true, then the access is granted. Otherwise, the access is denied.

6.1.2. *Allow access if all sub-policies allow access*

This approach is the opposite of the previous approach and is used to evaluate the final results of a strict meta-policy that grants access according to the outcomes of *all* sub-policies. The approach "ANDs" the results of sub-policies and permits access if each sub-policy outcome is true. Thus, the final result of the approach can be represented as (sp_1 AND sp_2 AND ... AND sp_n).

Algorithm 2, which is based on the aforementioned approach, also takes a meta-policy and an access request as input. Each sub-policy and the access request are then passed to the function *evaluate_subpolicy*. If the function returns true, then the same process is repeated until the last sub-policy is evaluated. Otherwise, the algorithm terminates the evaluation of the remaining sub-policies and returns false (denies access).

Algorithm 2: Evaluate_if_all_subpolicies ($Ar = (u, o, r)$, $P = \{sp_1, sp_2, \ldots, sp_n\}$).

```
count ← 0
result ← false
sp_num ← number of sub − policies ∈ P
for each sub − policy (sp) ∈ P do
    result ← evaluate_subpolicy(sp, Ar)
    if (result == true) then
        count ← count + 1
    else
        break
    end
end
if (count == sp_num) then
    return true
end
return false
```

The first and second approaches are, respectively, similar to permit-override and deny-override algorithms of XACML.

6.2. *Specification of meta-policies through unified database schema*

The proposed unified database schema shown in Figs. 2 and 3 is capable of specifying various types of security policies, namely, ABAC, RBAC and DAC. For simplicity, the aforementioned policies are represented as sp_1, sp_2 and sp_3, respectively. These security policies have limited decision making dimensions, and individually, they are not capable of specifying and enforcing one or more security requirements, such as information flow control, fine-grained access control, ease of policy management, etc. Hence, combining one security policy with another can help in overcoming their individual drawbacks. Here, combining two security policies (say, RBAC and ABAC) means that either an access control requirement is specified through the two policies or one of the policy specifies the access control element and the other defines a condition for ensuring fine-grained access. Such combinations also enable specification of MDMG security policies that consider multiple dimensions (e.g., role, time, location, security level, etc.) and different levels of granularity of dimensions (e.g., year, hour, minute, second, etc.).

For specifying and evaluating meta-policies from the unified database schema, we make use of different types of security policies and PRCAs. Then, as per the need of protection, we group security policies and PRCAs broadly into four categories to enable specification and evaluation of different types of meta-policies, which are as follows.

6.2.1. *Single policy combination and single PRCA*

This approach enables specification of a meta-policy which consists of a single combination of policies (e.g., *sp*1 and *sp*2) and a single PRCA (e.g., allow access if all sub-policies allow access). Algorithm 3 presents this approach, which takes an access request as input. To evaluate the access request, it uses the function *evaluate_if_all_subpolicy* given in Algorithm 2. The function takes a meta-policy and an access request as input and returns true, if the access is granted.

Algorithm 3: Single policy combination and Single PRCA ($Ar = \{u, o, r\}$).

```
result ← false
P = {sp1, sp2}
result ← evaluate_if_all_subpolicy(Ar = (u, o, r),
P = {sp1, sp2})if (result == true) then
    return true
end
return false
```

As an example, let there be an access control requirement of the banking system described in Section 1, which states that a user *#u* of the *Manager* grade in the *Branch Banking* department, through the role *Customer Service Officer*, can perform *read* and *write* operations on an object *#o* during working hours from the branch of posting. To specify the requirement as a policy, an access control mechanism (like RBAC) would define a part of the specification as role-based policy and the rest as application code, whereas ABAC would specify the entire requirements as an attribute-based policy.

However, both of these access control mechanisms have their own advantages and disadvantages. In contrast, our approach can specify the aforementioned requirement in any one of the following forms through specification of a meta-policy.

PRCA: Allow access if all sub-policies allow access

RBAC Policy: UR = (u, Customer Service Officer), PR = (Customer Service Officer, Perm), Perm = (o, {read, write})

ABAC Policy: P_{uav} = (P, {Customer Service Officer, Manager, Branch Banking}), P_{oav} = (P, Saving account), P_r = (P, {read, write}), P_{eav} = (P, {Working hour, Branch of posting})

OR

PRCA: Allow access if all sub-policies allow access

RBAC Policy: UR = (u, Customer Service Officer), PR = (Customer Service Officer, Perm), Perm = (o, {read, write})

Attribute-Based Condition: C_{uav} = (C, {Manager, Branch Banking}), C_{oav} = (C, Saving account), C_{eav} = (C, {Working hours, Branch of posting})

The first form specifies requirements as RBAC as well as ABAC policies, whereas the second form specifies essential elements as RBAC policy and ensures fine-grained access through an attribute-based condition. In the aforementioned meta-policy, PRCA ensures enforcement of both *RBAC* and *ABAC* policies or *RBAC* policy and *Arribute-Based Condition* on an access request. In the RBAC policy, UR, PR and Perm represent the user to role assignments, permission to role assignments and the set of all permission. These are stored in the relations *user_role_assignment, permission_role_assignment* and *permission_object_assignment*, respectively. In the ABAC policy, P_{uav}, P_{oav}, P_r and P_{eav} represent policy to user attribute value assignments, policy to object attribute value assignments, policy to right assignments and policy to environment attribute value assignments. These are stored in the relations *rule_uav_assignment, rule_oav_assignment, rule_right_assignment* and *rule_eav_assignment*, respectively. In the attribute based condition, C_{uav}, C_{oav} and C_{eav} represent the conditions for user attribute value assignments, object attribute value assignments and environment attribute value assignments. These are stored in the relations *rule_uav_assignment, rule_oav_assignment* and *rule_eav_assignment*, respectively.

6.2.2. *Single policy combination and multiple PRCAs*

Similar to the previous approach, this approach uses a single combination of policies (e.g., sp_1, and sp_2) and enables specification of flexible meta-policies through multiple PRCAs (e.g., allow access if any sub-policy allows access, allow access if all sub-policies allow access). These meta-policies protect some of the resources through a set of policies and the remaining resources through another set of policies.

Algorithm 4 is based on the aforementioned approach, which contains two meta-policies (procedures) named as *MP1* and *MP2* that use the functions *evaluate_if_any_subpolicy* and *evaluate_if_all_subpolicy* given in Algorithm 1 and Algorithm 2, respectively. These functions take a set of sub-policies and a request as input and return true, if the access is permitted. To identify the applicable meta-policy, the algorithm uses the Meta-policy Identification (MI) function given in Algorithm 7. *MI* takes a request as input and returns a meta-policy, if any exist. Then, the request is evaluated through that meta-policy.

Algorithm 4: Single policy combination and Multiple PRCAs ($Ar = \{u, o, r\}$).

```
result ← false
Procedure ← MI(Ar, R_poa, R_ooav)
procedure MP1(Ar)
    P = {sp1, sp2}
    result ←
evaluate_if_any_subpolicy(Ar = (u, o, r), P = {sp1, sp2})
if (result == true) then
    return true
    end
end procedure
procedure MP2(Ar)
    P = {sp1, sp2}
    result←
evaluate_if_all_subpolicy(Ar = (u, o, r), P = {sp1, sp2})
if (result == true) then
    return true
    end
end procedure
return false
```

As an example, consider the other three access control requirements of the banking system described in Section 1. The first requirement states that a user *#u*, who is a *Manager* in the *Branch Banking* department, through the role *Customer Service Officer*, can *approve* a transaction *#Tx* which is not initiated by that user and is within her approval limit, during working hours from the branch of posting. The second requirement states that a user *#u* through the role *Relationship Manager* can *initiate* a transaction *#Tx*. The third requirement states that a user *#u*, who is a *Manager* in the *Forex and Treasury Banking* department, through the role *TxB Customer Service Officer*, can *initiate* a transaction *#Tx* during working hours from the branch of posting. These requirements are captured through specification of meta-policies as follows.

Meta-Policy MP1:

PRCA: Allow access if all sub-policies allow access

RBAC Policy: UR = (u, Customer Service Officer), PR = (Customer Service Officer, Perm), Perm = (Tx, {approve})

ABAC Policy: P_{uav} = (P, {Customer Service Officer, Manager, Branch Banking}), P_{oav} = (P, Transaction), P_r = (P, {approve}), P_{eav} = (P, {Working hours, Branch of posting, Not initiated transaction, Within approval limit})

Meta-Policy MP2:

PRCA: Allow access if any sub-policy allows access

RBAC Policy: UR = (u, Relationship Manager), PR = (Relationship Manager, Perm), Perm = (Tx, {initiate})

ABAC Policy: P_{uav} = (P, {TxB Customer Service Officer, Manager, Forex and Treasury Banking}), P_{oav} = (P, Transaction), P_r = (P, {initiate}), P_{eav} = (P, {Working hours, Branch of posting})

The meta-policies MP1 and MP2 are composed of ABAC and RBAC policies that are captured in the schemas shown in Figs. 2 and 3, respectively. MP1 specifies the first requirement and allows those users to *approve* the transaction *#Tx* who possess both role and attribute values mentioned in RBAC and ABAC policies, respectively. In contrast, MP2 specifies the second and third requirements and allows those users to ini-

tiate the transaction #*Tx* who possess either the role or the attribute values specified in the RBAC and ABAC policy, respectively.

6.2.3. *Multiple policy combinations and single PRCA*

This approach enables specification of meta-policies which contain multiple combinations of policies (e.g., sp_1 and sp_2, sp_1 and sp_3, etc.) and a single PRCA (e.g., allow access if all sub-policies allow access).

Algorithm 5 is based on the aforementioned approach,

Algorithm 5: Multiple policy combinations and Single PRCA ($Ar = \{u, o, r\}$).

```
result ← false
Procedure ← MI(Ar, R_poa, R_ooav)
procedure MP3(Ar)
    P = {sp1, sp2}
    result ←
evaluate_if_all_subpolicy(Ar = (u, o, r), P = {sp1, sp2})
if (result == true) then
    return true
end
end procedure
procedure MP4(Ar)
    P = {sp1, sp3}
    result ← evaluate_if_all_subpolicy (Ar = (u, o, r),
P = {sp1, sp3})
if (result == true) then
    return true
end
end procedure
return false
```

which comprises two meta-policies (procedures) named as *MP3* and *MP4* that use the function *evaluate_if_all_subpolicy* given in Algorithm 2. In *MP3* and *MP4*, the function takes a different set of sub-policies and an access request as input and returns true, if the access is permitted. To find the relevant meta-policy, the algorithm takes an access request as input and passes it to the function *MI* given in Algorithm 7. If the function returns a meta-policy, then the access request is verified through that meta-policy. Otherwise, the access is denied.

As an example, consider the two access control requirements of the banking system described in Section 1. The first requirement states that a user #*u*, who is a *Manager* in the *Forex and Treasury Banking* department, through the role *TxB Customer Service Officer*, can perform *read* operation on an object #*o* during working hours from the branch of posting. The second requirement states that a user #*u*, who is a *Manager* in the *Business Banking* department, through the role *Relationship Manager*, can perform *read* operation on an object #*o* during working hours from the branch of posting, and the user #*u* has *read* access right on the object #*o*. These requirements are captured through specification of meta-policies as follows.

Meta-Policy MP3:

PRCA: Allow access if all sub-policies allow access

RBAC Policy: UR = (u, TxB Customer Service Officer), PR = (TxB Customer Service Officer, Perm), Perm = (o, {read})

ABAC Policy: P_{uav} = (P, {Manager, Forex and Treasury Banking}), P_{oav} = (P, Saving account), P_r = (P, {read}), P_{eav} = (P, {Working hours, Branch of posting})

Meta-Policy MP4:

PRCA: Allow access if all sub-policies allow access

ABAC Policy: P_{uav} = (P, {Relationship Manager, Manager, Business Banking}), P_{oav} = (P, Saving account), P_r = (P, {read}), P_{eav} = (P, {Working hours, Branch of posting})

DAC Policy: (u, o, {read})

The meta-policies MP3 and MP4 are composed of RBAC and ABAC, and ABAC and DAC policies, respectively, that are captured in the schemas shown in Figs. 2 and 3. MP3 specifies the first requirement and allows those users to *read* the object #*o* who possess appropriate role and attribute values mentioned in RBAC and ABAC policy, respectively. In contrast, MP4 specifies the second requirement and allows those users to *read* the object #*o* who possess attribute values specified in the ABAC policy and satisfy the DAC policy.

6.2.4. *Multiple policy combinations and multiple PRCAs*

Unlike the aforementioned approach, this approach allows specification of meta-policies that use multiple combinations of policies (e.g., sp_1 and sp_2, sp_1 and sp_3, etc.) and multiple PRCAs (e.g., allow access if all sub-policies allow access, allow access if any sub-policy allows access, etc.) for obtaining the final results of the meta-policies.

Algorithm 6 is based on the aforementioned approach, which comprises two different meta-policies (procedures) named as *MP5* and *MP6* that use the functions *evaluate_if_all_subpolicy* and *evaluate_if_any_subpolicy* given in Algorithms 2 and 1, respectively. These functions take a different set of sub-policies and a request as input and return false, if the access is not permitted. To identify the applicable meta-policy, the algorithm takes an access request as input and passes it to the function *MI* given in Algorithm 7 . If the

Algorithm 6: Multiple policy combinations and Multiple PRCAs ($Ar = \{u, o, r\}$).

```
result ← false
Procedure ← MI(Ar, R_poa, R_ooav)
procedure MP5(Ar)
    P = {sp1, sp2}
    result ←
evaluate_if_all_subpolicy(Ar = (u, o, r), P = {sp1, sp2})
if (result == true) then
    return true
end
end procedure
procedure MP6(Ar)
    P = {sp1, sp3}
    result ← evaluate_if_any_subpolicy(Ar = (u, o, r),
P = {sp1, sp3})
if (result == true) then
    return true
end
end procedure
return false
```

Algorithm 7: Meta-policy Identification (MI) ($Ar = (u, o, r)$, R_{poa}, R_{ooav}).

```
object_attribute_values[o] ← null
P_EP ← null
if Ar.o == R_ooav.object then
    object_attribute_values[o] ←
    object_attribute_values[o] ∪ {oav}
end
if
object_attribute_values[o] ⊇ object_attribute_values[p].R_poa
AND Ar.r == R_poa.right
then
    P_EP ← P_EP ∪ {p}
end
return P_EP
```

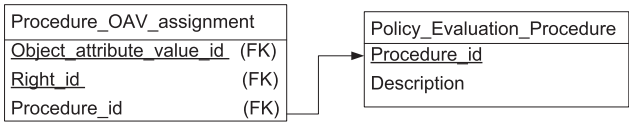

Fig. 6 – Schema for identifying meta-policy.

function returns a meta-policy, then the access request is verified through that meta-policy. Otherwise, the access is denied.

As an example, consider the other three access control requirements of the banking system described in Section 1. The first requirement states that a user *#u*, who is a *Deputy Manager* in the *Branch Banking* department, through the role *Customer Service Officer*, can *approve* a transaction *#Tx* which is not initiated by that user and is within her approval limit, during working hours from the branch of posting. The second requirement states that a user *#u*, who is a *Assistant Manager* in the *Branch Banking* department, through the role *Customer Service Officer*, can *initiate* a transaction *#Tx* during working hours from the branch of posting. The third requirement states that any user *#u* can *initiate* a transaction *#Tx*. These requirements are captured through specification of meta-policies as follows.

Meta-Policy MP5:

PRCA: Allow access if all sub-policies allow access

RBAC Policy: UR = (u, Customer Service Officer), PR = (Customer Service Officer, Perm), Perm = (Tx, {approve})

ABAC Policy: P_{uav} = (P, {Deputy Manager, Branch Banking}), P_{oav} = (P, Transaction), P_r = (P, {approve}), P_{eav} = (P, {Working hours, Branch of posting, Not initiated transaction, Within approval limit})

Meta-Policy MP6:

PRCA: Allow access if any sub-policy allows access

ABAC Policy: P_{uav} = (P, {Customer Service Officer, Assistant Manager, Business Banking}), P_{oav} = (P, Transaction), P_r = (P, {initiate}), P_{eav} = (P, {Working hours, Branch of posting})

DAC Policy: (u, Tx, {initiate})

The meta-policies MP5 and MP6 are composed of ABAC and RBAC policies that are captured in schemas shown in Figs. 2 and 3, respectively. The MP5 specifies the first requirement and allows those users' to *approve* the transaction *#Tx* who possess role and attribute values mentioned in RBAC and ABAC policy, respectively. In contrast, MP6 specifies second and third requirements and permits those users to *initiate* the transaction *#Tx* who either possess attribute values specified in the ABAC policy or satisfy the DAC policy.

6.3. *Meta-policy identification mechanism*

Depending on the sensitivity of objects, a specific type of access (say, read) on the objects can be controlled by a particular meta-policy (procedure), whereas another type of access (say, write) on the same objects can be controlled by another meta-policy (procedure). So, there is a need to capture the information about meta-policy specification which would help in identifying the appropriate meta-policy to be used for evaluating an access request. One of the simplest ways to do so is to capture the specification information in the form of a triple (meta-policy, object, right) which states that the specific meta-policy governs the particular right on the object. However, such type of representation can cause performance and maintenance overhead in case of frequent changes in the access types of objects or the creation/deletion of objects. Thus, for minimizing the overhead and the number of policy combinations, we present a meta-policy identification schema which captures a meta-policy and a right along with object attributes instead of object id or name.

As shown in Fig. 6, the meta-policy identification schema consists of two relations, namely *Policy_Evaluation_Procedure* and *Procedure_OAV_assignment*. The first relation captures meta-policies (procedures) and their description. The second relation captures the association of meta-policies (procedures) with object attribute values and rights, which represents that the particular meta-policy (procedure) controls the specific types of rights on objects.

An algorithm to identify a meta-policy is presented in Algorithm 7. The algorithm takes an access request *Ar*, the relation procedure oav assignment R_{poav} and the relation object oav assignment R_{ooav} as input and fetches the attribute values associated with the object *#o* through the relation R_{ooav}. Then, it searches a meta-policy through the relation R_{poav} in which object attribute values are either a subset or same as of the above-fetched object attribute values and access right is same as the right *#r*. If the meta-policy exists, then the request is verified through that meta-policy, otherwise, discarded.

From the above discussions, it may be observed that, Category 1 protects objects through a single combination of policies and a PRCA. Categories 2–4 use the meta-policy identification mechanism and protect a subset of the objects through one meta-policy and the remaining ones through another meta-policy. Thus, these categories present all possible ways of combining ABAC with RBAC and DAC policies through PRCAs to enable specification and enforcement of unified security policies.

7. Policy evaluation

Once the access control policies have been appropriately specified, the next related issue is their evaluation. The access control system needs to determine if the existing policies allow

Table 20 – Models and schemas referred for processing administrator requests.

AR no.	Model	Schema	Access request
1	ABAC	Fig. 2	Objects that a user #*u* can access.
2			Rights that a user #*u* can perform on an object *o*.
3			Users that can access an object #*o*.
4			Users that can exercise a right #*r* on an object #*o*.
5	RBAC	Fig. 3	Users who have a permission #*p*.
6			Permissions available to a user #*u*.
7			Roles that have a permission #*p*.
8			Permissions associated with a role #*r*.
9			Roles assigned to a user #*u*.
10	DAC	Fig. 3	The complete set of privileges various users have over different objects.

access. Such evaluation of a request is done by issuing a query to the policy vault shown in Fig. 1.

It may be noted that users or administrators are not expected to directly express their requests in the concerned query language statements (e.g., SQL). Rather, these requests would be embedded in the host language using which the application making the request is developed. There are broadly two categories of queries, namely evaluation query and enforcement query, that are, respectively, used to evaluate administrator request (AR) and user request (UR) as given below.

7.1. *Evaluation query*

Evaluation queries are executed by administrative users to either obtain more insight into the policies or verify the correctness of policies in case of any change in them. Table 20 presents the details of a few ARs as well as access control models and schemas related to them. A detailed description of the execution of ARs through queries by using the relations shown in Figs. 2 and 3 is given below.

- To evaluate AR1, an evaluation query first obtains the attribute values associated with the user #*u* through the relation *user_uav_assignment*. Next, it fetches those rules through the relation *rule_uav_assignment* in which user attribute values are either a subset or same as of the user #*u* attribute values. Then, it retrieves the object attributes through the relation *rule_oav_assignment* in which rules belong to the above-retrieved rules. Finally, it fetches the objects through the relation *object_oav_assignment* in which object attribute values are either a subset or same as of the above-retrieved object attribute values.
- To evaluate AR2, an evaluation query first finds the rules through the relations *rule_uav_assignment* and *rule_oav_assignment* in which user attribute values and object attributes values are either a subset or same as of the user #*u* attribute values and object #*o* attribute values, respectively. Finally, it fetches the rights through the relation *rule_right_assignment* in which rules belong to the above-retrieved rules.
- To evaluate AR3, an evaluation query first finds the attribute values associated with the object #*o* through the relation *object_oav_assignment*. Next, it fetches the rules through the relation *rule_oav_assignment* in which user attribute values are either a subset or same as of the object #*o* attribute values. Then, it retrieves the user attribute values through the relation *rule_uav_assignment* in which rules belong to the above-fetched rules. Finally, it fetches the users through the relation *user_uav_assignment* in which user attribute values are either a subset or same as of the above-retrieved user attribute values.
- To evaluate AR4, an evaluation query first finds the rules through the relations *rule_oav_assignment* and *rule_right_assignment* in which object attribute values are either a subset or same as of the object #*o* attribute values and access right is same as the right #*r*, respectively. Then, it fetches the user attribute values through the relation *rule_uav_assignment* in which rules belong to the above-retrieved rules. Finally, it obtains users through the relation *user_uav_assignment* in which user attribute values are either a subset or same as of the above-fetched user attribute values.
- To evaluate AR5, an evaluation query first finds the roles that have the permission #*p* directly or indirectly through the relations *permission_role_assignment* and *role_hierarchy*, respectively. Finally, it obtains the users associated with the above-retrieved roles through the relation *user_role_assignment*.
- To evaluate AR6, an evaluation query first finds the roles assigned to the user #*u* directly or indirectly, respectively, through the relations *user_role_assignment* and *role_hierarchy*. Finally, it fetches the permissions associated with the above-fetched roles through the relation *permission_role_assignment*.
- To evaluate AR7, an evaluation query fetches the roles which have the permission #*p* directly or indirectly through the relations *permission_role_assignment* and *role_hierarchy*, respectively.
- To evaluate AR8, an evaluation query first finds the roles available to the role #*r* directly or indirectly through the relation *role_hierarchy*. Finally, it retrieves the permissions associated with the above-retrieved roles through the relation *permission_role_assignment*.
- To evaluate AR9, an evaluation query fetches the roles assigned to the user #*u* directly or indirectly through the relations *user_role_assignment* and *role_hierarchy*, respectively.
- To evaluate AR10, an evaluation query directly refers to the relation *right_assignment* and enables the administrator know the complete set of DAC privileges which the users have on different objects.

7.2. *Enforcement query*

Enforcement queries are executed for checking whether a user request should be granted. The details of a few user requests as well as access control models and schemas related to them are presented in Table 21. The detailed description of user requests evaluation through queries by using the relations shown in Figs. 2 and 3 is given below.

- To verify UR11, an enforcement query first finds the roles associated with the user #*u* directly or indirectly,

Table 21 – Models and schemas referred for processing user requests.

UR no.	Model	Schema	Access request
11	RBAC	Fig. 3	A user #u can exercise a right #r
12	ABAC	Fig. 2	on an object #o.
13	ABAC	Fig. 2	A user #u can exercise a right #r on an object #o at time #t.
14	ABAC	Fig. 2	A user #u can exercise a right #r on an object #o at time #t and location #l.
15	Unified	Figs. 3	A user #u can exercise a right #r
16	Policies	and 2	on an object #o.
17	DAC	Fig. 3	A user #u can exercise a right #r on an object #o.

respectively, through the relations *user_role_assignment* and *role_hierarchy*. Next, it obtains the permissions assigned to the above-fetched roles through the relation *permission_role_assignment*. Finally, it searches the permission through the relation *permission_object_assignment* that comprises the object #o and right #r, and belongs to the above-retrieved permissions. If the permission exists, then the access is granted; otherwise, the access is denied.

- To verify UR12, an enforcement query first finds the user attribute values and object attribute values associated with the user #u and object #o through the relations *user_uav_assignment* and *object_oav_assignment*, respectively. Next, it finds the rules through the relation *rule_uav_assignment* in which user attribute values are either a subset or same as of the user #u attribute values. Then, it fetches the rules through the relation *rule_oav_assignment* in which object attribute values are either a subset or same as of the object #o attribute values, which belong to the above-retrieved rules. Finally, it retrieves the rule through the relation *rule_right_assignment* in which right is same as the right #r, which belongs to the above-retrieved rules. If the rule exists, then the access is permitted. Otherwise, the access is denied.
- To verify UR13, an enforcement query first finds the user attribute values and object attribute values associated with the user #u and object #o through the relations *user_uav_assignment* and *object_oav_assignment*, respectively. Next, it finds the rules through the relation *rule_uav_assignment* in which user attribute values are either a subset or same as of the user #u attribute values. Then, it fetches the rules through the relation *rule_oav_assignment* in which object attribute values are either a subset or same as of the object #o attribute values, which belong to the above-fetched rules. Finally, it retrieves the rule through the relation *rule_right_assignment* in which right is same as the right #r, which belongs to the above-fetched rules. If the above-fetched rule is available at the time #t that is verified through the relation *rule_eav_assignment*, then the access is permitted. Otherwise, the access is denied.
- To verify UR14, an enforcement query first finds the user attribute values and object attribute values associated with the user #u and object #o through the relations *user_uav_assignment* and *object_oav_assignment*, respectively. Next, it finds the rules through the relation *rule_uav_assignment* in which user attribute values are either a subset or same as of the user #u attribute values. Then, it fetches the rules through the relation *rule_oav_assignment* in which object attribute values are either a subset or same as of the object #o attribute values, which belong to the above-fetched rules. Finally, it retrieves the rule through the relation *rule_right_assignment* in which right is same the right #r, which belongs to the above-fetched rules. If the above-fetched rule is available at the time #t and location #l that is verified through the relation *rule_eav_assignment*, then the access is permitted. Otherwise, the access is denied.
- To verify UR15, an enforcement query, which is composed of ABAC, RBAC and DAC policies, evaluates the request similar to UR11, UR12 and UR17, respectively. If at least one policy outcome is true, then the access is granted; otherwise, the access is denied.
- To verify UR16, an enforcement query, which is composed of ABAC, RBAC and DAC policies, evaluates the request similar to UR11, UR12 and UR17, respectively. If each policy outcome is true, then the access is granted; otherwise, the access is denied.
- To verify UR17, an enforcement query directly searches for an instance which contains the user #u, object #o and right #r in the relation *right_assignment*. If the instance exists, then the access is granted. Otherwise, the access is denied.

There could be various other user and administrator requests. However, due to space constraint, we only considered frequently occurring user and administrative access requests, and described their evaluation using the schemas shown in Figs. 2 and 3. The execution time of these access requests is reported and discussed in the next section. Similar access requests can also be written to list the administrative policies, as well as to evaluate administrative access using the schema shown in Fig. 4. Furthermore, requests that check for violation of SoD, prerequisite and cardinality constraint can also be developed on the schema of Fig. 3.

8. Experimental results

In this section, we present data set descriptions and details of the implementation of the proposed approach along with the outcomes of experiments on scalability and speed-up. The schemas shown in Figs. 2 and 3 of Section 4 were specified as tables in Oracle 12c (imdb) database on a system having a 3.10 GHz Intel i5 processor, 4 GB RAM and 64 bit Windows 7 operating system.

8.1. Data set details

Table 22 summarizes the details of different feature sizes of five synthetic data sets. For Data Sets 1–5, the size of each relation is shown in terms of their number of rows increase. The relations that influence the execution time of requests are shown in Table 22, whereas the remaining relations are populated as follows. The number of rows in the *Right_assignment* relation is captured in such a way that each user can perform

Table 22 – Data set details showing the number of rows in various relations.

Parameters	Data Set 1	Data Set 2	Data Set 3	Data Set 4	Data Set 5
Number of users	100	500	500	1000	5000
Number of objects	100	500	1000	5000	25,000
Number of rights	5	10	10	10	10
Number of roles	10	50	50	100	100
Number of permissions	150	750	1500	7500	40000
Number of permission role assignments	150	750	1500	7500	40,000
Number of user role assignments	200	1000	1000	2000	10,000
Number of user attributes	5	25	25	50	250
Number of user attribute values	10	50	50	100	500
Number of user to user attribute value assignments	200	1000	2000	10,000	50,000
Number of object attributes	5	25	50	250	1250
Number of object attribute values	10	50	100	500	2500
Number of object to object attribute value assignments	200	1000	2000	10,000	50,000
Number of ABAC rules	10	25	50	250	1250

at least one right on an object and each object can be accessed by at least one user through some access right (e.g., read, write, delete, etc.). The number of such access rights is usually limited in database systems (e.g., insert, update, delete, select, etc.) or operating systems (e.g., read, write, append, execute, etc.). Therefore, the number of rights was limited to 5 for Data Set 1, and 10 for Data Sets 2–5.

To emulate real-world scenarios, it has been ensured that each role is part of at least one role hierarchy, and each role hierarchy has at most four levels. The various parameters of different sizes shown in Table 22 were selected as follows:

1. To represent from small to large organizations, the number of users was varied from 100 to 5000, and the number of roles was maintained at 10% of the number of users for Data Sets 1–4. In Data Set 5, the number of roles was kept the same as Data Set 4, since after a certain point, the number of roles in an organization does not increase as compared to the number of users. The number of permissions was created in such a way that each object is included in at least one permission via a particular access right. However, in permission to role assignment, every permission was assigned to at least one role and some permissions were assigned to more than two roles. Roles were assigned to users in a similar manner.
2. For ABAC, the number of user attributes was maintained at 5% of the number of users and the number of object attributes was also kept at 5% of the number of objects. Every user attribute and object attribute contained at least one user attribute value and one object attribute value, respectively. However, in user attribute values to users assignment, every user attribute value was assigned to at least one user and some of the user attribute values were assigned to more than two users. Object attribute values were assigned to objects in a similar manner.

Data Sets 2, 3, 4 and 5, respectively, contain all the entries of Data Sets 1, 2, 3 and 4 as well as some extra entries. Those entries represent various components of ABAC, RBAC, and DAC, such as user, role, object, permission, user-role assignment, etc. Thus, a data set represents the size of an organization in terms of users, resources, access rights and authorization policies which restrict user access to resources.

For each data set, every request (AR1-AR10 and UR11-UR17) was executed 73 times with different inputs to ensure 95% confidence level with 5% margin of error. Due to a lower limit on the data storage size for the in-memory database, none of the relations from Data Sets 1–3 was stored in-memory. Therefore, no change was observed in the execution time of the requests for Data Sets 1–3.

8.2. *Scalability study and speed-up analysis*

Scalability of the proposed system, which refers to its ability of handle a small to large variation in the number of entries of its various components, has been demonstrated by evaluating variation in execution time of user and administrative access requests against those five data sets. Table 23 shows the average disk-based execution time, in-memory execution time and speed-up for AR1-AR10 and UR11-UR17 for Data Sets 4 and 5.

AR1-AR4, which retrieve ABAC policy-related information regarding a user, object, or right, execute in less than 0.3 s and exhibit a gain of 8.6 in the speed-up for Data Set 5. AR5-AR9, which retrieve policy-related information regarding a particular user or role or permission through RBAC relations, execute in less than 0.1 s for Data Set 5. Moreover, in-memory execu-

Table 23 – Access request execution time (seconds).

Access request	Data Set 4			Data Set 5		
	Disk-based	In-memory	Speed-up	Disk-based	In-memory	Speed-up
AR 1	0.185	0.102	1.8	0.306	0.118	2.6
AR 2	0.007	0.004	1.8	0.053	0.008	6.6
AR 3	0.011	0.004	2.8	0.026	0.003	8.6
AR 4	0.008	0.004	2.0	0.025	0.005	5.0
AR 5	0.010	0.002	5.0	0.026	0.003	8.7
AR 6	0.008	0.004	2.0	0.011	0.004	2.8
AR 7	0.001	0.001	1.0	0.002	0.001	2.0
AR 8	0.004	0.003	1.3	0.005	0.003	1.7
AR 9	0.002	0.001	2.0	0.003	0.001	3.0
AR 10	0.027	0.022	1.2	0.159	0.124	1.3
UR 11	0.005	0.005	1.0	0.028	0.008	3.5
UR 12	0.005	0.004	1.3	0.009	0.003	3.0
UR 13	0.010	0.007	1.4	0.014	0.004	3.5
UR 14	0.247	0.202	1.2	0.291	0.126	2.3
UR 15	0.016	0.012	1.3	0.025	0.020	1.3
UR 16	0.025	0.017	1.5	0.033	0.022	1.5
UR 17	0.004	0.001	4.0	0.039	0.001	39.0

tion of AR5 for Data Set 5 shows a substantial gain of 8.7 in the speed-up.

For Data Set 5, AR10 runs in less than 0.2 s and 0.1 s for disk-based and in-memory based access, respectively. Moreover, AR10 does not exhibit any significant gain in speedup because retrieving the complete set of rows (privileges) from the relational database (row format) is as fast as the in-memory database (column format).

UR11 is RBAC enforcement request and executes in less than 0.1 s for Data Set 5. UR12-UR14 are ABAC enforcement requests that contain the environment attribute as null, time, time and location, respectively. Among these, UR14 is a computationally expensive request, however, executes in less than 0.3 s for Data Set 5.

UR15-UR16 are unified access control enforcement requests, which execute in less than 0.1 s for Data Set 5. UR15-UR16 were evaluated through meta-policies which are composed of ABAC, RBAC and DAC policies, and grant access according to the result of any one policy and all policies, respectively.

UR17 is a DAC enforcement request that directly verifies access through a dimensionally modeled relation *right_assignment* and shows a significant gain of 39 in the speed-up for Data Set 5.

It may also be noted from Tables 22 and 23 that even with 50-fold rise in the number of users, 250-fold rise in the number of permissions, and 250-fold or more rise in the number of objects, the execution time of user and administrator requests does not grow significantly. Thus, it may be concluded that the proposed approach is quite scalable from small to large organizations and in-memory storage of policies results in a significant gain in speed-up and execution time of access requests.

9. Related work

In the field of access control, several efforts have been made to develop unified policy specifications, such as XACML (Committee, 2015) and Ponder (Damianou et al., 2001). XACML, which is an OASIS standard, is an attribute based policy specification and enforcement language that comprises a policy specification language and a request/response language coded in XML. Access control policies are described using the policy specification language, and access requests are validated using the request/response language. Moreover, XACML provides an attribute-based profile for the specification of a particular type of access control policy. XACML has certain potential disadvantages. For example, its Policy Decision Point is stateless and does not index policies. On the other hand, Ponder is an object-oriented policy specification language that can specify policies for managing and securing various applications, such as firewalls, storage, etc. In Ponder, policies are specified using its specification language which can then be transformed into various security mechanisms for enforcement. Thus, policy specification is separated from its enforcement, which can also be considered as a drawback of Ponder.

Barker (2009) proposed a meta-model which uses a logic-based language for the specification of access control policies and demonstrated that several access control policies can be specified using the meta-model. However, the meta-model does not address the issue of granularity, complexity of analysis, flexibility, hierarchy and enforcement overhead. On the other hand, Ferraiolo et al. (2010) present a framework, referred to as the Policy Machine (PM), that can specify and enforce various types of access control policies (e.g., DAC, MAC, Chinese Wall and RBAC). They also introduce a mechanism for combining these policies. However, Ferraiolo's framework does not address the issue of multiple dimensions and granularities.

Several competing proposals, such as role-trust management framework (Li et al., 2002), flexible authorization framework (FAF) (Jajodia et al., 2001), SecPAL (Becker et al., 2010) and metamodeling with formal semantics (Abd-Ali et al., 2015), have been made for defining a general, declarative framework that can specify a variety of access control policies for distributed systems. The role-based trust management frame-

work presents an approach for combining the benefits of RBAC and trust-management to address the issue of access control in a large scale, decentralized system. FAF is a unified language based framework that enables the specification of exclusive policies, such as positive authorization and negative authorization. It can easily represent traditional access control policies and can capture real-world application requirements. SecPAL is a declarative logic based authorization language that enables specification of decentralized policies (e.g., delegation, constraints and negation). Metamodeling with formal semantics is a First Order Logic based approach for specifying a model, meta-model, and an instance of a meta-model.

Kuhn et al. (2010) proposed several strategies for combining RBAC and ABAC policies. Following this, Huang et al. (2012) introduced a two-layered framework for integrating ABAC policies into RBAC. The two layers are called the aboveground level and underground level. The first layer represents the extended RBAC, in which user-role assignments and permission-role assignments are constrained by attribute-based policies. These policies are composed of the attributes of environments and are defined at the second layer. Jin et al. (2012b) propose an approach that restricts the availability of permissions to roles through filter functions based on user and object attributes. The *TargetFilter* function maps a subset of filter functions to each object by using conditions which are composed of object attributes and determines the applicability of filter functions on objects.

Similar to Jin et al. (2012b) and Rajpoot et al. (2015a,b) also present a mechanism for restricting the availability of permissions to roles through conditions which can be composed of attributes of user, object and environment. Moreover, they define permission as a right on an object expression instead of a right on an object. An object expression comprises object attributes and is used to identify an object. Thus, the association of an object expression and conditions with each permission can cause enforcement and maintenance overhead.

Recently, Qi et al. (2016) have proposed an approach for combining the RBAC and ABAC models that divides access control into two components: static (upper) and dynamic (lower). The upper part contains RBAC policies, whereas the lower part is constrained by ABAC policies. Unlike Huang et al. (2012) and Jin et al. (2012b), they divide session establishment into two components: session_part1 and session_part2. The first part determines the roles available to a user on the basis of RBAC and ABAC policies defined by using user, role and environment attributes, whereas second part determines permissions available to a user on the basis of the roles retrieved in the former part and ABAC policies composed of role, object and environment attributes. On the other hand, Hong et al. (2018) present an approach which combines time and attributes to restrict the access on time-sensitive data.

It may be noted that, all the above-mentioned approaches either present a language or a framework for specifying various types of access control policies and enforcing the policies. None of these approaches addresses their implementation, configuration and maintenance. Moreover, the approaches do not consider multiple dimensions and different granularity levels of policies. Therefore, most of the aforementioned access control approaches have not been employed in real life.

In contrast, our approach makes use of multi-dimensional data models and effectively addresses specification, enforcement and maintenance of attribute-based, role-based, standard and unified security policies.

10. Conclusion and future work

In this work we have presented a novel approach for the concurrent specification and enforcement of access control policies using data warehousing and in-memory databases. Specifically, we have designed a unified database schema that is capable of representing various types of access control policies which is also easily extensible to various access control models. We have also presented meta-policy specification and evaluation mechanisms, and shown that the decision whether to actually provide the requested access or not depends on a policy result combination mechanism, e.g., “Allow access if any policy allows access” and “Allow access if all policies allow access”. To improve access enforcement performance, we have analyzed the relations and stored critical relations in-memory. Experimental results show that the proposed approach, when implemented in an in-memory database like Oracle 12c, is quite scalable from small to large organizations due to a significant reduction in the execution time of access requests. Beyond specification, management and enforcement, analysis of security policies is essential for organizations to understand the implications of their policies. Therefore, in future, we plan to introduce a methodology for analyzing the security properties, such as safety and liveness, of the proposed unified security policies.

Conflict of interest

None.

Acknowledgments

Research reported in this publication was supported by the National Institutes of Health under award R01GM118574 and by the National Science Foundation under awards CNS-1564034, CNS-1624503, and CNS-1747728. The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research.

REFERENCES

Abd-Ali J, Guemhioui KE, Logrippo L. Metamodelling with formal semantics with application to access control specification. In: Proceedings of the third international conference on model driven engineering and software development; 2015. p. 354–62.

Aich S, Mondal S, Sural S, Majumdar AK. ESTARBAC: role based access control with spatiotemporal context for mobile applications. Trans Comput Sci 2009;IV:177–99.

Aich S, Sural S, Majumdar AK. STARBAC: spatio temporal role based access control. In: Proceedings of the 2007 OTM confederated international conference on on the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS – Volume Part II; 2007. p. 1567–82.

Barker S. The next 700 access control models or a unifying meta-model?. In: Proceedings of the fourteenth ACM symposium on access control models and technologies; 2009. p. 187–96.
Becker M, Fournet C, Gordon A. Design and semantics of a decentralized authorization language. J Comput Secur 2010;4(8):619–65.
Bell DE, LaPadula LJ. In: Technical Report MTR-2997. Secure computer systems: unified exposition and multics interpretation. Bedford, MA: The Mitre Corporation; 1976.
Bertino E, Bonatti PA, Ferrari E. TRBAC: a temporal role-based access control model. ACM Trans Inf Syst Secur 2001;4(3):191–233.
Bertolissi C, Fernandez M. A metamodel of access control for distributed environments: applications and properties. Inf Comput 2014;238(9):187–207.
Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology. SIGMOD Rec 1997;26(1):65–74.
Committee OT. OASIS extensible access control markup language (XACML), 2015. http://docs.oasis-open.org/xacml/3.0/xacml-profile-saml2.0-v2-spec-en.html.
Committee T. A survey of access control methods, 2009. http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf.
Damiani ML, Bertino E, Catania B, Perlasca P. GEO-RBAC: a spatially aware RBAC. In: Proceedings of the tenth ACM symposium on access control models and technologies; 2005. p. 29–37.
Damianou N, Dulay N, Lupu E, Sloman M. The ponder policy specification language. In: Proceedings of the workshop on policies for distributed systems and networks; 2001. p. 18–39.
Fatima A, Ghazi Y, Shibli MA, Abassi AG. Towards attribute-centric access control: an ABAC versus RBAC argument. Secur Commun Netw 2016;9:3152–66.
Ferraiolo D, Atluri V. A meta model for access control: why is it needed and is it even possible to achieve?. In: Proceedings of the thirteenth ACM symposium on access control models and technologies; 2008. p. 153–4.
Ferraiolo D, Atluri V, Gavrila S. The policy machine: a novel architecture and framework for access control policy specification and enforcement. J Syst Archit Embed Syst Des 2010;57(4):412–24.
Graham GS, Denning PJ. Protection principles and practice. In: Proceedings of the American federation of information processing societies spring joint computer conference; 1972. p. 417–29.
Hong J, Xue K, Xue Y, Chen W, Wei DSL, Yu N, Hong P. TAFC: time and attribute factors combined access control for time-sensitive data in public cloud. IEEE Trans Serv Comput 2018:1–14.
Hu VC, Ferraiolo D, Kuhn R, Friedman AR, Lang AJ, Cogdell MM, Schnitzer A, Sandlin K, Miller R, Scarfone K. Guide to attribute based access control (ABAC) definition and considerations (Draft), 2013. https://csrc.nist.gov/csrc/media/publications/sp/800-162/final/documents/sp800_162_draft.pdf.
Huang J, Nicol DM, Bobba R, Huh JH. A framework integrating attribute-based policies into role-based access control. In: Proceedings of the seventeenth ACM symposium on access control models and technologies; 2012. p. 187–96.
Jajodia S, Samarati P, Sapino ML, Subrahmanian VS. Flexible support for multiple access control policies. ACM Trans Inf Syst Secur 2001;26(2):214–60.
Jin X, Krishnan R, Sandhu R. A role-based administration model for attributes. In: Proceedings of the first international workshop on secure and resilient architectures and systems; 2012a. p. 7–12.
Jin X, Sandhu R, Krishnan R. RABAC: role-centric attribute-based access control. In: Proceedings of the twelfth international conference on mathematical methods, models and architectures for computer network security; 2012b. p. 84–96.
Joshi JBD, Bertino E, Latif U, Ghafoor A. A generalized temporal role-based access control model. IEEE Trans Knowl Data Eng 2005;17(1):4–23.
Kuhn DR, Coyne EJ, Weil TR. Adding attributes to role-based access control. IEEE Comput 2010;43(6):79–81.
Li N, Mitchell JC, Winsborough WH. Design of a role-based trust-management framework. In: Proceedings of the IEEE symposium on security and privacy; 2002. p. 114–30.
Li N, Wang Q, Qardaji W, Bertino E, Rao P, Lobo J, Lin D. Access control policy combining: theory meets practice. In: Proceedings of the fourteenth ACM symposium on access control models and technologies; 2009. p. 135–44.
Oracle Technical Committee. Oracle database in-memory, 2015. http://www.oracle.com/technetwork/database/in-memory/overview/twp-oracle-database-in-memory-2245633.html.
Qi H, Luo X, Di X, Li J, Yang H, Jiang Z. Access control model based on role and attribute and its implementation. In: Proceedings of the international conference on cyber-enabled distributed computing and knowledge discovery; 2016. p. 66–71.
Rajpoot QM, Jensen CD, Krishnan R. Attributes enhanced role-based access control model. In: Proceedings of the twelfth international conference on trust, privacy and security in digital business; 2015a. p. 3–17.
Rajpoot QM, Jensen CD, Krishnan R. Integrating attributes into role-based access control. In: Proceedings of the twenty-ninth annual IFIP WG 11.3 working conference on data and applications security and privacy; 2015b. p. 242–9.
Rao P, Lin D, Bertino E, Li N, Lobo J. An algebra for fine-grained integration of XACML policies. In: Proceedings of the fourteenth ACM symposium on access control models and technologies; 2009. p. 63–72.
Ray I, Kumar M, Yu L. LRBAC: a location-aware role-based access control model. In: Proceedings of the second international conference on information systems security; 2006. p. 147–61.
Ray I, Toahchoodee M. A spatio-temporal role-based access control model. In: Proceedings of the twenty-first annual IFIP WG 11.3 working conference on data and applications security; 2007. p. 211–26.
Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans Inf Syst Secur 1999;2(1):105–35.
Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. IEEE Comput 1996;29(2):38–47.
Servos D, Osborn SL. Current research and open problems in attribute-based access control. ACM Comput Surv 2017;49(4) 65:1–65:45.
Singh MP, Sural S, Atluri V, Vaidya J, Yakub U. Managing multi-dimensional multi-granular security policies using data warehousing. In: Proceedings of the ninth international conference on network and system security; 2015. p. 221–35.

Mahendra Pratap Singh is a Ph.D. candidate in the Department of Computer Science & Engineering at IIT Kharagpur. His research interests are in access control, security analysis, and other aspects of information security.

Shamik Sural is a Professor at the Department of Computer Science and Engineering, IIT Kharagpur. He received the Ph.D. degree from Jadavpur University. He is a recipient of the Alexander von Humboldt Fellowship for Experienced Researchers. He has published more than 150 research papers in reputed international

journals and conferences. His research interests include computer security, data mining and database systems.

Jaideep Vaidya is the RBS Deans Research Professor in the MSIS department at Rutgers University. He received the Ph.D. degree in Computer Science from Purdue University. His general area of research is in data mining, data management, security, and privacy. He has published over 150 technical papers in peer-reviewed journals and conference proceedings, and has received several best paper awards.

Vijayalakshmi Atluri is a Professor of Computer Information Systems in the MSIS Department at Rutgers University. Her research interests include Information Security, Privacy, Databases, and Workflow Management. She has published over 150 papers in international journals and conferences. She was the recipient of the National Science Foundation CAREER Award, the Rutgers University Research Award for untenured faculty, and the 2014 IFIP WG11.3 Workgroup on Data and Application Security and Privacy outstanding research award.