Object-to-Object Relationship-Based Access Control: Model and Multi-Cloud Demonstration

Tahmina Ahmed, Farhan Patwa and Ravi Sandhu
Institute for Cyber Security and Department of Computer Science
University of Texas at San Antonio
San Antonio, Texas, USA
tahmina.csebuet@gmail.com, farhan.patwa@utsa.edu and ravi.sandhu@utsa.edu

***Abstract*—Relationship Based Access Control (ReBAC) has been recognized as a distinctive form of access control since the advent of online social networks (OSNs). In the OSN context, ReBAC typically expresses authorization policy in terms of interpersonal relationship between users. OSN-inspired ReBAC models primarily focus on user-to-user relationships, although some have also considered user-to-resource and resource-to-resource relationships. An OSN has very specific type of resources (photos, comments, notes etc.) which are closely related to users, so it is natural to consider resource relationships in OSNs as occurring through users. However user-independent resource-to-resource (or object-to-object) relationships have been around for decades in information systems. For instance, object-oriented systems maintain inheritance, composition and association relationships among objects, version control systems use derived-from relationships between different versions, and digital content management systems use fundamental-relationships between different media files. To our knowledge no existing ReBAC model considers user-independent generic relationships between objects, as a useful means to express authorization policies. This paper proposes a novel Object-to-Object ReBAC model (OOReBAC) which uses object relationships for controlling access to objects. We build a proof-of-concept implementation of OOReBAC using the open source OpenStack cloud platform and specifically its Swift object storage service.**

***Keywords*-access control; authorization; ReBAC; object relationship; Openstack; Swift**

I. Introduction

Recent growth of on-line social networks (OSNs) such as Facebook, Twitter and LinkedIn, has introduced a distinct form of authorization based on relationships between the accessing user and the content owner, commonly called relationship-based access control (ReBAC). Traditional access control models (DAC—discretionary access control, MAC—mandatory access control, RBAC—role-based access control and even ABAC—attribute-based access control) utilize user identity or some kind of user credentials (security label, role, age, sex, organizational affiliation etc.) to evaluate the access authorization of the user to resources. ReBAC introduces the concept of considering relationship path or path pattern between accessing user and target resources for authorization, bringing a new dimension to access control authorization.

Most ReBAC models build upon user-to-user relationships [1], [2], [3], [4], [5], [6], [7], while a few of them also consider user-to-resource and resource-to-resource relationships [8]. An OSN has very specific kind of resources such as photos, comments, notes etc. Tagging a user in a photo establishes a user-to-resource relation, and commenting on a photo is an example of a resource-to-resource relationship. For the special nature of OSNs, relationship between resources is meaningful primarily in context of users. Thus OSN-inspired ReBAC models typically focus on user-to-user and user-to-resource relationships as compared to resource-to-resource relationships.

Recently access control researchers have expanded the concept of ReBAC for general computing systems beyond the social environment [9], [10]. These models consider organizational structure as a relationship graph where nodes are users, resources or any kind of logical entities such as groups, projects, organizations etc. Though these models consider any kind of resources in the relationship graph they include users as part of the graph, and the authorization policy is expressed in terms of a path or path pattern including the accessing user and target resource/user as endpoints. None of the existing models consider only resource-to-resource relationship without user, while this kind of relationship is actually very important in enterprise environments. Object-oriented systems, version control systems, digital access management, digital library, recommender systems, and document clustering already maintain user-independent relationships between objects. Though there is considerable use of relationship between objects (equivalently resources) in enterprise environments, to our knowledge there is no formal ReBAC model so far which considers object-to-object relationships independent of users.

This paper propose a novel object-to-object relationship based access control model called OOReBAC, as the first model to explicitly consider user-independent object-to-objects relationship as the basis for authorization. As a proof of concept implementation we demonstrate our theoretical model in the open source cloud IaaS platform OpenStack [11] and specifically in its object storage Swift service [12].

The rest of the paper is organized as follows. Section II provides motivation of using object relationship for authorization. Section III provides detail characteristics of our proposed model, Section IV gives the formal definition of OOReBAC model. Section V provides an example application of OOReBAC policy configuration. Section VI provides the implementation detail of the defined model for OpenStack object storage Swift. Section VII presents related work which

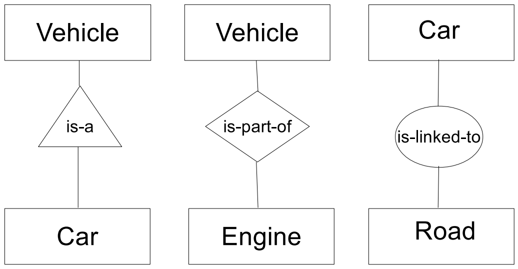

Fig. 1. *Object Relationships in Object-Oriented Systems (Inheritance, Composition and Association).*

considers using relationship in authorization. Section VIII concludes the paper.

II. Motivation

Object-to-object relationships have been considered in information systems for decades. Object-oriented systems are built upon the concept of object relationships. Inheritance maintains an “is-a” relationship where one object (superclass) allows its properties to pass to the other object (subclass) [13]. Composition maintains an “is-part-of” relationship between two objects when life cycle of two objects are dependent on one another [14]. Association maintains a “is-linked-to” relationship between two objects when the objects are independent of each other during their life cycle while somehow associated with each other [15], [16]. Figure 1 shows object relationships use in object-oriented systems. Here car is-a vehicle (inheritance), an engine is-part-of a vehicle (composition) and a car is-linked-to a road (association). Digital library uses categorical relationships between items. Digital asset management maintains fundamental relationship between different media file variations [17]. Relationships between different versions and contents are a core feature in content management systems. Version control system maintains “derived from” relationship with different versions of an object [18]. Figure 2 shows the directed acyclic graph that maintains the history of a Git (a version control system) project where each node is a commit/version/revision of the project. Co-citation [19] maintains a coupling relationship between two documents depending upon the frequency with which the documents are cited together. Document clustering uses correlation between documents [20]. Object relationships are also used in organizing and accessing large volumes of data. In May 2016 Panama paper leaks, the International Consortium of Investigative Journalists got 2.6 TB of data and 11.5 million files from the Mossac Fonseca company [21]. They have used neo4j graph database [22] to make the object-relational graph so as to organize and publish the data.

In the rest of this section we motivate the importance of building an access control model based on object relationships via some sample use cases.

Use Case 1: An enterprise content management system has contents such as images, web contents, electronic documents, videos or other media. A typical use of such a system is

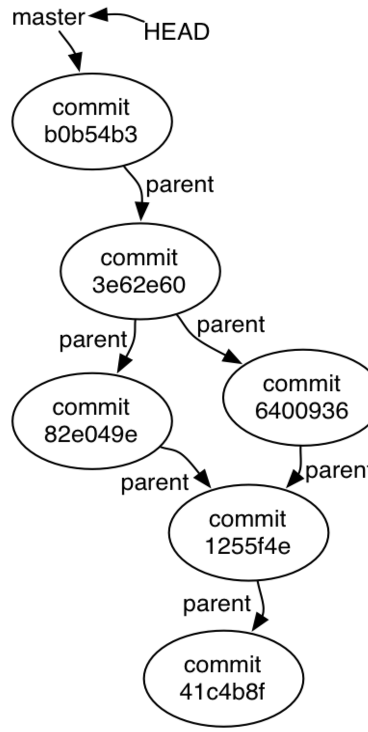

Fig. 2. *History of a Git (a Version Control System) Project is a DAG [23].*

document collaboration where a single document is accessed by several users and that document needs to have its own version control. Maintaining relationship between different versions and managing access for multiple users requires object-to-object relationship between versions, through which users can access the exact version of interest.

Use Case 2: Consider a patient’s health records in different specialities where a person went to his primary care physician with certain symptoms such as chest pain, the primary care physician created a record of his symptoms and medications he was taking at that time and referred him to a gastroenterologist, the gastroenterologist created a record of his symptoms and investigations and depend upon the results referred him to a cardiologist, the cardiologist then referred him to an endocrinologist who also referred him to an ophthalmologist and a nephrologist. In every stage of his treatment a new document is created considering the the speciality the doctor is treating him and a relationship between every document has been established. In every stage of his treatment a new document is created considering the the speciality the doctor is treating him and a relationship between every document has been established. The doctor who creates a particular document has a direct access to that document. Every time a specific doctor tries to give him a treatment he needs to look at his medical history and current treatments by other specialists using the relationship between the records. Figure 3 shows the treatment scenario of the patient. If the nephrologist needs to see the records of the gastroenterologist for that patient, he can use

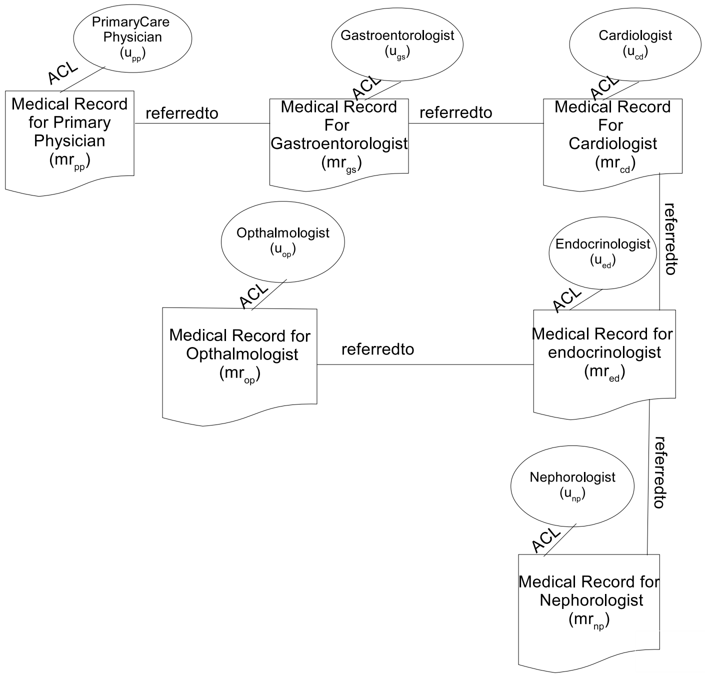

Fig. 3. *Object Relationship in Medical Record.*

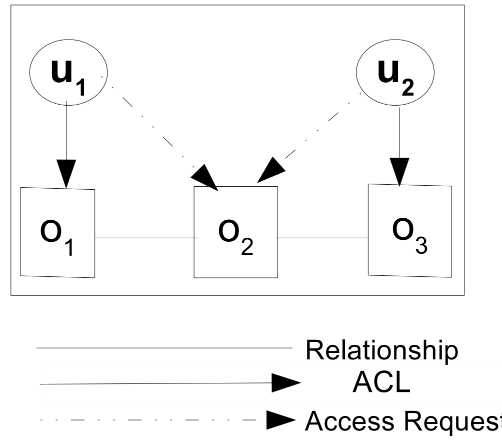

Fig. 4. *Object-to-Object Relationship Based Access Control.*

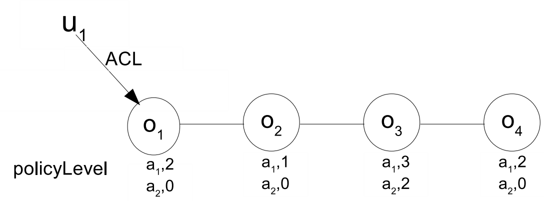

Fig. 5. *Policy Level Example.*

the relationship between records to do so.

Use Case 3: Resource relationship is also important for accessing different versions of a particular software. For example consider the scenario where different versions of a software maintain a relationship and the company who developed the software declares that user who purchased a registered version of that software can access all the earlier versions without any registration. Here to access different versions of that particular software a user needs to use the relationship between them.

III. Object-to-Object Relationship-Based Access Control Model Characteristics

In this section we discuss the general characteristics of an object-to-object relationship model for access control. To our knowledge this is a first step towards this direction. Hence we will keep our model simple, raising the question as to what are the minimum requirements to realize such a model. A typical access request in any access control model arises when a user (or subject) tries to perform an action on a resource or object. So a set of users, a set of objects and a set of actions are mandatory components for any access control model. Our main focus is on expressing authorization policy considering object relationships, so the model obviously needs a set of possible (binary) relationship types and a data structure (preferably a relationship graph) to store relationships between objects. To keep the model definition simple we will consider only one type of symmetric relationship.

We need a special direct access from a user to object which can be maintained by a system function or access control list (ACL), starting from where additional related objects can be accessed. We propose to limit, in an object specific and action specific manner, the number of relationship links (or hopcount) that can be traversed to access a related object from a given starting point. For example if the system specifies the relationship level of a particular object is 0 for write and 1 for read that means the object is not allowed to be accessed through relationship chain for write, however it allows 1 level relationship chain for read. A system function would specify the relationship level consideration for authorization of a particular object for a particular action.

Figure 4 shows how the model relationship and access would work. The system has two users u_1 and u_2, and 3 objects o_1, o_2, o_3. The relationships are $\{\{o_1,o_2\}, \{o_2, o_3\}\}$. The system function ACL would take an object as input and returns a list of users. Here ACL(o_1) = $\{u_1\}$, ACL(o_2) = $\{\}$ and ACL(o_3) = $\{u_2\}$. When user u_1 tries to access o_1 he can directly do that without using relationships. When u_1 tries to access o_2 or o_3 the access control system needs to consider relationship between $\{o_1, o_2\}$ and $\{\{o_1, o_2\}, \{o_2,o_3\}\}$ respectively.

Figure 5 shows the policy level specification of objects. Here ACL(o_1) = $\{u_1\}$, ACL(o_2) = $\{\}$, ACL(o_3) = $\{\}$, ACL(o_4) = $\{\}$. There are two actions in the system, a_1 and a_2. We have the following values of policy level as listed in Figure 5.

$$\text{policyLevel}(a_1,o_1) = 2, \text{policyLevel}(a_2,o_1) = 0$$
$$\text{policyLevel}(a_1,o_2) = 1, \text{policyLevel}(a_2,o_2) = 0$$
$$\text{policyLevel}(a_1,o_3) = 3, \text{policyLevel}(a_2,o_3) = 2$$

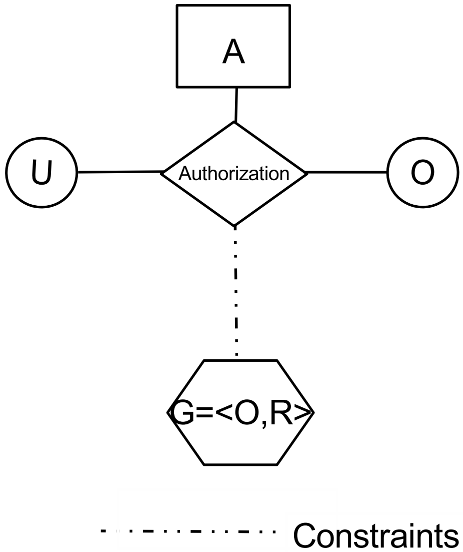

Fig. 6. *OOReBAC Model.*

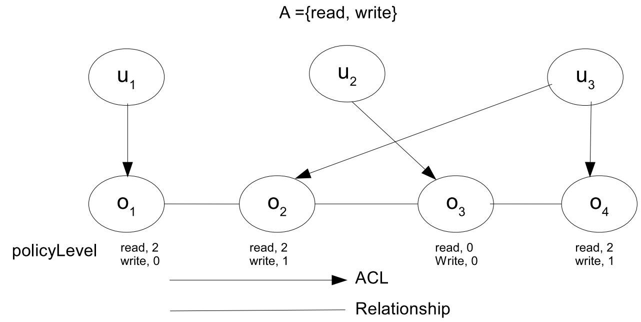

Fig. 7. *An Example of OOReBAC State I_1.*

$$\text{policyLevel}(a_1,o_4) = 2,\ \text{policyLevel}(a_2,o_4) = 0$$

When u_1 tries to do an action a_1 or a_2 on o_1 the access request would be granted as u_1 is in ACL of o_1. When u_1 tries to do action a_1 on o_2 the access would be granted because though u_1 is not in o_2's ACL, however o_2 allows upto 1 level of relationship chaining for action a_1 authorization and it maintains a 1 level relationship with o_1 and u_1 is in o_1's ACL. When u_1 tries to do a_2 on o_2 the authorization would denied as u_1 is not in o_2's ACL and o_2 allows 0 level relationship chaining for action a_2. When u_1 tries to do a_1 or a_2 on o_3 both of the actions would be granted. On the other hand when u_1 tries to do a_1 or a_2 on o_4 both the actions will be denied.

IV. OOREBAC: MODEL DEFINITION

In this section we define a model OOReBAC which considers object to object relationships in authorization policy. The model components are as follows: **U** is a set of **users**. A user is a human being who performs action on objects. **O** is a set of **objects**. Objects are resources in the system which need to be protected. **R** is a set of symmetric **relationships** between objects. **G** = $\langle$ **O, R** $\rangle$ is the relationship graph where objects are nodes and relationship between objects are edges. There is a system function **ACL** which takes an object as input and returns a set of users as output. There is another system function **policyLevel** which takes an object and an action as input and returns a natural number indicating the relationship level that object would allow for authorization of that particular action. **A** is a set of actions. Each action a $\in$ **A** has a single authorization policy **Authz**$_a$**(u:**U, **o:**O**)** which takes u and o as inputs and returns true or false. Here u and o are formal parameters. The authorization policy is a boolean function which considers object relationships, ACL and policyLevel. If Authz$_a$ (u,o) returns true then u is authorized to do action a on object o. On the other hand if Authz$_a$(u,o) returns false then u is not authorized to do action a on o.

Figure 6 shows the model components. Table I shows the formal representation of the model definition and the language for authorization policy. OOReBAC is an operational model. Create/delete users or objects, add/update relationships between objects, configure/update ACL or policy levels are administrative operations and out of scope of OOReBAC model. These would be specified in an administrative model.

An instantiation of authorization policy for OOReBAC is given below.

- A = {read, write}
- Authz$_{read}$(u:U,o:O) $\equiv$ u $\in$ P$^{policyLevel(read,o)}$
- Authz$_{write}$(u:U,o:O) $\equiv$ u $\in$ P$^{policyLevel(write,o)}$

An example configuration of OOReBAC and an instantiation of OOReBAC policy is given below.

- U = $\{u_1, u_2, u_3\}$
- O = $\{ o_1, o_2, o_3, o_4\}$
- R = $\{\{o_1, o_2\}, \{o_2, o_3\}, \{o_3, o_4\}\}$
- ACL(o_1) = $\{u_1\}$
  ACL(o_2) = $\{u_3\}$
  ACL(o_3) = $\{u_2\}$
  ACL(o_4) = $\{u_3\}$
- policyLevel(read, o_1) = 2
  policyLevel(write, o_1) = 0
  policyLevel(read, o_2)= 2
  policyLevel(write,o_2) = 1
  policyLevel(read,o_3) = 0
  policyLevel(write,o_3) = 0
  policyLevel(read,o_4) = 2
  policyLevel(write,o_4) = 1

Figure 7 shows an example state I_1 of this system. The following are some actions that different users try in state I_1 and their outcome.

- read(u_1,o_3), write(u_1,o_3) are denied
- read(u_2, o_1) is allowed, write(u_2, o_1) is denied
- read(u_1,o_4, write(u_1,o_4) are denied

V. OOREBAC:APPLICATIONS

Application of OOReBAC model is restricted to the systems where single type symmetric relationship is used. For example document co-citation, document clustering, medical record system etc. Consider our previous use case 3 defined in Section II shown in Figure 3. Let's the policy specifies that every specialist would be able to write only on a document

TABLE I
OOReBAC Model

- U is a set of users
- O is a set of objects
- R $\subseteq$ {z | z $\subset$ O $\wedge$ | z | = 2}
- G=$\langle O, R\rangle$ is an undirected relationship graph with vertices O and edges R
- A is a set of actions
- $P^i(o_1)$ = { o_2 | there exists a simple path of length p in graph G from o_1 to o_2}
- policyLevel: O $\times$ A $\rightarrow$ $\mathbb{N}$
- ACL: O $\rightarrow$ 2^U which returns the Access control List of a particular object.
- There is a single policy configuration point. Authorization Policy. for each action a $\in$ A, $Authz_a$(u:U,o:O) is a boolean function which returns true or false and u and o are formal parameters.
- Authorization Policy Language:
  Each action "a" has a single authorization policy $Authz_a$(u:U,o:O) specified using the following language.
  ϕ := u $\in$ $PATH_i$
  $PATH_i$:= $ACL(P^0(o)) \cup \ldots \cup ACL(P^i(o))$ where i = min(| O | - 1, policyLevel(a,o))
  where for any set X, ACL(X) = $\bigcup_{x \in X}$ ACL(x)

for which he/she is assigned in the ACL of that document. Reading any document is allowed through the relationship for a particular patient. To specify this policy in respect of our OOReBAC model we need to first capture the OOReBAC instantiation of the scenario as follows:

- U = { u_{pp}, u_{gs}, u_{cd}, u_{op}, u_{ed}, u_{np} }
- O = { mr_{pp}, mr_{gs}, mr_{cd}, mr_{op}, mr_{ed}, mr_{np} }
- R = {{mr_{pp}, mr_{gs}}, {mr_{gs}, mr_{cd}}, {mr_{cd}, mr_{ed}}, {mr_{op}, mr_{ed}},{mr_{np}, mr_{ed}}}}
- ACL(mr_{pp}) = {u_{pp}},
  ACL(mr_{gs}) = {u_{gs}},
  ACL(mr_{cd}) = {u_{cd}},
  ACL(mr_{op}) = {u_{op}},
  ACL(mr_{ed}) = {u_{ed}},
  ACL(mr_{np}) = {u_{np}}
- Action ={read, write}
- policyLevel(read,mr_{pp})=∞, policyLevel(write,mr_{pp})=0,
  policyLevel(read,mr_{gs})=∞, policyLevel(write,mr_{gs})=0,
  policyLevel(read,mr_{cd})=∞, policyLevel(write,mr_{cd})=0,
  policyLevel(read,mr_{op})=∞, policyLevel(write,mr_{op})=0,
  policyLevel(read,mr_{ed})=∞, policyLevel(write,mr_{ed})=0,
  policyLevel(read,mr_{np})=∞, policyLevel(write,mr_{np})=0
- Authorization policy:
  $Authz_{read}$(u,o) $\equiv$ u $\in$ $P^{policyLevel(read,o)}$
  $Authz_{write}$(u,o) $\equiv$ u $\in$ $P^{policyLevel(write,o)}$

Some sample operations and their outcomes are given below.

1) read(u_{np}, mr_{pp}) : authorized
2) read(u_{cd}, mr_{np}) : authorized
3) write(u_{np}, mr_{np}) : authorized
4) write(u_{np}, mr_{pp}) : denied
5) write(u_{np}, mr_{pp}) : denied

VI. Implementation

Using object-to-object relationship brings in a new dimension when we consider relationship between objects from cross-origin environment. Organizations often use multicloud environment for independent and parallel work, including reducing reliance on any single vendor, increasing flexibility through choice, and mitigating against disasters, etc. This is similar to the use of best-of-breed applications from multiple developers on a personal computer, rather than the defaults offered by the operating system vendor. Using multiple infrastructure providers for different workloads, deploying a single workload load balanced across multiple providers (active-active), or deploying a single workload on one provider, with a backup on another (active-passive) [24], are common multicloud applications. Sharing resources between multiple clouds IaaS is very important in today's multicloud world. Using object-to-object relationship can be one way to share our objects between different clouds.

For this implementation we use homogeneous multicloud and the platform is open source cloud IaaS OpenStack [11] for both clouds. In OpenStack we have used OpenStack object storage Swift. In this section we provide a brief description of the implementation of OOReBAC. We first review OpenStack object storage Swift and its original authorization module.

A. Swift Storage Structure

Swift is a highly available, distributed, eventually consistent object/blob store. Organizations can use Swift to store lots of data efficiently, safely, and cheaply [12]. Swift users use RESTful API [25] to upload or download objects to and from Swift object storage. Inside Swift, a project is assigned as an account. The account holds containers. Containers are similar to directories, however containers cannot be nested. A user associated with a Swift account can have multiple containers. To manage accounts, containers and objects Swift uses account servers, container servers and object servers accordingly.

1) Swift Authorization for Object Access and its limitations: In OpenStack object storage Swift authorization (request to an object access) is currently done by Access Control List (ACL). Swift has two levels of ACL: Account Level ACL and container level ACL [26]. Container Level ACL is associated with containers in terms of read (download any object of that container) or write (upload an object in the container) or list [27]. Account ACLs allow users to grant account level access to other users. The limitations of Swift authorizations are:

- It cannot express object level ACL. To specify object level ACL every object needs to be stored in a separate container.
- It cannot give user access to a particular object if the user is not a member of the account/project.
- It doesn't support multicloud resource sharing.

Our proposed model for Swift authorization can be named as relationship based resource sharing for OpenStack object storage Swift. It enables the following features.

- Object specific ACL.

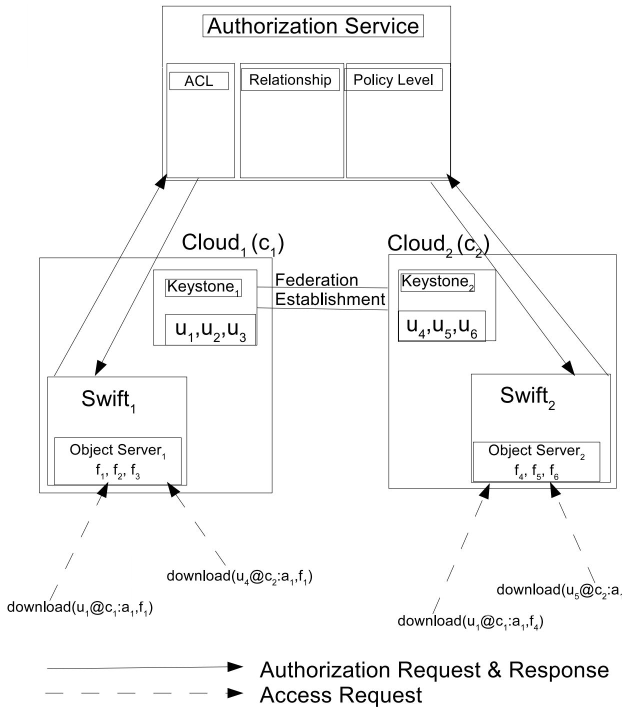

Fig. 8. *MultiCloud Implementation of OOReBAC Model.*

Algorithm 1 authorize(u,f,G)

if u in ACL(f) **then**
  return true
else
  policyLevel = policyLevel(f)
  for depth limited search upto min(policyLevel, |O| - 1) **do**
    if if any of the file's ACL contains u **then**
      return true
    end if
  end for
  return false
end if

- Allow users to access objects through relationship along with ACL.
- Allow users outside projects/accounts to access an object through relationship.
- Overall this proposed model would be able to work in multicloud environment.

To enable these features we are proposing an authorization service for Swift access control.

B. Proposed Authorization Service for Swift

An authorization service for Swift would take care of the authorization of objects. We would store all the container level ACL and relationship between files in authorization service. The collaboration between different clouds are done through federation. Once federation is established every file can be accessed by two types of user, local user and federated user. Swift operations are of two types: **Administrative Operations** and **User Operations**. Creating ACL entry for a particular object, updating ACL, creating relationship between objects, updating relationship, configuring policy levels and updating policy levels are **Administrative Operations**.

The proposed OOReBAC theoretical model is defined for operational authorization and does not include an administrative model. Therefore, for our implementation we have defined a simple administrative model for Swift authorization service. This administrative model allows an admin user from any of the collaborating clouds to configure and update relationships, ACLs and policy levels. To configure and update relationship admin user and at least one file for which relationship is being configured should be from same cloud. To configure and update ACL and policyLevel admin user and the corresponding file should be from same cloud. Admin user can directly issue a RESTAPI command from Swift to the authorization service database to create relationships, update relationships, create an ACL, update an ACL, create policy level and update policy level. In Swift **User Operations** are uploading a file and downloading a file. Only the creator of the container can upload a file. In our implementation the upload operation is kept as it is. The authorization of downloading a file is done through authorization service.

Figure 8 shows the implementation detail of the model. In this figure we are considering two clouds c_1 and c_2. First we need to establish federation between these two clouds. The authorization service would contain all the ACL information of every files, relationship information and policy level information. To configure our OOReBAC model for this implementation platform users should contain cloud and current account information along with their name as user identification. Files or objects also need to contain filename along with cloud name, account name and container name. Each user is identified as username@cloudname:accountname, each file is identified as filename@cloudname:accountname:containername.

When a download request comes from a user for a local file, the user's request triggers a RESTAPI call to the authorization service. The authorization service looks up the ACL table to determine if this user has direct access to the file. If so it returns true, else it goes to the policyLevel table to find out how many levels of relationship the file allows. Then it looks up to the policy level depth in relationship table whether any of the file up to that depth has an ACL authorizing the accessing user. If it finds any it returns true, otherwise it returns false.

Algorithm 1 shows the pseudocode of the algorithm in the authorization service to evaluate access authorization. Here we have used depth limited search upto a fix depth considering the policy level of a particular object for a particular action. Depth limited search searches upto a fix limited depth for all possible paths. Depth first search is a special case of depth limited search where limit is ∞. The overall time complexity of the algorithm is O(| O | $^{|O|}$), although with small policy limits the performance will be considerably better.

Table II specifies the administrative commands and oper-

TABLE II
Functional specification.

Functions	**Conditions**	**Updates**
	Administrative Actions	
CreateRelationship (u,$filename_1$,$filename_2$)	admin $\in$ role(u) $\wedge$ cloud($filename_1$) = cloud(u) $\wedge$ $filename_1 \notin$ RelationshipSet($filename_2$) $\wedge$ $filename_2 \notin$ RelationshipSet($filename_1$)	RelationshipSet($filename_1$) $\cup= \{filename_2\}$ RelationshipSet($filename_2$) $\cup= \{filename_1\}$
DeleteRelationship (u,$filename_1$,$filename_2$)	admin $\in$ role(u) $\wedge$ cloud($filename_1$) = cloud(u) $filename_1 \in$ RelationshipSet($filename_2$) $\wedge$ $filename_2 \in$ RelationshipSet($filename_1$)	RelationshipSet($filename_1$) $\backslash= \{filename_2\}$ RelationshipSet($filename_2$) $\backslash= \{filename_1\}$
IncludeAUserinACL (u,$filename_1$,$username_1$)	Role(u) $\in$ Admin$\wedge$ cloud($filename_1$) = cloud(u)$\wedge$ $username_1 \notin$ ACLSet($filename_1$)	ACLSet($filename_1$) $\cup= \{username_1\}$
ExcludeAUserFromACL (u,$filename_1$,$username_1$)	Role(u) $\in$ Admin$\wedge$ cloud($filename_1$) = cloud(u)$\wedge$ $username_1 \in$ ACLSet($filename_2$)	ACLSet($filename_1$) $\backslash= \{username_1\}$
ConfigurePolicyLevel (u,filename,num)	Role(u)$\in$ Admin $\wedge$ cloud($filename_1$) = cloud(u) num $\leq$ | O |	PolicyLevel(filename)= num
	Operational Command	
download (u,$filename_1$)	u$\in$ U $\wedge$ authorize(u,$filename_1$,G)	allow user u to download file $filename_1$

TABLE III
Relationship.

SourceFileName	TargetFileList
f_1@$cloud_1$:$account_1$:$container_1$	{f_2@$cloud_1$:$account_1$:$container_1$, f_3@$cloud_2$:$account_1$:$container_1$}
...	...

TABLE IV
ACL.

Filename	UserList
f_1@$cloud_1$:$account_1$:$container_1$	{u_1@$cloud_1$:$account_1$, u_2@$cloud_1$:$account_1$}
...	...

ational commands of the implemented model for the authorization service. Administrative function **CreateRelationship** creates relationship between two files by cloud admin. It takes a user and two filename as input. It checks whether the cloud admin and the first file are from same cloud and that no relationship exists between the two files. Administrative function **DeleteRelationship** deletes an existing relationship between two files, **IncludeAUserinACL** includes a user in the ACL list of a file by the cloud admin. **ExludeAUserinACL** excludes a user in the ACL list of a file by the cloud admin, and **ConfigurePolicyLevel** configures the policy level of a file by the cloud admin.The only user operation is **download**. It takes a user and a file as input, and checks whether the user is an existing user and using the **authorize** algorithm from authorization service it returns true or false.

Tables III, IV, and V shows the structure of the Relationship, ACL and policyLevel table in the authorization service. In Relationship table the graph is stored as adjacency list format. In ACL table ACL information are stored as file specific userlist and in Policy Level table file specific policylevel is stored.

TABLE V
Policy Level

FileName	Policy Level
f_3@$cloud_1$:$account_2$:$container_3$	(download,2)
...	...

VII. Related Work

Access Control based on user relationships emerged initially for online social networking (OSN). This is commonly referred to as relationship-based access control (ReBAC) [28]. A number of ReBAC models have been proposed in literature specially for OSN, most containing user-to-user relationship only [1], [2], [3], [4], [5], [6], [7]. Some models also include user-to-resource and resource-to-resource relationships [8], [9]. Recently, there has been consideration of applicability of ReBAC beyond the OSN context [9], [10]. Some models consider attributes of users and relationships [6]. A number of administrative models also have been proposed for ReBAC [10], [29].

Most of the above mentioned models are for online social network and the main feature of online social network is interpersonal relationship. So the core concerns of these models are based on user-to-user relationship. Though some of them addressed user-to-resource or resource-to-resource relationship, these are also considered in context of users. The main reason behind this consideration is OSN has very specific type of resources such as photos, comments, notes etc. which are closely related to users rather than resource-to-resource independently. Though some models [9] expanded the concept of ReBAC for general computing system they still need users in the relationship graph.

On the other hand, object relation without involvement of user is already a well accepted concept. There are some previous work that use special kind of object relationships for authorization. Object-oriented systems maintains specific form of relationships among objects (inheritance, composition, association etc.). Some access control model defined for object-oriented system use this type of specific relationship to access object [14].

VIII. Conclusion

This paper presents an object to object relationship based access control model (OOReBAC), giving a different perspective on ReBAC from the traditional one. In today's interconnected world object relationship becomes a very important feature for enterprise systems. Using this relationship to specify authorization policy would allow an access control model to specify finer-grained access control. We also have demonstrated a proof-of-concept implementation of the proposed model for open source cloud IaaS OpenStack platform. We have used OpenStack object storage Swift to specify and use object-to-object relationship in multicloud environment. The application of this simple model is restricted to systems where single symmetric relationship between objects is used. Though we are motivated by object-to-object relationship in object-oriented systems and version control systems, our proposed model is more influenced by ReBAC in social context. It only considers one type of symmetric relationship whereas object-oriented systems contain different types of asymmetric relationship (inheritance, composition, and association). Version control system considers one type of relationship "derived from" however the graph is a directed acyclic graph (DAG) and the relationship is asymmetric. As it is our first attempt towards this direction, we have kept the model definition simple to fundamentally understand the actual impact of considering object relationships in authorization policy. The proposed OOReBAC model is unable to configure object-oriented systems or version control systems. It would be interesting future work to develop a model evolved from OOReBAC, which can instantiate access control for an already existing object relationship application such as object-oriented systems and version control systems.

Acknowledgment

This research is partially supported by NSF Grants CNS-1111925 and CNS-1423481.

References

[1] G. Bruns, P. W. Fong, I. Siahaan, and M. Huth, "Relationship-based access control: its expression and enforcement through hybrid logic," in *CODASPY*, 2012, pp. 117–124.

[2] B. Carminati, E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, p. 6, 2009.

[3] P. W. Fong, M. Anwar, and Z. Zhao, "A privacy preservation model for facebook-style social network systems," in *Computer Security–ESORICS 2009*. Springer, 2009, pp. 303–320.

[4] P. W. Fong, "Relationship-based access control: protection model and policy language," in *Proceedings of the first ACM conference on Data and application security and privacy*. ACM, 2011, pp. 191–202.

[5] Y. Cheng, J. Park, and R. Sandhu, "A user-to-user relationship-based access control model for online social networks," in *Data and applications security and privacy XXVI*. Springer, 2012, pp. 8–24.

[6] ——, "Attribute-aware relationship-based access control for online social networks," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2014, pp. 292–306.

[7] P. W. Fong and I. Siahaan, "Relationship-based access control policies and their policy languages," in *Proceedings of the 16th ACM symposium on Access control models and technologies*. ACM, 2011, pp. 51–60.

[8] Y. Cheng, J. Park, and R. Sandhu, "Relationship-based access control for online social networks: Beyond user-to-user relationships," in *PASSAT, 2SocialCom*. IEEE, 2012, pp. 646–655.

[9] J. Crampton and J. Sellwood, "Path conditions and principal matching: a new approach to access control," in *Proceedings of the 19th ACM symposium on Access control models and technologies*. ACM, 2014, pp. 187–198.

[10] S. Z. R. Rizvi, P. W. Fong, J. Crampton, and J. Sellwood, "Relationship-based access control for an open-source medical records system," in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*. ACM, 2015, pp. 113–124.

[11] "http://www.openstack.org/software/icehouse."

[12] "Swift docs," "http://docs.openstack.org/developer/swift/", [Online; accessed 8-June-2016].

[13] M. P. Atkinson, F. Bancilhon, D. J. DeWitt, K. R. Dittrich, D. Maier, and S. B. Zdonik, "The object-oriented database system manifesto." in *DOOD*, vol. 89, 1989, pp. 40–57.

[14] W. Kim, E. Bertino, and J. F. Garza, "Composite objects revisited," in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '89. New York, NY, USA: ACM, 1989, pp. 337–347. [Online]. Available: http://doi.acm.org/10.1145/67544.66958

[15] J. Brunet, "Modeling the World with Semantic Objects," in *IFIP - WG 8.1 Conf. on "The Object Oriented Approach in Information Systems"*, Québec, Canada, Oct. 1991, p. 1. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00708949

[16] E. Andonoff, G. Hubert, A. Le Parc, and G. Zurfluh, "Modelling inheritance, composition and relationship links between objects, object versions and class versions," in *Advanced Information Systems Engineering*. Springer, 1995, pp. 96–111.

[17] "Digital asset management," "http://www.adobepress.com/articles/article.asp?p=2129363", [Online; accessed 14-June-2016].

[18] "Version control," "https://en.wikipedia.org/wiki/Version_control", [Online; accessed 14-June-2016].

[19] H. Small, "Co-citation in the scientific literature: A new measure of the relationship between two documents," *Journal of the American Society for information Science*, vol. 24, no. 4, pp. 265–269, 1973.

[20] Z. Su, Q. Yang, H. Zhang, X. Xu, and Y. Hu, "Correlation-based document clustering using web logs," in *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE, 2001, pp. 7–pp.

[21] "Panama paper leaks," "http://info.neo4j.com/05262016---ICIJ-and-Panama-Papers-OnDemand_LP-Video.html?aliId=38013278", [Online; accessed 8-June-2016].

[22] "http://www.neo4j.org/."

[23] "Version control," "http://web.mit.edu/6.005/www/sp16/classes/05-version-control/", [Online; accessed 20-June-2016].

[24] "Multi-cloud," "https://en.wikipedia.org/wiki/Multicloud", [Online; accessed 8-June-2016].

[25] "Swift api," "http://docs.openstack.org/developer/swift/api/object_api_v1_overview.html", [Online; accessed 8-June-2016].

[26] "Swift authorization," "http://docs.openstack.org/developer/swift/overview_auth.html", [Online; accessed 8-June-2016].

[27] P. Biswas, F. Patwa, and R. Sandhu, "Content level access control for openstack swift storage," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '15. New York, NY, USA: ACM, 2015, pp. 123–126. [Online]. Available: http://doi.acm.org/10.1145/2699026.2699124

[28] C. Gates, "Access control requirements for web 2.0 security and privacy," *IEEE Web*, vol. 2, no. 0, 2007.

[29] S. D. Stoller, "An administrative model for relationship-based access control," in *Data and Applications Security and Privacy XXIX*. Springer, 2015, pp. 53–68.