PROVENANCE-BASED ACCESS CONTROL MODELS

APPROVED BY SUPERVISING COMMITTEE:

_____
Ravi Sandhu, Ph.D., Co-Chair

_____
Jaehong Park, Ph.D., Co-Chair

_____
Weining Zhang, Ph.D.

_____
Gregory White, Ph.D.

_____
Kay Robbins, Ph.D.

Accepted: _____
Dean, Graduate School

DEDICATION

To my mother

whose boundless devotion

inspires me in life

and

to my family members

who supported and encouraged me

throughout every moment

of this endeavor.

PROVENANCE-BASED ACCESS CONTROL MODELS

by

DANG NGUYEN, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
August 2014

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

ACKNOWLEDGEMENTS

No words can express my most sincere gratitude and appreciation for my advisors, Dr. Ravi Sandhu and Dr. Jaehong Park, without whom this dissertation would not be accomplished. Their precious guidance and profound advice helped me advance while their immense patience and constant encouragement kept me motivated during my doctoral studies. Learning from their wisdom, dedication and exemplary leadership has enlightened me significantly beyond the scope of this dissertation.

I would like to thank the other members of my committee, Dr. Gregory White, Dr. Kay Robbins, and Dr. Weining Zhang, for their valuable time and insightful comments. I would also like to thank Dr. Rajendra Boppana and Dr. Shouhuai Xu for their constructive feedback in the proposal of this dissertation.

Finally, I would like to thank my best friends and colleagues, Yuan Cheng, Khalid Bijon, Bo Tang and Xin Jin, for their companionship in this academic journey. I am also very grateful for the help and support of many other friends and colleagues at the University of Texas at San Antonio.

This work has been graciously supported by the National Science Foundation grant by CNS-1111925, AFOSR MURI and the State of Texas Emerging Technology Fund.

This Masters Thesis/Recital Document or Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Masters Thesis/Recital Document or Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Masters Thesis/Recital Document or Doctoral Dissertation must still conform to all other requirements explained in the Guide for the Preparation of a Masters Thesis/Recital Document or Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this Masters Thesis/Recital Document or Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Masters Thesis/Recital Document or Doctoral Dissertation attest to the accuracy of this statement.

August 2014

PROVENANCE-BASED ACCESS CONTROL MODELS

Dang Nguyen, Ph.D.
The University of Texas at San Antonio, 2014

Supervising Professors: Ravi Sandhu, Ph.D. and Jaehong Park, Ph.D.

Provenance data of a system resource provides historical information including the pedigree of and past activities on the resource. This information is useful and has been demonstrated to be effectively usable in various computing systems in different scientific as well as business application domains. Incorporating provenance-awareness into systems has garnered considerable recent attention and been the focus of academic and industrial communities. One of the concerns is the question of how cyber security can be achieved and enhanced in systems that are provenance aware. Security tasks include how to utilize information and knowledge of provenance data to enhance existing issues of insider threat detection, malicious data dissemination, et cetera. In scenarios where provenance data is more critical than the associated system data, it is also essential to secure the provenance data.

This dissertation primarily investigates the security of provenance-aware systems from access control point of view. In provenance-aware systems, the information can be utilized for secure access control of the regular system resources as well as the associated provenance data of such resources. The two approaches can be termed provenance-based access control (PBAC) and provenance access control (PAC). A provenance data model, which is built on causality dependencies of provenance entities capturing system events, provides a foundation for achieving desirable access control goals. Built on the data model, the focus of this dissertation is on provenance-based access control models that enable efficient and expressive access control features.

PBAC models can be applied in single, distributed, and multi-tenant cloud systems. This dissertation demonstrates the application of PBAC in a single system through extending the standard XACML framework and evaluate a proof-of-concept implementation in the context of an online

homework grading system. The dissertation also demonstrates the possibility of incorporating PBAC mechanisms into cloud computing systems through developing and evaluating a proof-of-concept PBAC extension to several service components of the open-source OpenStack cloud management software. The study on a variety of deployment architecture approaches further consolidates the insights and knowledge of the process. Experimental results from these case studies demonstrate the feasibility of the approach and promise enhanced and secure access control foundation for future computing systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

Provenance is a universal term in art which refers to the origin and history of an art artifact. For example, the provenance of the ever famous painting "Mona Lisa", in the simplest form, tells the original painter is Leonardo Da Vinci, the Louvre museum is where the painting is located, and has inspired various creative and fascinating parodied artworks. More detailed provenance can provide information on past ownerships as well as any restoration works ever performed on the painting. The linkage of such past information to the painting in the present allows art enthusiasts to further appreciate and experts to correctly judge the authenticity and value of the art object.

When computer scientists and researchers adopt the concept into the digital world, they generally define the notion as the origin and all processes that led to any specific state of a data object within an information system. An example can be the documentation of software such as the Windows operating system and its evolution following the release of different versions over the years. Essentially, the provenance of a data object reflects who the original creator is, who have been involved in accessing the object, how such processes have been performed and what additional information has been captured. A provenance-aware system is capable of capturing, storing, and providing such information, as provenance data, for any applicable utility and purpose. Provenance data essentially forms a direct-acyclic graph and provides a linkage structure of history information of any data object of interest. This characteristic enables and facilitates a traversal capability on provenance data from which useful information can be extracted and utilized for different tasks and purposes.

1.1 Motivations

In recent years, the academic and industrial communities have witnessed the rapid growth of digital systems influence on human beings on the levels of personal daily lives to sophisticated financial transactions that are vital to global commerce. As a result of this phenomenon of information explosion, the growth of data that is being generated and exchanged by all involved parties also

explodes exponentially. With the spurt in data quantity, there arises a need for more insights and knowledge on these data-sets. Provenance data becomes the adopted solution for achieving this thorough knowledge goal.

In fact, many works have been performed based on motivations for provenance presence in scientific and business domains [12, 32, 39, 81, 82]. In the scientific research community where collaborative efforts are increasingly gaining momentum through the use of dynamic Virtual Organization [36] for sharing data, issues involving trust, quality, and copyright of data become very important. In many application domains such as life sciences research or material engineering, it becomes critical to determine that the available data-sets, being used or generated, possess high veracity and quality. In these settings, the associated provenance data can provide trustworthy and detailed sources of input data and all relevant transformations that have been performed. In the business domain, data communication processes are vital to organizational successes and bad data identification is critical to the organizations' operations. Collected provenance data can assist in this identification process.

Provenance data originally found useful purposes in database systems but has over the years become an integrated and significant part of many research areas including semantic webs, scientific work-flows, et cetera [18, 23, 43]. In each of these areas, provenance data generates new use cases and helps improve methodologies in achieving the respective computation goals. As a result of the different nature of each domain, the corresponding perspectives on provenance data also vary. Specifically, as different aspects of provenance are important to each specific domain, different data models for provenance are created to address these unique traits. As a consequence, it is a challenge for applications to exchange or utilize the provenance data of other applications. This led to a movement by the research community to collaborate on realizing a general purpose provenance data model that can be useful in various domains and enable inter-domain connections of provenance data [57]. Such efforts are further reinforced with the advent of cloud computing as the dominant trend of computation. In various works, researchers recognize the integral role of provenance data within this new computing paradigm [28, 59, 60]. Essentially, in a cloud comput-

ing environment that is dynamic and heterogeneous, service components, which separately manage but share physical and virtual resources, must inter-operate. Collecting and storing the provenance data of these resources can help cloud administrators and cloud users in management as well as utilization of these resources. At the same time, it is also crucial to achieve provenance understanding and communication between application domains that utilize the cloud infrastructure and resources.

As provenance finds its importance within many different areas of computer science, researchers from these areas also start recognizing the necessity for the protection and security of provenance data. A majority of recent research places a strong emphasis on methods and mechanisms to protect critical provenance information. On the other hand, other researchers focus on utilizing provenance information for security purposes including intrusion detection, insider threats, etc. Finding the solutions for addressing the security issues in a provenance-aware environment, either through using provenance data to enhance the security of the system or taking the priority in protecting the provenance data itself, remains a critical and exciting challenge for the both the academic as well as industrial communities.

1.2 Research Challenges

The myriad of software applications available in both private and public sectors demand more complex protection mechanisms for the resulting large networks of information flow. Capturing and storing provenance data in an information system enable higher trustworthiness and elevates the utility of the underlying data. Provenance-aware systems bring about additional utilities and enhancements that are uniquely enabled with provenance information. Specifically, as data provenance provides utilities such as pedigree information search, usage tracking and versioning, using provenance data for access control allows more versatile control capability such as controls based on pedigree information, past usage of data, past activity of users and versioning information. This further supports dynamic separation of duties and work-flow controls in stand-alone and multi-tenant cloud systems.

In this dissertation, the emphasis is placed on exploring the utility of provenance data to elevate access control mechanisms that can bring about the enhancement of many computing systems' security goals. Specifically, provenance-based access control (PBAC) mechanisms have great potential to become a strong foundation for securing data in single, distributed, and cloud systems. In this line of research, there have been relatively few works which establish effective and standardized models and mechanisms for access control security in such provenance-aware systems.

1.2.1 Enhancing Features of Access Control Approaches

Throughout the history of access control, there have been various notions that were used as basic decision parameters in access control policies such as clearance, role, trust-level, user-relationship, risk, et cetera. These traditional access control mechanisms are built for specific purposes and are not easily configured to address the complex demands associated with these new technologies [17, 41, 42, 84]. While this may be true, it is possible that access control systems, which are built upon provenance data by fully utilizing its unique characteristics, can provide a foundation for new access control mechanisms that are highly capable of supporting features that were not easily achievable with traditional access control solutions.

Dynamic Separation of Duties (DSOD) is a well-known and important concept in cyber security, which has been extensively studied in the literature. For example, in the context of role-based access control, a certain set of roles and the associated permissions can be designated as conflicting and therefore cannot be assumed by a particular user of the system. Various other constraints of conflicting aspects also exist and are discussed in the literature. The published literature mostly assumes that necessary information for enabling DSOD constraints is readily available. As such, there has been little discussion on the tasks of capturing, storing, extracting, and utilizing necessary historical information. Since this information is often in the form of system events history, provenance data is naturally suitable as the source for DSOD-related information. DSOD concerns can be found in many applications. This dissertation illustrates the application of PBAC in addressing DSOD issues through a homework grading system. In brief, in such a system, students can upload

a homework document to the system, after which they can replace it multiple times before they submit the homework. Once it is submitted, the homework can be reviewed by other students or designated graders until it is graded by the teaching assistant (TA).

In addition to the capability to handle traditional DSOD issues, provenance-based access control can enable additional DSOD features that have not been discussed previously. As shown in Chapter 4.2.5, dependency path-aware refers to cases where the units of conflict are not individual roles, objects, or actions. Instead, conflicts can arise between different dependency paths, which more expressively capture meaningful relations in the system. Another feature, past attribute-aware, refers to how context information of past transactions can be utilized for DSOD constraints. Both of these novel features are further discussed in the chapter. The identification of these additional issues can play a significant part in an environment with more dynamic resource interaction such as the cloud.

1.2.2 Security Aspects of Provenance in Cloud Infrastructure-as-a-Service

Cloud computing paradigm has recently risen as a popular approach that allows efficient utilization of computing resources that can simultaneously minimize related costs and achieve massive scalability. The concept has real-world practicality and development efforts are heavily invested by both academic and industrial sectors [8]. One of the important properties of cloud computing is multi-tenancy [56], where resources within a physical system are allocated and divided between tenants. The notion of tenants allows organized and secure administration of resources and management of privileged users/consumers of the resources.

In most recent cloud Infrastructure-as-a-Service management software such as OpenStack, traditional access control mechanisms have been the most common authorization solutions. More specifically, most cloud systems utilize a variation of role-based access control or simplified forms of attribute-based access control to address their authorization requirement. In most cases, these forms of access control mechanisms are sufficient in handling the specific cloud software's authorization requirements. However, this dissertation envisions that more expressive and finer-grained

access control mechanisms are necessary to accompany the ever evolving state of cloud computing. Ultimately, provenance-based access control can be beneficial for the cloud.

Essentially, PBAC can effectively be employed in a multi-tenant Infrastructure-as-a-Service (IaaS) cloud environment. This dissertation defines a tenant from the perspective of a Cloud Service Provider (CSP), as an independent customer of the CSP responsible for paying for services used by that tenant. Payment is the norm in a public cloud while in a community cloud there often will be other methods for a tenant to obtain services. From the perspective of the tenant, a tenant could be a private individual, an organization big or small, a department within a larger organization, an ad hoc collaboration, and so on. This aspect of a tenant is typically not visible to the CSP in a public cloud.

In this environment, users (e.g., Virtual Machine (VM) creators) and data objects (e.g., VM images, VM snapshots, VM instances) are involved in multiple tenants that are being configured with different authorization settings. The utilization of PBAC within a multi-tenant environment under multiple controlling principals can serve to elevate the authorization capabilities of cloud IaaS infrastructures, including but not limited to secure information flow control and prevention of privileges abuse. For example, a VM resource can be created in one tenant, shared and potentially modified in another tenant and then saved as an image for later use. Tenant administrators can specify access control on the shared VM resource based on its provenance data capturing its pedigree in the original tenant. In order to achieve these authorization goals with PBAC, it is essential to enable tenant-awareness in PBAC. Furthermore, it is necessary to develop an infrastructure for adopting PBAC mechanisms into an existing cloud platform. To the author's knowledge, there do not exist effective provenance-based access control mechanisms for cloud computing platforms. The open-source, popular and ever-growing OpenStack platform can benefit greatly from the incorporation of PBAC. It remains to be seen how the incorporation process fares and how it can provide a starting step to enable PBAC in other cloud computing platform solutions as well.

1.3 Thesis

The central thesis of this dissertation is as follows:

Provenance data forms a directed-acyclic graph where graph edges exhibit the causality dependency relations between graph nodes that represent provenance entities. A provenance data model that can enable and facilitate the capture, storage and utilization of such information through regular expression based path patterns can provide a foundation for enhancing access control mechanisms. In essence, provenance-based access control models can provide effective and expressive capabilities in addressing access control issues, including traditional and previously not discussed dynamic separation of duties, in single systems, distributed systems, and within a single tenant and across multiple tenants cloud environment.

1.4 Summary of Contributions

In this dissertation, the following contributions are accomplished:

- A provenance data model that is built on causality dependency of data and system entities and enables the capture and usage of provenance data from potential system transaction events.
- A framework of provenance-based access control models, $PBAC_B$ and $PBAC_C$, which provide a foundation for utilizing provenance data for enhanced and finer-grained access control in single, distributed, and cloud systems.
  - The base ***PBAC*** Model ($PBAC_B$) is an access control model which bases the authorization decision on provenance data. In order to effectively utilize provenance information, the $PBAC_B$ model makes use of a specific provenance data model which captures provenance data in directed-acyclic graph format. This format allows effective ways to extract information through graph traversal queries.
  - The extended model $PBAC_C$ can capture and utilize contextual information associated

with the primary entities of system events as attributes and store these as additional provenance data.

- A framework of architecture variations that allows the incorporation of PBAC mechanisms in a multi-tenant cloud Infrastructure-as-a-Service system such as OpenStack. Essentially, this dissertation identifies a variety of deployment architecture to incorporate provenance-aware as well as PBAC-enabled features into existing cloud environments and discuss the strengths and weaknesses of each variation.

- Proof-of-concept prototypes that incorporate and extend industry standards such as XACML implementation in a single system as well as OpenStack service components including Nova (which is responsible for computing tasks such as allocation and scheduling of virtual resources) and Glance (which is responsible for management of virtual machine images).

1.5 Organization of the Dissertation

Chapter 2 reviews background and related works. Chapter 3 introduces the provenance data model that serves as the foundation of subsequent provenance-based access control models. In Chapter 4, a framework of provenance-based access control models is provided in detail with the implementation and evaluation of a proof-of-concept prototype in a XACML-aware system. Chapter 5 describes and discusses the different architecture variations for adopting PBAC in a cloud Infrastructure-as-a-Service environment with the implementation and evaluation of a proof-of-concept prototype in an OpenStack cloud system. Chapter 6 discusses provenance sharing approaches for PBAC deployment in a distributed, multi-organization system in the context of a group-centric collaboration environment. Chapter 7 provides discussion of future works and concludes this dissertation.

Chapter 2: BACKGROUND AND RELATED WORK

Acknowledgment: The materials in this chapter are published in [62, 63, 67].

This chapter provides background on the concepts and tools which this dissertation utilizes and builds on. In addition, the chapter also summarizes related work in the existing literature.

2.1 Overview of Traditional and Existing Access Control Models

Access control has always played a vital role in the security of a computing system. Earliest access control approaches consist of discretionary access control (DAC) [78] and mandatory access control (MAC) [76]. DAC enforces the control over resources within a system as per resources' owners. Essentially, the owner is responsible for specifying in which manner a particular resource is accessible to specific system users. MAC enforces the control over resources in a partial-ordered lattice of labels and clearances assigned to users and resources. The access is specified through read and writes rules according to the relations between these labels and clearances. Over time, systems have adapted to new demands and evolved. As a result, access control mechanisms have also been required to follow suit.

Role-based access control [35, 79] (RBAC) has been a popular authorization solution in enterprise software and systems. The use of role constructs facilitates permissions and users management. At the same time, RBAC can suffer from the issue of roles explosion. Attempts to enhance RBAC features have driven the development of newer access control models [45, 74, 75, 89, 90].

In recent years, attribute-based access control [4, 44, 46–48] has gained momentum and recognition in the academic and industrial community. In this model, access control decisions are made based on the values of the attributes associated with particular users, resources and other entities of interest within a system. Other works [13–15, 24, 85–88] focus on designing and deploying ABAC in multi-tenant clouds as well as placing various forms of constraints in the model. To a certain extent, provenance-based access control can also be considered a subset of ABAC. However, current ABAC models do not support provenance data and the associated complexity and

unique characteristics that provenance data enables.

Apart from the above mainstream approaches on access control, other variations of access control mechanisms exist within the literature. Notably, the following access control mechanisms are related to the research in this dissertation.

History-based Access Control (HBAC) models provide access control to data objects based on the action and request history of the subjects [9, 11, 34]. In HBAC policies, a subject's request is decided based on what actions and action-requests the subject had performed before. In this context, the main motivation is to differentiate the "goodness" of subjects based on their past behaviors. Such information can be retrieved from provenance data. More specifically, it is captured in transactions data in the model. In contrast, PBAC emphasizes the history of data objects, which creates dependency chains, and utilizes the intrinsic dependencies, which can also be extracted from transactions data, between these objects for access control purposes.

Relationship-based Access Control (ReBAC) for online social graphs can also be built on path patterns of relation edges. Specifically, Cheng et al [29] specify policies that utilizes regular expression-based path patterns of relationship types between graph entities such as users and resources for finer-grained and more expressive access control on online social networks. This dissertation utilizes regular expression-based path patterns to capture the causality dependency relations between graph entities instead of user-to-user, user-to-resource, or resource-to-resource relationships [30, 31, 70, 71].

Usage control [68, 69] (UCON) is another approach that can greatly benefit from provenance information. In essence, having the knowledge of all influencing activities by any involved party on a particular resource can provide additional utility in pre-, ongoing-, and post- obligations and authorizations models. Other works [49, 50] that focus on usage control of data dissemination in a distributed-systems environment can also benefit from the information provided by provenance data.

2.2 Dynamic Separation of Duties Variations

As mentioned earlier, the use of provenance data in access control brings about the achievement of previously unexplored features including new variants of dynamic separation of duties (DSOD), which is an essential feature of access control. This section provides a summary of existing DSOD works in the literature.

Separation of Duties (SOD) has long been studied and accepted as a fundamental approach to prevent fraud and privileges misuse and abuse. Two major variations of SOD, Static SOD (SSOD) and Dynamic SOD (DSOD), have been mainly discussed in the context of Role-based Access Control [79]. Both SSOD and DSOD have demonstrated usefulness and effectiveness in that domain, as evident by the extensive literature.

SSOD has a major limitation as it mainly deals with role assignment and thereby cannot address issues that arise within a dynamic active session. The original concept of DSOD addresses this limitation. Yet, this approach is also limited in its narrow role-centric scope as it is solely concerned with role activation. For expansion, researchers have proposed variations of DSOD, each of which addresses a separate issue. The variations include Object-based (ObjDSOD), Operational (OpsDSOD), and to a broader extent, History-based DSOD (HDSOD) [38,83]. These approaches rely on the history information of system events, which is assumed to be readily available. However, there lacks exact specifications on how such information can be captured and utilized. Provenance data naturally provides such information and its unique characteristics enable even more sophisticated DSOD features that have not been recognized so far in the literature. Therefore, this dissertation proposes a DSOD approach that utilizes provenance information. The expressive power of provenance utilization can further enable finer-grained DSOD policies and address other DSOD-related issues such as object-based conflicts and work flows.

Table 2.1 shows several classic DSOD variations that are identified in the literature [38,73,83] and some of their distinguishing characteristics. So far most of the published research on DSOD, and more generally SOD, has been in conjunction with RBAC. In particular, SOD is developed

around the main concept of dividing a business task into smaller sub-tasks or other identifiable units such as actions, each of which is assigned to individual roles. These roles are classified into conflicting role sets from which policies can be specified and enforced to achieve SOD.

Table 2.1: DSOD Variations and Features

	Simple DSOD	ObjDSOD	OpsDSOD	HDSOD	TCE
Per Role	✓	✓	✓	✓	✓
Per Action		✓	✓	✓	✓
Per Object		✓		✓	✓
Task-aware			✓	✓	✓
Order-aware				✓	✓
Weighted Action-aware					✓

A definition of simple DSOD is given in [83]. Informally, the concept can be described as follows. Given a set of conflicting roles in a RBAC system, no single user can activate two or more roles from this set at the same time. For example, in the homework grading scenario, no user should be able to activate the roles Student and Reviewer at the same time. This notion of DSOD is limited in that it cannot support per-action control and therefore different variations have been introduced [83].

One variation of DSOD is Object-based DSOD (ObjDSOD) [83]. Informally, the concept can be described as follows. Given a set of conflicting roles and a set of conflicting actions allowed in the conflicting roles, while a user may have conflicting roles activated at a given time, a user is not allowed to perform an action allowed in a role on an object if she performed a conflicting action allowed in conflicting roles on the same object. Here, the focus is on a singular data object upon which conflicting actions are considered. For example, a user should not be allowed to review the homework object that user submitted earlier while the user is allowed to review other homework.

Operational DSOD (OpsDSOD) [83] is another approach that can be described as follows. Given a set of conflicting roles and a set of actions allowed in the conflicting roles, while a user may be assigned to the conflicting roles, the user is not allowed to perform all the actions allowed by the conflicting roles if the union of the actions can complete a particular task. For example, suppose students in a business course are required to participate in an online role-playing project

and provide inputs as different roles such as CEO and CTO to finish a given task. While students can acts as multiple roles in the project, one student cannot perform all the actions of roles where the roles are necessary to finish the required task.

History-based DSOD (HDSOD) [83] is defined as the combination of ObjDSOD and OpDSOD to allow finer control that can be both object and task aware. More specifically, while operational DSOD prohibits users performing a set of action types that can finish a particular task, it is object-unaware and does not distinguish certain actions that are performed in a task for different objects. History-based DSOD resolves this issue by combining object-based DSOD and operational DSOD. In other words, it limits users from performing certain actions for a task, say T1, if the actions are identified as necessary for another task T2 and the user performed other actions which together with the attempted actions can complete the task T2. This issue is resolved in history-based DSOD by allowing object-based control. One added characteristic of history-based DSOD is the consideration of order-dependent sub-task sequences for identifying SOD conflicts. Order-dependent conflicts arise when the sub-tasks are allowed or obliged to occur in a certain order. For example, homework should always be submitted before it can be reviewed.

Transaction Control Expression (TCE) [73] is another traditional approach toward DSOD issues which possesses its own interesting features. TCE is flexible in that it can cover a wide range of DSOD features exhibited by other DSOD variations. At the same time, it also assumes a readily available system-maintained history to rely on for access decisions. TCE also introduces the notion of weighted sub-tasks. From this point of view, each sub-task is assigned an integer value as a weight. The conflict is then determined by whether the total weight of actions exceeds a particular weight threshold. An example of this can be seen in how peer-review processes are introduced in the Homework Grading System (HWGS). A fellow student can review another student's homework, whereas each review process can be assigned a weight 1. Homework can also be reviewed by designated graders, each of whose reviews can be assigned a weight 2. A TA can only grade a reviewed homework if the combined weight of all review processes exceeds 3, which can happen by various combinations of student and grader reviews.

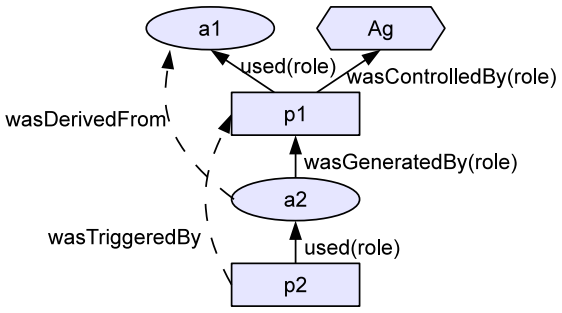

Figure 2.1: OPM Causality Dependencies (©2011 IEEE [67])

2.3 Open Provenance Model

Large numbers of data models exist for capturing and representing provenance data. As each particular data model is designed for a specific application domain in mind, there exists a gap in the characteristics of each data model associated with each different domain. It becomes a necessity for the development of a general provenance data model that can capture the different characteristics belonging to the different domains and provide a common ground to be shared among them [25–28].

Recently, the Open Provenance Model (OPM) [57] core specification v1.1 has been proposed by a group of researchers based on various requirements associated with the usage and employment of provenance in various application domains that were identified in a series of information provenance challenges. The OPM provides a technology-agnostic definition of provenance.

The main concern of OPM is to represent the execution process that led to a particular state of a data object. In essence, OPM aims to capture the causality dependencies of the computing operations, data objects, and execution context between any two object states. In the OPM graphical representation, there are three main types of nodes: artifact which represents a state of a data object, process which represents an operation, and agent which represents an execution context. The direct causality dependency relationships between any pair of these nodes are captured by five different types of edges: *used(Role)*, *wasGeneratedBy(Role)*, *wasControlledBy(Role)*, *wasTriggeredBy*, *and wasDerivedFrom*, which altogether form a directed acyclic graph.

Figure 2.1 illustrates how the above nodes and edges interact in a generic use case. The three types of nodes are distinguished by different graphical representations: artifacts are represented by ellipses, processes by rectangles, and agents by hexagons. The *used(Role)*, *wasGeneratedBy(Role)* and *wasControlledBy(Role)* edges are used to express the system-captured relationships between the nodes. They are represented by solid lines, differentiated by the annotations on the edges. Roles are used to give additional semantics to the associated edges. The *wasTriggeredBy*, and *wasDerivedFrom* edges are represented by dashed lines. They are used to provide additional dataflow-oriented and process-oriented views of the provenance data. They may not be fully captured by the system and may require the user's manual declaration in such cases. Figure 2.1 can be described as follows. The agent **Ag** controlled the process **p1** which used the artifact **a1** to generate the new artifact **a2** which was then used by the process **p2**. Notice the direction of the arrows specifies a causality relationship instead of a data flow. The source of the arc represents the effect while the destination represents the cause. Also, although **p1** used **a1** and generated **a2**, it is not guaranteed that **a2** was derived from **a1** and therefore that needs to be asserted with the *wasDerivedFrom* edge from **a2** to **a1**. The *wasTriggeredBy* edge in Figure 2.1 shows the dependency of processes. This dissertation does not utilize this last type of edge in the model.

To distinguish nodes of the same type that are captured within the same graph, OPM assigns each of the nodes a unique identifier. For example, two instances of a process that perform the same operation are differentiated by their unique identities. The usage of assigned identities is also applied to other components of OPM where distinguish-ability is required under the same context.

OPM is also capable of describing multiple views of the same process at different levels of abstraction within the same graph. A specific abstraction view and its associated semantics are captured in an abstract form of a series of operations which are called "accounts" in OPM. The use of accounts to provide all ranges of description between abstract and detailed levels gives the users efficient utilization of provenance data.

To capture the unique semantics of the operations within a particular application domain, OPM allows more detailed descriptions to be associated with the nodes and edges in the provenance

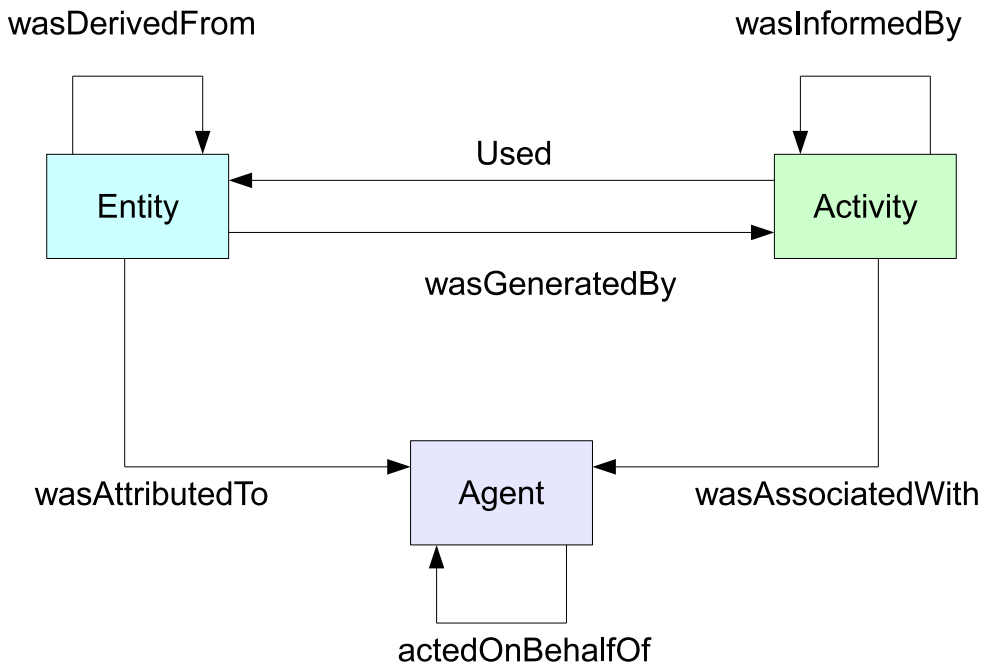

Figure 2.2: Prov-DM Model Components

graph. This is enabled through the annotation framework. The framework allows subtypes of edges to be defined and properties of nodes to be annotated. These subtypes of edges or node dependencies are defined in an OPM profile for a specific application domain.

In any active system, transactions occur and involve subjects, objects, and the corresponding actions describing the interaction between these. The log of all such transactions can be seen as the basis of provenance information. However, without relevant semantics to be assigned to the transactions log, only limited benefits can be gained. For transactions information to be considered useful provenance information, causality dependencies of transaction data should be utilized. Without causality dependency as semantics foundation, it is hard to utilize transaction flows and associated information. The dissertation acknowledges this essential provenance property, for which purpose there requires a suitable provenance model. The work builds on OPM since it provides a foundation for the causality dependencies of provenance data.

2.3.1 The Provenance Data Model (Prov-DM)

Figure 2.2 depicts a resulting model from more recent efforts of the community in realizing a general provenance data model for computing systems. This model, Prov-DM, is built on the initial foundations of OPM. In particular, the model utilizes additional types of causality dependencies to be exhibited between the main model components of Entity, Activity, and Agent. These components correspond directly to the OPM components of Artifact, Process, and Agent. The additional causality dependencies include the direct causality dependency between Entity and Agent, *wasAttributedTo*, and the indirect dependencies of *actedOnBehalfOf* between Agents. Other modifications include change to the semantics of OPM indirect dependencies: *wasTriggeredBy* is changed to *wasInformedBy*, *wasControlledBy* is changed to *wasAssociatedWith*. Note that these modifications are provided to enrich the expressiveness of provenance causality dependencies in order to capture more variety of application domains that can benefit from provenance-aware mechanisms. For the purpose in access control approaches, these modifications are additional features that do not directly influence the approach. Therefore, the provenance data model is built on the simpler OPM which provides sufficient features to enable the approach.

2.4 Standards and Tools

This section provides some background on the standards and associated tools on which the extended PBAC features are implemented.

2.4.1 Resource Description Framework

The Resource Description Framework (RDF) [51] is a framework for expressing information about resources which can be physical, digital or abstract.

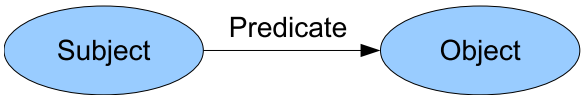

Figure 2.3: RDF Model Components

Overview

RDF lays a common foundation for applications to exchange information on the web for processing without loss of meaning. This can result in broader availability of the information for utility in applications other than those the information is originally intended for. Essentially, RDF provides links between the resources. This allows the users to follow the links and aggregate data about these resources. This capability of RDF allows its utility in different communities of practice. Examples include making information available to different Web pages in such a way that search engines can display the information in enhanced format or third-party applications can perform automatic processes. Another example involves the construction of distributed social networks through the interlinks of RDF descriptions across multiple Web sites. Additional practical uses of RDF can be found in various other domains. This dissertation utilizes the framework to capture provenance data for storage and query capabilities.

RDF Data Model

Essentially, the RDF framework achieves information linkage through utilizing a graph-based data model that captures the relations in the forms of triples. A RDF statement is specified as a triple of (*subject*, *predicate*, *object*) elements to express the relationship between any two resources. Here, the *subject* and *object* elements represent the resources and the *predicate* element represents the essence of the relationship, which is directional from the *subject* to the *object*. A graphical illustration of a RDF statement is shown in Fig 2.3. *Subjects* and *objects* are depicted as nodes and *predicates* as edges. A RDF graph is essentially a set of these statements describing the links and relationship between resources. In an RDF graphs, three kinds of nodes exist as literals, IRIs (Internationalized Resource Identifier), or blank nodes. IRI provides a unique identity as reference

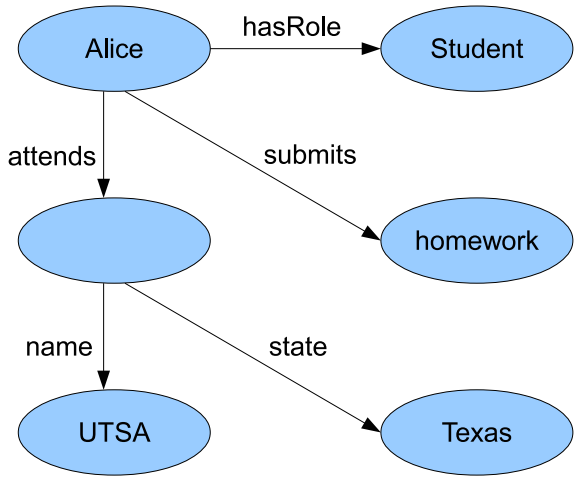

Figure 2.4: A RDF Example

to a particular resource. Literals are used for values such as strings and numbers. Blank nodes are used to express more complex types of nodes. Figure 2.4 illustrates the concept with an example. Essentially, the RDF graph describes some information about Alice who is a student who attends the UTSA University in Texas and submits a homework document. The graph can be captured and stored as the following RDF statements:

<**Alice**><**hasRole**><**Student**>

<**Alice**><**submits**><**homework**>

<**Alice**><**attends**><**opm:agent**>

<**blankNode**><**name**><**UTSA**>

<**blankNode**><**state**><**Texas**>

As RDF captures resources and their relationships in the form of triples, it naturally suits provenance data which often forms directed-acyclic graphs. Utilizing RDF allows the capture of links

between data objects and their lineage. For example, an OPM statement can be naturally be captured as a RDF statement where OPM nodes (processes, agents, and resources) can be represented by subjects or objects, and causality dependency edges can be represented by predicates.

Specifically, the five basic causal edges between three node types of artifact, process and agent, as identified in OPM, can be expressed in RDF representation as follows:

<**opm:process**><**opm:used**><**opm:artifact**>

<**opm:artifact**><**opm:wasGeneratedBy**><**opm:process**>

<**opm:process**><**opm:wasControlledBy**><**opm:agent**>

<**opm:process**><**opm:wasTriggeredBy**><**opm:process**>

<**opm:artifact**><**opm:wasDerivedFrom**><**opm:artifact**>

OPM graphs can then be captured as RDF graphs and stored in databases.

SPARQL Query Language

Information of RDF graphs can be extracted through graph traversal queries. Specifically, the SPARQL query language is designed for that exact purpose. This subsection illustrates a simple SPARQL query on the RDF example given in Figure 2.4.

Suppose the following query is made to ask for the university Alice attends:

```
SELECT ?u
WHERE {
    { Alice attends ?x .}
    { ?x name ?u . }
```

```
}
```

The path pattern on the graph matches the RDF statements and returns "UTSA" as the result.

RDF and SPARQL Tools

RDF is also an industry-compliant standard and has many extant implementations. This dissertation utilizes the open-source tool, Apache Jena, to implement a Java prototype that captures the provenance data of a homework grading system simulation. Additionally, another Python-based tool, RDFLib [7], is used to capture the provenance data of OpenStack components, specifically Nova and Glance. The provenance data allows further implementation and evaluation of PBAC uses in the single and multi-tenant cloud systems, implemented with Jena [21] and RDFLib respectively. The Jena tool has a library (ARQ) that can enable SPARQL query capabilities. RDFLib also implements SPARQL query capabilities.

2.4.2 Extensible Access Control Markup Language

The Extensible Access Control Markup Language (XACML) is an OASIS standard that serves as a general-purpose policy language for access control.

Overview

The XACML framework essentially provides an infrastructure which consists of a language for request/response templates that accompany a policy template that indicates the access control rules used for decision evaluation. In addition, the framework also includes an architecture that specifies all the necessary components that enable the infrastructure as well as enhanced features of access control mechanisms.

The XACML framework is popular as a result of several characteristics. First of all, as a standard language, it is supported by a large community of experts and users whose efforts facilitate the utilization and deployment of XACML in a system and increase the interoperability with other applications using the same language. The language is also generic as it can be used in any en-

vironment and not restricted by specific kind of resource or underlying system, allowing easier policy management across different applications. Furthermore, policies can be distributed across different systems but different users can still appropriately manage sub-components of the policies, allowing the aggregation of these different policies in the final decision. Last but not least, XACML is powerful in the sense that it allows and facilitates extension to the base language to accommodate specific use cases, and as a result increase the usability of XACML.

These characteristics are considered when taking an approach to extend the XACML with PBAC capabilities.

XACML Policy Language

The XACML policy language essentially comprises policy targets and policy rules. The policy targets include subjects, actions, and resources. The policy rules, which are used for access decisions, consist of conditions and potentially obligations. It is possible to express the informal policies for PBAC mechanism with the XACML policy language. Listings A.1 and A.2 display sample templates for XACML requests and responses. Listing A.3 displays a sample template for XACML policies. In essence, the XACML engine matches a particular request to appropriate policies based on certain matching parameters including subjects, actions, and objects. The exact matching parameters are specified in the policy rules, and are performed based on the unique IDs provided in the XACML request and policy matching parts.

XACML Architecture

As depicted in Figure 2.5, the main components of the XACML architecture include the Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Administration Point (PAP), and Policy Information Point (PIP). The PEP is responsible for receiving and enforcing access requests from a front end interface. The PDP receives the transferred request from the PEP and is responsible for evaluating the request. It does so by obtaining the appropriate policy sets and rules from the PAP. It also obtains relevant additional information from the PIP, which is responsible for looking

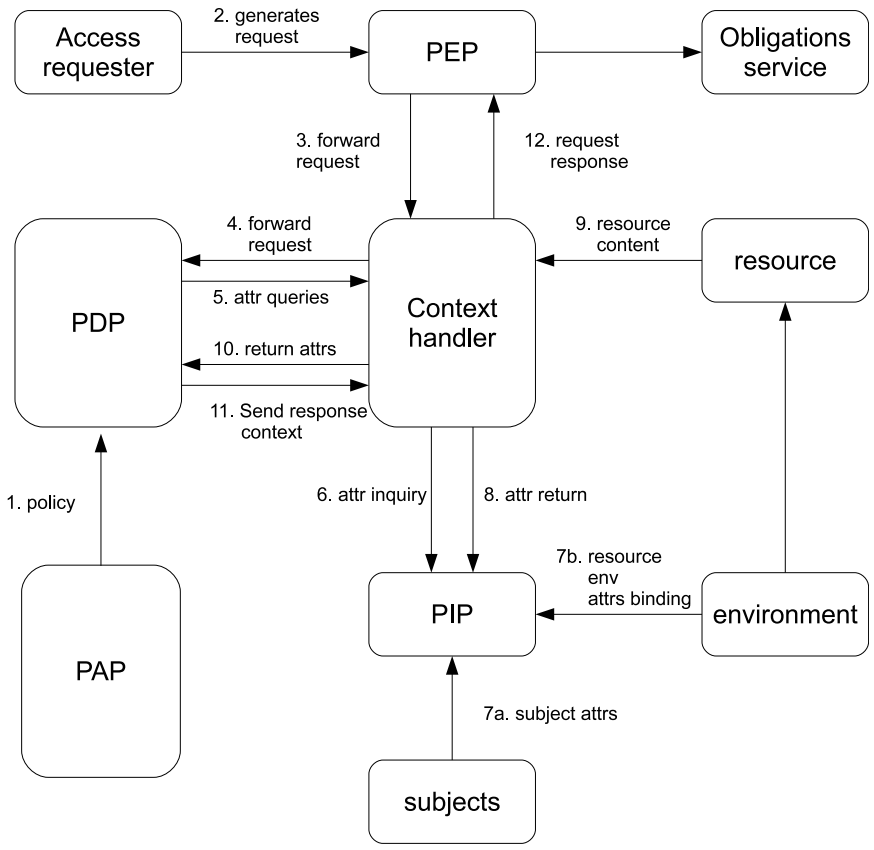

Figure 2.5: A Standard XACML Architecture

up information involving the subjects, objects and the system environment. After the PDP receives the required information, it evaluates the request based on the information and policy rules. The result is then used to form a response with a "yes" or "no" message. All the communication steps are enabled in the background with the Context Handler component. Obligations, which indicate the required actions for granting an access, also play important roles but the concept is outside the scope of this work and henceforth not considered.

2.5 Overview of OpenStack Architecture

In recent years, the popularity of the open-source OpenStack project [5] has risen and elevated its status as being in the same rank with mainstream cloud management platforms such as Amazon Web Services [1], Google Compute Engine [2], and Microsoft Azure Infrastructure Services [3], to name a few. The growth of the OpenStack cloud management platform is further boosted with large interest and intensive investment from various organizations in both academic and industrial sectors. This section provides an overview of the OpenStack architecture and several of its components.

Cloud computing consists of three primary service models: Software-as-a-Service, Platform-as-a-Service, and Infrastructure-as-a-Service (IaaS). Each of the service model type provides different types of resources that can be shared and used by consumers. The OpenStack platform provides IaaS model, which mainly deals with virtual resources that include virtual networks, virtual machine images and instances. Other OpenStack components also provide other service types that relate to the mentioned virtual resources, i.e. monitoring resources usage and graphical user interface.

As depicted on OpenStack website, the logical architecture includes the following components:[1]

- Horizon: provides graphical user interface to users for accessing and using the other OpenStack service components.

[1] http://docs.openstack.org/admin-guide-cloud/content/conceptual-architecture.html

- Nova: provides an API for controlling cloud computing resources and managing the consumers of those resources.
- Glance: provides an API for management of virtual machine images.
- Swift: provides an API for object storage of virtual resources.
- Heat: provides an API for cloud applications orchestration.
- Cinder: provides an API for block storage of virtual resources.
- Neutron: provides an API for defining network connectivity in the cloud.
- Keystone: provides an API for maintenance of users' information and identity for authentication purposes.
- Ceilometer: provides an API for monitoring and collecting information on the movements and usages of virtual resources.

In this dissertation, the focus is on enabling PBAC-enabled authorization service for virtual resources including virtual machine images and instances. Specifically, the emphasis is placed on logical architecture of the Nova and Glance service components over the other described components.[2]

[2]http://docs.openstack.org/admin-guide-cloud/content/logical-architecture.html

Chapter 3: FOUNDATION OF ACCESS CONTROL IN PROVENANCE-AWARE SYSTEMS

Acknowledgement: The materials in this chapter are published in [61, 67].

This chapter discusses the essential characteristics of provenance data in computing systems. From there, it describes a base provenance data model, built on the OPM described in the previous chapter, which allows the utilization of provenance information for enhancement of access control goals. More specifically, the base provenance data model captures the causality dependency relations of provenance information in a way that allows unique constructs to be formed and used as control unit for access control purposes. The chapter demonstrates the use of the model in capturing provenance data of a sample scenario.

3.1 Characteristics of Provenance Data

In recent years, researchers have studied data provenance issues extensively in various computing and application environments. Generally speaking, many of these studies emphasize that data provenance can provide pedigree, usage tracking, versioning capability, et cetera. While this could be true in theory, in a real world system, some of these utilities can be more critical than others. Fundamentally, the utilities of provenance largely depend on the kinds of provenance data that are captured in a system. Capturing complete provenance data for all the operations occurred in a system is neither feasible nor necessary.

In a provenance system, while many computing operations and data dependencies can be captured by the system, there are certain data object (or node) dependencies that can be captured properly only by users' manual declaration. For example, if a user creates a new document from two existing documents, only the user herself can tell whether the newly created document is derived from any or all of the existing documents or not. While this could be done automatically by a system to a certain degree, for example, by comparing contents of these documents, there is no

guarantee for the accuracy of the result since ultimately it is the intention of the user that defines the dependency.

In addition, even with users' manual input, capturing a complete list of provenance data or data dependency is not likely to be possible in a system. This is largely because human memory is not capable of identifying all the source information of their ideas or creations. Consider an example where a researcher writes a scientific article with a list of citations. While the author may try as hard as possible to identify all the sources from where the ideas are derived, some ideas could be simply based on years of study and experience. Hence it is not likely to be possible to generate a complete list of data dependencies.

At the same time, capturing some information of the activities that occurred in a system may not provide any additional utility of provenance. For example, attribute update operations could be critical for authorization process, but capturing these operations in provenance data may not provide any additional utility. Also, it is not necessary to capture provenance data of all activities if they do not contribute in achieving particular goals of a provenance system. Depending on the goals of a provenance system, some activities are not necessary to be considered in the provenance system.

Having these constraints, it is necessary identify the kinds of operations that can be and need to be captured as provenance data, how the captured provenance data can be used in a provenance system and what utilities of provenance can be achieved with the given provenance data. To properly discuss these issues, there requires a specific computing application environment where a set of operations can be specified and expressed and some reasonably significant utilities of the provenance data can be identified. This dissertation emphasizes on the utilization of provenance data for access control purposes in computing systems.

3.2 Base Provenance Data Model

As described and commonly well recognized within the research community, provenance data by its inherent characteristic forms a directed-acyclic graph (DAG). Thanks to this strong property,

provenance data is well suited for capturing and storing information in such a way that traversal-capable queries can be executed to extract useful information in an efficient and convenient manner. Additionally, the DAG structure also enables unique and useful information to be captured and stored. Different types and shapes of sub-graphs can convey specific information that is not usually captured with regular data.

As a result of this specific nature of provenance data, numerous different data models are designed and proposed to suit specific tasks regarding provenance data. Although each of the different variations is capable of catering to the specific need of each domain, the community recognized the necessity for a unified and general data model for provenance. Research community's efforts resulted in the formation of the Open Provenance Model (OPM), and subsequently its successor, the Prov-DM (as described in 2.3.1). Both of these models identify a set of main entities exhibited by provenance data, and additionally, the relationship between any pair of such entities that establishes a causality dependency relation. Such modeling facilitates the capture and extraction of provenance data.

OPM is designed with the purpose to capture the provenance information of any possible resource, either physical or digital, in any given context or application domain. As such, OPM is not specifically designed to capture provenance data in computing systems in a way that allows access control goals to be realized. This section describes a base data model for provenance data that is built upon the concepts of causality dependency relations exhibited by the OPM, which was reviewed in Chapter 2. The model is used to capture provenance data from specific system events and provides a foundation on which it becomes possible to enable, enhance, and enrich access control mechanisms.

3.2.1 Model Components

In a provenance-aware system, it is assumed that all user transactions can be captured by the mechanisms available within the capabilities of the underlying system. Without loss of generality, it is assumed that the capture of a user transaction can be simplified to a construct of the form

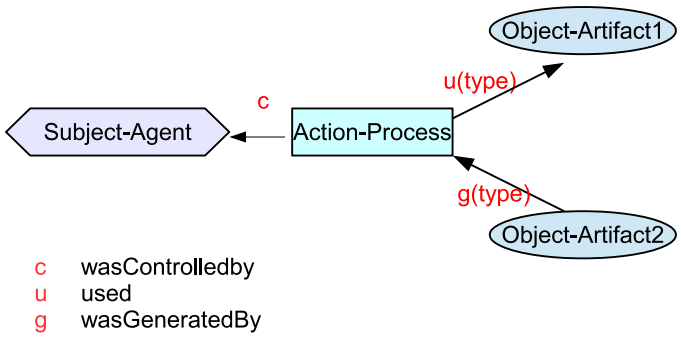

Figure 3.1: A Base Provenance Data Model

(*Subject, Action, InputObject, OutputObject*). Depend on the use cases, either *InputObject* or *OutputObject* can be optional (but not both). Note that transaction data is different from a request in that transaction data knows exactly what objects are used and generated, while a request does not include any objects that are generated from the performed request.

The **base provenance data** (PD_B) stores user transactions data that are captured as a result of performed actions and form a directed graph. Each transaction is stored as a set of triples that consists of two entities and one causality dependency.

The main entities to be captured from system events are classified into three types, each of which is represented as a vertex or node on the graph, and depicted with different geometric shapes:

- Subjects correspond to OPM agents and are graphically depicted by hexagons.
- Actions correspond to OPM processes and are graphically depicted by rectangles.
- Objects correspond to OPM artifacts are graphically depicted by ellipses.

The causality dependency relations between several pairs of node entities are captured by edges of three main types:

- *wasGeneratedBy* captures the relation between an Object entity and an Action entity. This relation specifies the object as an output of the action.

- *wasControlledBy* captures the relation between an Action and a Subject. This relation specifies the subject as performing the action.

- *used* captures the relation between an Action and an Object. This relation specifies the action uses the object as an input.

The causality dependency utilizes three basic dependency types of 'wasControlledBy', 'wasGeneratedBy' and 'used' out of five causality dependencies identified in OPM [57]. The remaining two dependencies are not used in the base provenance data since they are indirect dependencies that either can be system-computed from the direct dependencies or are user-declared. In the context of a specific use case scenario, each type of causality dependency can carry different semantics. OPM allows a notion of role for distinguishing these semantics of three base dependencies. The model identifies this notion as "subtypes" instead of "roles" and utilizes them for only 'wasGeneratedBy' and 'used' dependencies to specify how objects are generated or used. The model does not allow "subtypes" for 'wasControlledBy' since it is assumed that there is only one acting user per action instance. For example, suppose a user $u1$ appended object $o1$ to object $o2$. The transaction data contains $< u1, append1, (o1, o2), o1v2 >$ and corresponding provenance data contains $< append1, u1, wasControlledBy >$, $< append1, o1, used(source) >$, $< append1, o2, used(ref) >$ and $< o1v2, append1, wasGeneratedBy(append) >$. Here, $o1$ and $o2$ are input objects, $o1v2$ is the output object and $source, ref, append$ are subtypes.

The identification of subtypes enriches the ways the behavior of system entities can be expressed. Specifically, different path patterns of dependency edges (or **dependency path patterns**) ($DPATH$) can be used to capture such behavior which represent the provenance information of the entities. In addition, it is also possible to capture relationships that arise naturally from the causality dependency of provenance data captured in different application domains. For example, from a particular object, a path pattern which states which action the object "was generated from" and which subject the process "was controlled by" can provide the information on the user's relation with that object. The expressiveness of dependency path patterns can be further enriched with

the use of regular expressions. For example, the following regular expression-based dependency path pattern $wasGeneratedBy(replace) + .wasControlledBy$ specifies a dependency path pattern which matches any path that starts from an object being "replaced" one or more times and ends at all subjects that performed the "replace" action instances. In access control aspect, the knowledge of this behavior and relationship information can be of beneficial use.

The base provenance data model provides efficient usages of provenance data and also supply essential tools in laying a foundation for access control in provenance-aware systems [61]. The general foundation for access control based upon provenance data via semantic constructs called abstracted **dependency names** (DN) for dependency path patterns. Essentially, these constructs provide abstractions over semantic relationships between multiple data objects. These meaningful dependency name constructs can be used as effective control units for access control policy specification and enforcement.

Dependency lists (DL) are constructed as pairs of abstracted dependency names and corresponding expressions of dependency path patterns. Each $dn \in DN$ is paired with, and defined by, exactly one $dpath \in DPATH$, where $dpath$ may use other previously defined dn's. Recursive or cyclic definitions are not permitted so each $dpath$ can be reduced to a regular expression using only base dependency types by expanding the dn definitions inline. There can be object dependency lists and acting user dependency lists. Object dependency lists include dependencies between objects and other entities in provenance data such as other objects, acting users (agents in OPM), or action instances. Likewise acting user dependency lists include dependencies of acting users. The proposed model considers only object dependency lists as object dependency is an essential notion of the base provenance-based access control model.

3.2.2 Model Specifications

1. S, A, AT, O and OR are subjects, action instances, action types, objects, and object roles respectively.

2. G, U, G^{-1} and U^{-1} are sets of role-specific variations of 'wasGeneratedBy' and 'used' de-

pendencies and matching sets of inverse dependencies, respectively.

3. $\{$'c', 'c^{-1}'$\}$ is the set of 'wasControlledBy' dependency and its inverse dependency.

4. Base provenance data PD_B forms a directed graph and is formally denoted as a triple $< V_B, E_B, D_B >$:

   - $V_B = S \cup A \cup O$, is a finite set of subjects, action instances, and objects that have been involved in transactions in the system and are represented as vertices;
   - $D_B = \{$'c'$\} \cup U \cup G \cup \{$'c^{-1}'$\} \cup U^{-1} \cup G^{-1}$, is a finite set of base dependency types;
   - $E_B \subseteq \{(A \times S \times$ 'c'$) \cup (A \times O \times U) \cup (O \times A \times G) \cup (S \times A \times$ 'c^{-1}'$) \cup (O \times A \times U^{-1}) \cup (A \times O \times G^{-1})\}$, denoting dependency edges, is the set of existing base dependencies in the provenance data.

5. DN_O, disjoint from D_B, is a finite set of abstracted names for dependencies of objects.

6. Let Σ be an alphabet of terms in $D_B \cup DN_O$. The set $DPATH$ of regular expressions is inductively defined as follows:

   - $\forall p \in \Sigma, p \in DPATH; \epsilon \in DPATH$;
   - $(P_1|P_2), (P_1.P_2), P_1*, P_1+, P_1? \in DPATH$, where $P_1 \in DPATH$ and $P_2 \in DPATH$.

7. $DPATH_B \subseteq DPATH$, is the set of regular expression using only alphabet of terms in D_B.

8. $DL_O : DN_O \rightarrow DPATH$, defines each $dn \in DN_O$ as a path expression. DL_O is also viewed as a list of pairs of object dependency names and corresponding dependency paths.

9. $\lambda_O : DN_O \rightarrow DPATH_B$, maps each $dn \in DN_O$ to a path expression using only base dependency types $d_b \in D_B$ by repeatedly expanding the definitions of any $dn_i \in DN_O$ that occurs in $DL_O(dn)$.

Discussion

Unlike OPM, the provenance data model makes use of all the matching inverse dependencies of the dependencies that are captured as a result of transactions. Using normal dependencies, provenance data can be traced only backward in time. The inverse dependencies are necessary for traversing some dependency data since, for example, a request to modify an object may need some verification whether its newer version had been viewed or not. This rule can be verified by traversing the provenance data forward in time. It may also require changes in direction multiple times in case, for example, one may want to check whether any of the related objects and object versions was ever viewed or accessed by someone.

For the base data model, note that while acting user dependency information is available in provenance data, it is not likely to be the main information that provenance data captures. In fact, by definition, provenance is history of objects not acting users though it also includes user's activity histories. This restriction is relaxed in the extended provenance data model.

Note that only certain kinds of edges can exist (no O to O edge for example) and only certain labels can be applied to certain kinds of edges (A to S edge must be labeled 'c' for example). By definition each edge is accompanied by its inverse edge to facilitate traversal in forward and backward direction. If the inverse edges are dropped the graph becomes acyclic as in the OPM model.

3.2.3 Use Case Scenarios and Sample Usages

This subsection describes a homework grading scenario and the associated provenance data, as graphically illustrated in Figure 3.2. It then describes the transaction events that occurred and the corresponding provenance data that can be captured utilizing the base provenance data model.

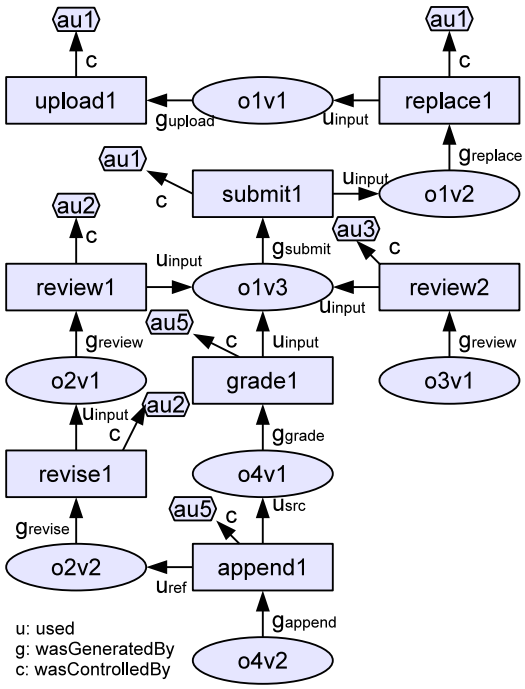

Figure 3.2: Sample Provenance Data for Online Grading System in OPM Graph Representation (©2012 IEEE [67])

A Homework Grading Scenario

Essentially, subject $au1$ represents a student who uploaded an initial homework object, replaced the object with a newer version, and then submitted it. Subjects $au2$ and $au3$ represent other students who reviewed the submitted homework object and generated review objects of the homework object. The review object can then be revised, as performed by $au2$. Subject $au5$ represent a teaching assistant who graded the homework object and then appended a revised review object to the graded homework object.

Sample Transactions and equivalent Provenance Data:

The list below shows some sample transaction data and the matching base provenance data of the above scenario.

1. $(au_1, upload1, o_{1v1})$: $< upload1, au_1, c >, < o_{1v1}, upload1, g_{upload} >$

2. $(au_1, replace1, o_{1v1}, o_{1v2})$: $< replace1, au_1, c >$,
   $< replace1, o_{1v1}, u_{input} >, < o_{1v2}, replace1, g_{replace} >$

3. $(au_1, submit1, o_{1v2}, o_{1v3})$: $< submit1, au_1, c >$,
   $< submit1, o_{1v2}, u_{input} >, < o_{1v3}, submit1, g_{submit} >$

4. $(au_2, review1, o_{1v3}, o_{2v1})$: $< review1, au_2, c >$,
   $< review1, o_{1v3}, u_{input} >, < o_{2v1}, review1, g_{review} >$

5. $(au_3, review2, o_{1v3}, o_{3v1})$: $< review2, au_3, c >$,
   $< review2, o_{1v3}, u_{input} >, < o_{3v1}, review2, g_{review} >$

6. $(au_2, revise1, o_{2v1}, o_{2v2})$: $< revise1, au_2, c >$,
   $< revise1, o_{2v1}, u_{input} >, < o_{2v2}, revise1, g_{revise} >$

7. $(au_5, grade1, o_{1v3}, o_{4v1})$: $< grade1, au_5, c >$,
   $< grade1, o_{1v3}, u_{input} >, < o_{4v1}, grade1, g_{grade} >$

8. $(au_5, append1, o_{4v1}, o_{2v2}, o_{4v2})$: $< append1, au_5, c >$,
   $< append1, o_{4v1}, u_{src} >, < append1, o_{2v2}, u_{ref} >$,
   $< o_{4v2}, append1, g_{append} >$

Object Dependency List DL_O:

It is possible to generate a dependency list of dependency name constructs and corresponding dependency path patterns of the above causality dependency edges.

1. $< wasReplacedVof, g_{replace}.u_{input} >$

2. $< wasSubmittedVof, g_{submit}.u_{input} >$

3. $< wasReviewedOof, g_{review}.u_{input} >$

4. $< wasRevisedVof, g_{revise}.u_{input} >$

5. $< wasGradedOof, g_{grade}.u_{input} >$

6. $< wasAppendedVof, g_{append}.u_{src} >$

7. $< wasOneOfReviewOf, wasRevisedVof*.g_{review}.u_{input} >$

8. $< wasAuthoredBy, wasSubmittedVof?.wasReplacedVof*$
$.g_{upload}.c >$

9. $< wasReviewedBy, wasReviewedOof^{-1}.g_{review}.c >$

10. $< wasCreatedReviewBy, wasRevisedVof*.g_{review}.c >$

11. $< wasGradedBy, wasAppendedVof*.g_{grade}.c >$

The dependency list items 1 - 6 define some dependency names and their dependency path patterns using base dependencies. Items 7 - 11 define additional dependency names using both base dependencies and previously defined dependency names. Based on this object dependency list, it becomes possible to easily specify PBAC policies, as demonstrated in chapter 4.

3.3 Using Provenance for Access Control

The proposed provenance data model allows the identification dependency path patterns and the use of associated dependency names in enabling richer access control mechanisms in provenance-aware systems. More specifically, this approach can be utilized to pursue two directions: *provenance-based access control (PBAC)* and *provenance access control (PAC)*. In essence, PBAC focuses on how provenance data can be used to control access to data, while PAC concerns how access to provenance data should be controlled. Several features that can be enhanced through these approaches are shown in Figure 3.3.

PBAC and PAC are complementary to each other in that PBAC can be used to control access to provenance data and PAC can be used to elevate trustworthiness of provenance data. Furthermore, they both require mechanisms to capture, store and retrieve provenance data. Therefore, while

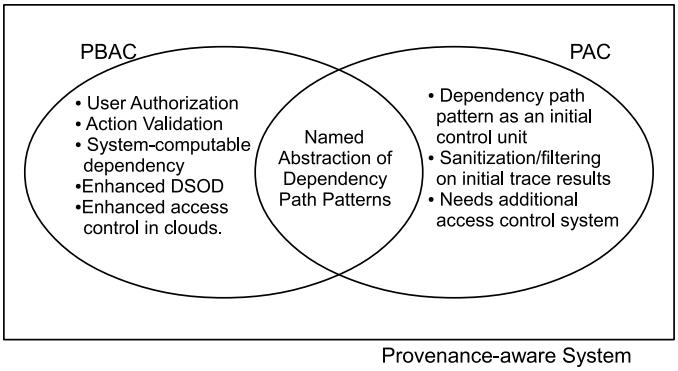

Figure 3.3: Access Control Features of Dependency Path Patterns-based Approach

this dissertation focuses on a foundational model for PBAC, the proposed model also provides a foundation for proper understanding of PAC since it identifies how provenance data should be structured and retrieved.

This section proceeds to briefly describe several insights on the topic of PAC. For the remainder of this dissertation, the main focus is on PBAC.

Insights

There has been considerable attention recently on securing provenance data [16, 17, 19, 20, 41, 42, 65, 84] including access control approaches. Note that while PAC needs some other access controls in play, such as RBAC and perhaps even PBAC can be used for PAC, this could be only a secondary concern of PAC. The main concern of PAC includes unique issues that access to provenance data presents.

In PAC, one of the most significant differences is that provenance data that users want to access are likely to be captured by traversing the provenance graph using some meaningful dependency paths. Furthermore, users may want to access the information that can be derived from the vertices found as a result of the provenance graph traversal. In this respect, there need multi-layer access control evaluations with different granularity.

At its core, PAC utilizes the dependency path patterns found in provenance data as a unit for access request as well as a control unit used in policy specification. The initial and most

essential control should be made through requiring access control to the dependency path patterns themselves. More precisely, access control is done on the semantic named-abstractions of these patterns that can be defined and assigned by system architects. Such assignments can be stored and utilized as a control unit for an access control solution enforced by the system.

Once the dependency path pattern is allowed, a finer grained control is necessary to determine how much of the resulting information (vertices) should be allowed. This requires vertex-level or type of vertex-level access control policies. For example, a student is allowed to know some revisers but not all or she is allowed to know department's information where the revisers belong but not individual revisers' names. Graph redaction or sanitization processes can be performed on provenance data to achieve this end [16, 20].

In addition, if the requester want to access information that are not available in provenance data but can be derived from the resulting provenance information, how to control access to dependency path patterns and how much of the resulting provenance data should be allowed for an access also have to be considered in PAC. These issues are essential for PAC and necessary to be investigated in depth. This work on PBAC can lay a foundation for further investigation on PAC.

Chapter 4: PROVENANCE-BASED ACCESS CONTROL

Acknowledgment: The materials in this chapter are published in [63, 67].

Traditional access control mechanisms often provide access protection on data objects through predefined constructs, e.g., roles in Role-based Access Control (RBAC) and clearances/classifications in Mandatory Access Control (MAC). PBAC aims to provide access control protection on data objects by utilizing the foundational construct of dependency path patterns found in provenance data as described in section 3. While PBAC can support various access control capabilities that are beyond those available in traditional access controls, it should be recognized that in a realistic system deployment, PBAC alone may not be sufficient. Combination with other access control mechanism such as RBAC would bring out more capable access control capabilities. This chapter elaborates on the $PBAC_B$ and $PBAC_C$ models which can enable many enhanced access control features, as identified in Figure 3.3. The chapter also elaborates on an associated proof-of-concept single-system deployed prototype, evaluate the prototype on the homework grading use case scenario, and provide the insights on the experimental results.

4.1 Base PBAC($PBAC_B$) Model

This section defines the base model and provides an access evaluation algorithm to show how access control decision in the proposed model is made. Also, it introduces a policy specification grammar that can support the base provenance-based access control ($PBAC_B$) model. Furthermore, the concepts can be demonstrated with a sample homework grading scenario as introduced in section 3.

4.1.1 Model Components

The proposed provenance-based access control model consists of several core components. Figure 4.1 depicts these components. They are subjects, action instances, action types, objects, object roles, provenance data, dependency lists, policies, and access evaluation function for user autho-

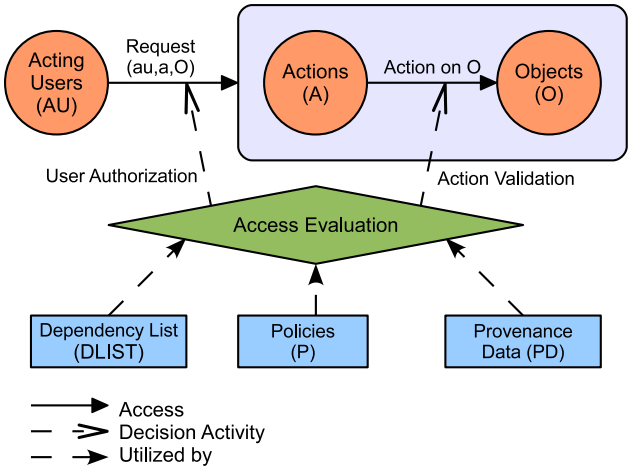

Figure 4.1: $PBAC_B$ Model Components (©2012 IEEE [67])

rization and action validation. Subjects, action instances and types, objects, and provenance data are components of the base provenance data model discussed in chapter 3. Further elaboration can be made on these components in the context of access control.

Subjects (S): represent acting users and interact with the system through initiating access control requests and performing granted requesting actions. Acting users represent human beings who initiate requests for actions against objects. However, provenance data captures acting subjects, not acting users. Capturing subjects can further address DSOD concerns. In a RBAC system, subjects correspond to sessions, where each session is initiated by an acting user activating a subset of his assigned roles. To become role-aware, the model can make the distinction and utilize the notion of subjects in place of acting users.

Action instances (A) are initiated by users through subjects for an access to objects. Provenance-based policies are defined by using **action types** (AT) rather than action instances. AT is a fixed finite set of action types predefined by system architects. It is assumed that a system can derive action types from given action instances.

Objects (O) are resource data that are accessed by subjects. The provenance-based access control model supports object versioning and allows multiple versions of an object. Provenance data captures object versions as vertices. When a transaction modifies an object version, the new

version is represented as a new vertex in provenance data. If an object version is copied or modified into a new object, the output is represented as a new object.

Provenance Data (PD**)** essentially comprises of transaction events that are captured as base provenance data using the base provenance data model introduced in Chapter 3.

A **Request** consists of a subject, an action instance and a set of objects with object roles that are to be accessed. Specific object roles for each action type are pre-defined in the system. In a request, the action type of the action instance may require multiple objects with different **object roles (**OR**)** assigned to them. For example, an $append$ action type requires one reference object and one source object to which the contents of the reference are appended. Object roles are necessary since objects with different roles can, in general, need different rules for granting an action request. Once a request is allowed and performed, the corresponding transaction data is captured and stored as part of the base provenance data. Transaction data includes subject, action instance, input object(s) and output object(s).

Policies include a set of rules that need to be evaluated for granting access. These rules are either for user authorization or action validation. **User authorization rules** specify whether the requester is qualified for the request or not while **action validation rules** specify whether the requested action can be performed against the requested objects. Both types of rules are specified using dependency names. There is only one policy per action type.

Access Evaluation function evaluates a request by utilizing user authorization rules and action validation rules found in the policy for the type of the requested action and returns a Boolean value. The algorithm for access evaluation is provided in Algorithm 4.1.

4.1.2 Policy Specifications

This subsection defines a policy specification grammar PG as shown in Table 4.1. The proposed grammar is designed with simplification to sufficiently capture policies in the sample use case that demonstrate some basic capabilities of the model. The grammar could be further extended to support more sophisticated policies. Policies for the proposed model consist of a set of user

Table 4.1: A Policy Specification Grammar PG

$Policy ::=$ "$allow$" $< Req >$ " $\Rightarrow$ " $< UARules >$ " $\wedge$ " $< AVRules >$ |
"$true$"
$Req ::=$ "(" $< Subject >$ "," $< ActType >$ "," $< ObjRoles >$ ")"
$ObjRoles ::= < ObjRole >$ | $< ObjRole >$ "," $< ObjRoles >$
$UARules ::= < UARule >$ |"(" $< UARules >$ ")"|
$< UARules >< Connect >< UARules >$
$AVRules ::= < AVRule >$ |"(" $< AVRules >$ ")"|
$< AVRules >< Connect >< AVRules >$
$Connect ::= \vee|\wedge$
$UARule ::= < Subject >< oper1 >< PathRule >$
$AVRule ::=$ "|" $< PathRule >$ "|" $< oper2 >< Number >$ |
$< PathRule >< oper3 >< PathRule >$
$PathRule ::=$ "(" $< ObjRole >$ "," $< DName >$ ")"
$oper1 ::=$ " $\in$ "|" $\notin$ "
$oper2 ::=$ " $=$ "|" $\neq$ "|" $\geq$ "|" $\leq$ "|" $<$ "|" $>$ "
$oper3 ::=$ " $=$ "|" $\neq$ "|" $\subseteq$ "
$DName ::= dn_1|dn_2|\ldots|dn_n$
$Number ::= [0-9]+$
$Subject ::= sub$
$ActType ::= at_1|at_2|\ldots|at_m$
$ObjRole ::= o_{role_1}|o_{role_2}|\ldots|o_{role_k}$

authorization rules ($UARules$) and action validation rules ($AVRules$). The overall result of both of these is combined by conjunction. Each rule is defined using path rules that consist of a starting node and a dependency name to which a regular expression-based dependency path pattern is mapped in a dependency list. (See the model definition section below.) A user authorization rule is defined using an acting user, a path rule and an operator and checks (non-)existence of acting user in the vertices found using the path rule, while an action validation rule is defined using one or two of the path rules and an operator and either checks (non-)existence or frequency of vertices in the path or compares two sets of vertices found in the two paths. These three types of rules (one user authorization rule and two action validation rules) are by no means exhaustive but are sufficient to capture the sample use case scenario presented in this dissertation. Each user authorization rule is individually evaluated to a Boolean result. The individual results are then combined using disjunction and conjunction as specified. Action validation rules are similarly individually evaluated and then the results are combined using disjunction and conjunction as specified.

4.1.3 Access Evaluation Procedure

The Access Evaluation Procedure is specified in Algorithm 4.1. In the algorithm, line 2 - 6 shows the rule collecting phase that identifies all the user authorization rules and action validation rules from the policy applied to the action type of the request. The user authorization phase (line 7 - 14) and action validation phase (line 16 - 25) differ in that action validation may need to compute multiple sets of vertices from multiple path rules while user authorization only evaluates one path rule. The user authorization phase compares the acting user of the request to the vertices found as a result of checking the path rules against the base provenance data (PD_B) and returns a Boolean value. An action validation rule evaluates the existence or number of vertices found by a path rule or compares multiple sets of vertices found by multiple path rules, and then returns a Boolean value. User authorization rules (as well as action validation rules) are connected using conjunctive and disjunctive connectives. Once the truth values of these two phases are computed, the algorithm evaluates the final truth value using conjunctive connective.

It is possible to provide a partial analysis for the complexity of this algorithm. It is trivial that all steps outside the FOR loop from line 7 - 13 and the nested FOR loops from line 16 - 24 are in PTIME. Since the number of rules and path rules are finite, all the extraction steps in the two FOR loops except lines 11 and 21 are also in PTIME. Hence the complexity is dominated by lines 11 and 21. Lines 11 and 21 require a tracing algorithm on the provenance graph. In practice this tracing algorithm would be embedded in queries that support regular expression-based path patterns. It can be conjectured that the complexity in some subsets of practical problem space is achievable in PTIME while in other cases may be NP-complete or worse.

As the current state-of-the-art data and query model for OPM graph is RDF [55] and SPARQL, it is possible to associate SPARQL queries that support regular expression through property path constructs with the tracing step in the algorithm. As an OPM graph is essentially a DAG, no paths in the graph can create cycles. [54] shows that the complexity of regular path queries with no cycles is PTIME. An assumption is made on the correctness of the tracing algorithm on which the

Algorithm 4.1 $AccessEvaluation(s, a, O)$

(Rule Collecting Phase)
$at \leftarrow a$'s action type
$p \leftarrow \gamma(at)$
$RULE_{UA} \leftarrow$ user authorization rules $UARule$ found in p
$RULE_{AV} \leftarrow$ action validation rules $AVRule$ found in p
(User Authorization Phase)
for all $rules$ in $RULE_{UA}$ **do**
    Extract the path rule $(ObjRole, DName)$ from $rules$
    Determine the object $o \in O$, whose role is $ObjRole$
    Extract dependency path expression $dpath_b$ in $DPATH_B$ from $DName$ using λ_O function
    Determine vertices by tracing base provenance data PD_B through the paths expressed in $dpath_b$ that start from the object o using δ_O function
    Determine the truth value by evaluating the result against the rule
$UAuth \leftarrow$ a combined truth value based on conjunctive or disjunctive connectives between rules
(Action Validation Phase)
for all $rules$ in $RULE_{AV}$ **do**
    Extract path rules $(ObjRole, DName)$ from $rules$
    for all path rules extracted **do**
        Determine the object $o \in O$, whose role is $ObjRole$
        Extract dependency path expression $dpath_b$ in $DPATH_B$ from $DName$ using λ_O function
        Determine vertices by tracing base provenance data PD_B through the paths expressed in $dpath_b$ that start from the object o using δ_O function
    Determine the truth value by evaluating the results of all the extracted path rules
$AVal \leftarrow$ a combined result based on conjunctive or disjunctive connectives between rules
Evaluate a final truth value of $UAuth$ and $AVal$ using conjunctive connective

correctness of the access evaluation algorithm relies. In particular, it is assumed that all vertices reachable by the query embedding regular path pattern are returned. The correctness of extracting rules and path rules is ensured from the policy language construct. Correctness of the two FOR loops can then be done with loop invariant.

4.1.4 Model Specifications

Based on the core components and formal specifications identified in Chapter 3, it is possible to formally define a base model for PBAC as follows.

1. S, A, AT, O and OR are subjects, action instances, action types, objects, and object roles respectively.

2. G, U, G^{-1} and U^{-1} are sets of role-specific variations of 'wasGeneratedBy' and 'used' dependencies and matching sets of inverse dependencies, respectively.

3. $\{$'c', 'c^{-1}'$\}$ is the set of 'wasControlledBy' dependency and its inverse dependency.

4. Base provenance data PD_B forms a directed graph and is formally denoted as a triple $< V_B, E_B, D_B >$:

   - $V_B = S \cup A \cup O$, a finite set of subjects, action instances, and objects that have been involved in transactions in the system and are represented as vertices;
   - $D_B = \{\text{`}c\text{'}\} \cup U \cup G \cup \{\text{`}c^{-1}\text{'}\} \cup U^{-1} \cup G^{-1}$, a finite set of base dependency types;
   - $E_B \subseteq \{(A \times S \times \text{`}c\text{'}) \cup (A \times O \times U) \cup (O \times A \times G) \cup (S \times A \times \text{`}c^{-1}\text{'}) \cup (O \times A \times U^{-1}) \cup (A \times O \times G^{-1})\}$, denoting dependency edges, is the set of existing base dependencies in the provenance data.

5. DN_O, disjoint from D_B, is a finite set of abstracted names for dependencies of objects.

6. Let Σ be an alphabet of terms in $D_B \cup DN_O$. The set $DPATH$ of regular expressions is inductively defined as follows:

   - $\forall p \in \Sigma, p \in DPATH; \epsilon \in DPATH$;
   - $(P_1|P_2), (P_1.P_2), P_1*, P_1+, P_1? \in DPATH$, where $P_1 \in DPATH$ and $P_2 \in DPATH$.

7. $DPATH_B \subseteq DPATH$, is the set of regular expression using only alphabet of terms in D_B.

8. $DL_O : DN_O \rightarrow DPATH$, defines each $dn \in DN_O$ as a path expression. DL_O is also viewed as a list of pairs of object dependency names and corresponding dependency paths.

9. $\lambda_O : DN_O \rightarrow DPATH_B$, maps each $dn \in DN_O$ to a path expression using only base dependency types $d_b \in D_B$ by repeatedly expanding the definitions of any $dn_i \in DN_O$ that occurs in $DL_O(dn)$.

10. PE is a language specified in the policy expression grammar PG.

11. $P \subseteq PE$, is a finite set of policies.

12. $\gamma : AT \rightarrow P$, a mapping of an action type to a policy.

13. $\delta_O : O \times DPATH_B \rightarrow 2^{V_B}$, a function mapping an object and a base dependency path to vertices in PD_B such that $o_2 \in \delta(o_1, dpath)$ iff there exists a path in PD_B from o_1 to o_2 whose edge labels form a string that satisfies the regular expression $dpath$.

Definitions 2-3 define base dependencies which are the building blocks of base provenance data as defined in definition 4. In definition 4, these dependencies can be used only between certain kinds of vertices. For example, 'c' and 'c^{-1}' can be used to connect an action instance to an acting user and an acting user to an action instance, respectively. The simplicity and effectiveness in policy specification and access control management are achieved with the utilization of dependency names and matching dependency paths in dependency list (DL_O) as shown in Definition 5-9. Definitions 10-12 provide the means for defining policies and attaching them to action types. Definition 13 defines the δ_O function necessary for access request evaluation with respect to the given PD_B.

4.1.5 A Case Study

The case study on the homework grading system, as introduced in section 3, can be further extended with the following policies:

1. Anyone can upload a homework document.

2. A user can replace an old version of a homework document with a new version (versioning control) if the user is the author of the old version and the old version has not been submitted.

3. An author can submit her homework (origin-based control) if it was not submitted already.

4. A user can review only a submitted homework (work-flow control) if she is neither the author nor one of the existing reviewers of the homework (dynamic separation of duty) and the homework has been reviewed less than 3 times and not been graded.

5. A review can be revised if the user created the review and the referred homework is not graded yet.

6. A homework document can be graded if it was reviewed at least 2 times.

7. A review can be appended into a grade if the acting user created the grade and the review was made for the homework that the grade is made against.

In PBAC, policies are defined using dependency names. These dependency names are defined in the dependency list and mapped to a combination of other dependency names and paths which eventually can be expressed in regular expressions ($DPATH_B$) that only utilizes base dependency types (D_B). It is possible to utilize the sample object dependency list, as introduced in section 3, to construct policies for the sample case shown below:

Sample Policies:

1. $allow(au, upload, o) \Rightarrow$ true.

2. $allow(au, replace, o) \Rightarrow au \in (o, wasAuthoredBy) \wedge$
   $|(o, wasSubmittedVof)| = 0.$

3. $allow(au, submit, o) \Rightarrow au \in (o, wasAuthoredBy) \wedge$
   $|(o, wasSubmittedVof)| = 0.$

4. $allow(au, review, o) \Rightarrow au \notin (o, wasAuthoredBy) \wedge$
   $au \notin (o, wasReviewedBy) \wedge$
   $|(o, wasSubmittedVof)| \neq 0 \wedge$
   $|(o, wasReviewedOof^{-1})| < 3 \wedge$
   $|(o, wasGradedOof^{-1})| = 0.$

5. $allow(au, revise, o) \Rightarrow au \in (o, wasCreatedReviewBy) \wedge$
   $|(o, wasOneOfReviewOf.wasGradedOof^{-1})| = 0.$

6. $allow(au, grade, o) \Rightarrow ((|(o, wasReviewedOof^{-1})| \geq 2 \wedge$
   $|(o, wasGradedOof^{-1})| = 0).$

7. $allow(au, append, o_{src}, o_{ref}) \Rightarrow au \in (o_{src}, wasGradedBy) \wedge$
   $(o_{src}, wasGradedOof) = (o_{ref}, wasOneOfReviewOf)$.

It is assumed users can access only the newest object versions in the system. If not, policies like (2) in the sample need an additional rule to make sure the object or the newer versions of the object have not been used for a submission.

Discussion

Although provenance-based access control utilizes provenance data to make access decision, it is likely the case that a real world system also requires other forms of access control systems together with PBAC. For example, consider a homework review and grading example in an online course management system presented in the sample case study. PBAC can support policies such as only the user who uploaded a homework document can replace it with a newer version or can submit it, the user who submitted a homework document cannot review the homework, or a user can append reviews to a grade report only if the review was completed for the homework. This application system is likely to additionally utilize role-based access control to enforce policies such as only students can submit homework or review other student's homework and only instructors can grade a homework document or append reviews to a grade report. For this reason, the policies in the proposed model are defined as necessary rather than sufficient for access, since additional non-provenance based policies may also come into play.

4.2 Contextual PBAC ($PBAC_C$) Model

The base PBAC model, $PBAC_B$, provides a foundation for enhanced access control mechanisms that utilize provenance information. While the $PBAC_B$ model certainly facilitates access control issues, including dynamic separation of duties and work-flow control, it is restrictive in the type of provenance data the model can capture, store, and extract. As a result, not all DSOD features identified in the Table 4.2 can be supported in the current $PBAC_B$ model. To address these issues, an extension model, $PBAC_C$, that utilizes contextual information of subjects, actions and objects

(which are captured as attributes and anchored to actions in provenance data) is proposed. Additionally, the base provenance data model, as discussed in Chapter 3, is also extended to capture the contextual information attributes.

Table 4.2: DSOD Variations and Features with PBAC (©2013 IEEE [63])

	Simple DSOD	ObjDSOD	OpsDSOD	HDSOD	TCE	***DSOD in PBAC***
Per Role	✓	✓	✓	✓	✓	✓
Per Action		✓	✓	✓	✓	✓
Per Object		✓		✓	✓	✓
Task-aware			✓	✓	✓	✓
Order-aware				✓	✓	✓
Weighted Action-aware					✓	✓
Dependency Path Pattern-aware						✓
Past Attribute-aware						✓

4.2.1 Model Components

Figure 4.2 provides an overview of the extended provenance-based access control model. This subsection provides more detailed descriptions of various newly added and modified components, which form the essential parts of the extended model. The subsection also discusses the interaction of these model components in an access control context.

Components

Essentially, the $PBAC_C$ model includes the following components.

Subjects (S) represent acting users and interact with the system through initiating access control requests and performing granted requesting actions.

Actions (A): represent the set of operations on data objects. These actions are supported by a Provenance-aware System (PAS) in the sense the corresponding provenance information can be captured, represented, stored, and extracted.

Objects (O): represent all data pieces that are stored and utilized by the system. Objects are the main target of protection around which access control mechanisms are built.

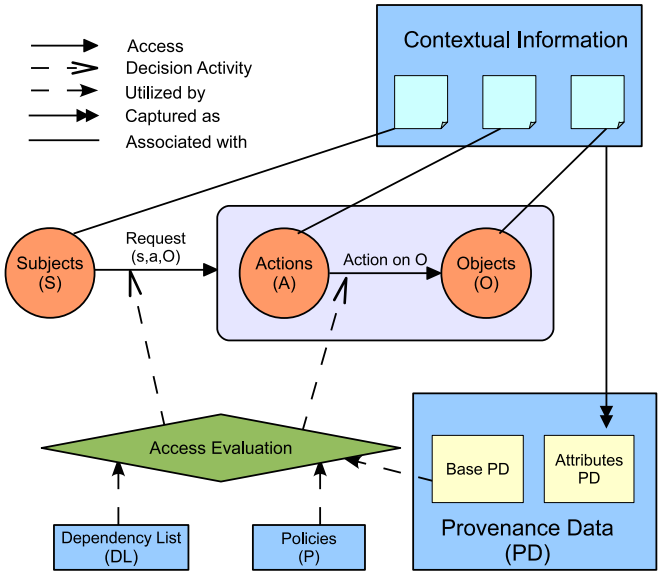

Figure 4.2: $PBAC_C$ Model Components (©2013 IEEE [63])

Requests (R) and **Policies (P)**: represent the initiation for access to resources and the applicable rules that determine the responses to such requests. Object Roles (OR) are constructs that enable more precise policy rule specification for requests which include multiple input objects as parameters. In addition, provenance information is utilized in assisting this process.

Dependency List (DL): contains all system-specific predefined pairs of **Dependency Name (DNAME)** and **Dependency Path (DPATH)**. Dependency names are semantic abstractions that are assigned to specific patterns of edges, termed dependency paths, which hold significance in the context of the specific application domains. These constructs facilitate policy specifications.

Provenance Data (PD): provenance data captures user transactions in a directed acyclic graph structure which allows edge-traversal enabled queries to be performed on a graph node and obtain resulting sets of vertices that provide lineage, versioning, and other provenance-related information. Provenance Data comprises two subtypes:

- Base Provenance Data: represents the causality dependencies between request components that are captured as results of an access request being granted and executed.

- Attribute Provenance Data: represents the contextual information that is associated with the

main components of a request (S, A, or O).

Contextual Information (CI): captures state values of the main components of a request at transaction occurrence time. Four different categories of CI are identified:

- Acting Users-related Context: In each application system, an acting user serves as an initiator on access request and associated system events. The context information of acting users can include typical information such as user ID and other forms of attributes.[1]

- Subjects-related Context: While the acting users are responsible for interacting with the system, the subjects are entities that perform actions on behalf of the acting users. In a typical RBAC system, subjects are equivalent to sessions. The contextual information of subjects can include session id, set of activated roles, etc.

- Actions-related Context: The contextual information, which is associated with each action instance that arises from a system event, is unique. The context information that relates actions include temporal aspects, location, weights, etc. Action context information is most suitable for more expressive forms of access control mechanisms.

- Objects-related Context: The object context information can include information such as object size, URI, etc. Although the object context information can also include information about the origin, list of previous versions, etc., these information does not need to be stored as contextual information as such information is likely to be captured and stored in provenance graph and can be retrieved by accessing the stored provenance data.

Essentially, the new components that are extended to the $PBAC_B$ model include CI and the associated Attribute Provenance Data. The other components were similarly defined in $PBAC_B$.

[1]The readers should note that while the acting user is not directly captured as a provenance entity, it is trivial to obtain the acting user information from the subjects that the user activates. Therefore, acting user context is effectively still captured in provenance attribute data.

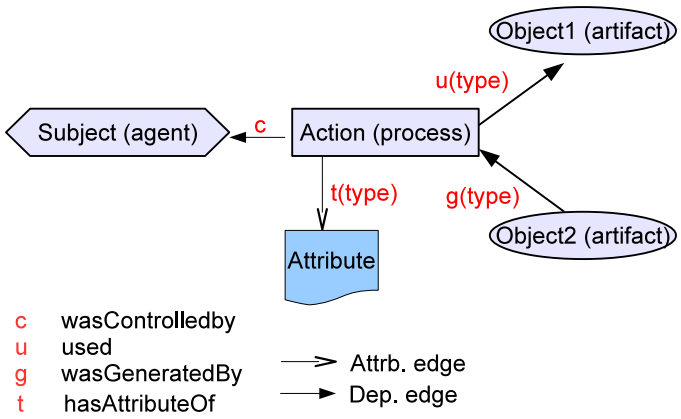

Figure 4.3: A Contextual Provenance Data Model (©2013 IEEE [63])

Interaction

As an access request is generated, the Access Evaluation module extracts the request information to locate the appropriate policies for evaluation. These policies typically make use of the dependency names and associated path expressions, both of which are maintained in the dependency list (DL). The path expressions are constructed using regular expression and carry application-specific semantics. They are used to identify the causality dependencies and contextual information of transaction components (i.e., subject, action, and object) that are stored in the provenance storage. Naturally, they can be used to traverse the graph-based provenance data to extract information that is necessary for making access decision. When an access request is granted, the current contextual information is stored as provenance data. This contextual information is uniquely anchored to the action instance of the access transaction in provenance data.

4.2.2 Contextual Provenance Data Model

In order to capture the necessary contextual information, the base provenance data model, which was discussed in Chapter 3, is extended. This subsection elaborates on this extension of the contextual provenance data model, as illustrated in Figure 4.3.

Contextual Provenance Data Model

Recall that in the base provenance data model, the dependency edges between these vertices are identified as *wasControlledBy* (or c), *wasGeneratedBy* (or g), *used* (or u), *wasDerivedFrom*, and *wasTriggeredBy*. $PBAC_B$ model utilizes mainly u, g, c dependencies for capturing the essential relationships between entities. For more expressive and finer-grained access control, an additional type of graph edge *hasAttributeOf* (or t) and an additional type of vertices, which capture the attribute types and values of the core provenance entities, are introduced. $PBAC_B$ model allows subtypes of u and g edges. The model further allows subtypes of edge t. Variation of edge subtypes is essential for finer-grained and meaningful policy expressions. In figure 4.3, these subtypes are represented with the parameter, *type*, assigned with specific edge subtypes. The base provenance data model notations are depicted together with the new notations introduced to capture the concepts presented in the extended model.

Capturing User Transactions in Provenance Data

This subsection describes the perception of how provenance and attribute data of user transactions can be captured in the system. The concept is discussed in the context of the OPM causality dependencies and additional notations that are discussed in the earlier subsection.

In a provenance-aware system, it is assumed that all user transactions can be captured by the mechanisms available within the capabilities of the underlying system. Without loss of generality, it is assumed that the capture of a user transaction can be simplified to a construct of the form (*Subject, Action, InputObject, OutputObject, ContextualInfoSet*). Depend on the use cases, either *InputObject* or *OutputObject* can be optional (but not both). *ContextualInfoSet* is a set of all contextual information related to the transaction which is configured and selected as necessary by the system design. Such a set should contain entries of the form (*actionInstance,attributeType,attributeValue*) where an *actionInstance* represents unique ID of the action performed. For example, *(upload1,weight,3)* captures the *weight* attribute of the *upload1* action

with the value of 3. The types of attributes and associated values to be captured are anchored to action instances even though attribute types are associated with either subjects/users, objects or action as discussed earlier.

As discussed earlier, while the context information of different transaction components (i.e., subjects, actions and objects) is captured as attribute provenance data, the attributes are anchored to the action instances. For example, within a homework grading system scenario, a transaction *(Subject1, Grade1, HW1, GradedHW1, ContextualInfoSet-Grade1)* can be captured as:

<**Grade1**><**used**><**HW1**>

<**Grade1**><**wasControlledBy**><**Subject1**>

<**GradedHW1**><**wasGeneratedBy(grade)**><**Grade1**>

If the essential ContextualInfoSet-Grade1 consists of the following information (ActingUser:Alice, Subject-Role:TA, Grade-Weight:2, HW1-size: 10MB), they can be captured with the following triples:

<**Grade1**><**hasAttribute(weight)**><**2**>

<**Grade1**><**hasAttribute(activeRole)**><**TA**>

<**Grade1**><**hasAttribute(actingUser)**><**Alice**>

<**Grade1**><**hasAttribute(object-size)**><**10MB**>

Anchoring transaction attributes to an action instance is necessary and particularly useful for effective management of attributes. Further discussion is made in the next section.

4.2.3 Model Specifications

This subsection provides the formal specifications for the extended components identified in the above subsection.

Formal Specifications

1. S, A, AT, O, OR and ATT are subjects, action instances, action types, objects, object roles, and attributes respectively.

2. G, U, G^{-1}, U^{-1}, T and T^{-1} are sets of type variations of 'WasGeneratedBy' and 'Used' dependencies, matching sets of their inverse dependency types, a set of variations of 'hasAttributeOf' attribute types, and a matching set of its inverse attribute types, respectively.

3. $\{\text{`}c\text{'}, \text{`}c^{-1}\text{'}\}$ is the set of 'WasControlledBy' dependency type, its inverse dependency type.

4. Provenance data PD forms a directed graph and is formally denoted as a triple $< V, E, L >$:

   - $V = S \cup A \cup O \cup ATT$, a finite set of subjects, action instances, objects, and attributes that have been involved in transactions in the system and are represented as vertices;
   - $L = \{\text{`}c\text{'}\} \cup U \cup G \cup T \cup \{\text{`}c^{-1}\text{'}\} \cup U^{-1} \cup G^{-1} \cup T^{-1}$, a finite set of dependency type labels and attribute type labels;
   - $E \subseteq \{(A \times S \times \text{`}c\text{'}) \cup (A \times O \times U) \cup (O \times A \times G) \cup (S \times A \times \text{`}c^{-1}\text{'}) \cup (O \times A \times U^{-1}) \cup (A \times O \times G^{-1}) \cup (A \times ATT \times T) \cup (ATT \times A \times T^{-1})\}$, denoting provenance graph edges, is the set of existing dependencies and attributes types in the provenance data. [2]

5. $DNAME$, disjoint from L, is a finite set of abstracted names for a composition of dependency types and attribute types.

[2]The readers should note that only certain kinds of edges can exist (no O to O edge for example) and only certain labels can be applied to certain kinds of edges (A to S edge must be labeled 'c' for example). By definition each edge is accompanied by its inverse edge to facilitated traversal in forward and backward direction. If the inverse edges are dropped the graph becomes acyclic as in the provenance graph.

6. Let Σ be an alphabet of terms in $L \cup DNAME$. The set $DPATH$ of regular expressions is inductively defined as follows:

   - $\forall p \in \Sigma, p \in DPATH; \epsilon \in DPATH;$
   - $(P_1|P_2), (P_1.P_2), P_1*, P_1+, P_1? \in DPATH$, where $P_1 \in DPATH$ and $P_2 \in DPATH$.

7. $DPATH_L \subseteq DPATH$, is the set of regular expression using only alphabet of terms in L.

8. $DL : DNAME \rightarrow DPATH$, defines each $dname \in DNAME$ as a path expression. DL is also viewed as a list of pairs of dependency names and corresponding dependency paths.

9. $\lambda : DNAME \rightarrow DPATH_L$, maps each $dname \in DNAME$ to a path expression using only dependency type and attribute type labels $l \in L$ by repeatedly expanding the definitions of any $dname_i \in DNAME$ that occurs in $DL(dname)$.

10. P is a finite set of informal XACML-compatible policies.

11. An access evaluation procedure $AccessEvaluation(s, a, O)$ which utilizes the following functions:

    - $\gamma : AT \rightarrow P$, a mapping of an action type to a policy.
    - $\delta : \{S \cup A \cup O\} \times DPATH_L \rightarrow 2^V$, a function mapping a vertex (except attributes in ATT) and a dependency path to vertices in PD such that $v_2 \in \delta(v_1, dpath)$ iff there exists a path in PD from v_1 to v_2 whose edge labels form a string that satisfies the regular expression $dpath$.
    - F is a set of evaluation functions which can be used to evaluate a set of vertices. These functions are application-specific and are designed to suit policy requirements of the underlying target system.
    - RF is a set of rule-evaluation functions which can be used on the result returned by a function $f \in F$ and returns a Boolean.

- $RuleCombine : 2^{Boolean} \rightarrow Boolean$, is a function which evaluates a set of Boolean values through conjunction and disjunction.

Discussion

This subsection describes how the contextual provenance data model can be utilized in the $PBAC_C$ model. Specifically, the components (S, A, O, and ATT) are captured as vertices on the provenance graph. Causality dependencies types and subtypes are used to label the directed edges between the vertices of types (S, A, and O). Attribute types and subtypes are also used to label the directed edges between A and ATT vertices. Also, it is currently allowed only incoming edges to attribute nodes and no outgoing edges from them. The u, g, t edge types (but not c) are further differentiated by assigning a subtype. These variations of edges are facilitated to capture different semantics or types of the dependency relationships between corresponding edges. In addition, each edge type is accompanied with an inverse edge type which allows traversal toward the future transaction instances in provenance data.

There are two ways to capture the attribute provenance data in the graph-based data model.

- The attribute data of a vertex s, a or o is stored as an attribute vertex connected to the corresponding vertex.

- The attribute data of a vertex s, a or o is stored as an attribute vertex connected to only the vertex a. Specifically, attributes of s and o are connected to a vertex a that is associated with s, o in the same transaction.

The model in this dissertation chooses the second approach. Contextual information of a transaction is stateful and only meaningful in the context of the associated action instance. Anchoring such stateful information to any other vertices (of either type S or O), which can be stateless, may result in an inconsistent and incorrect policy evaluation. For example, a single subject instance can be involved in multiple different action instances while activating different roles for different actions. In such case, if provenance data stores the list of active roles as an anchored attribute to

the corresponding subject, multiple attributes with different lists of active roles may exist. This then creates difficulty in identifying which list is for which transaction. On the other hand, as there could be only one action instance for each transaction, anchoring attributes to a corresponding action can eliminate this problem.

The readers should note the difference between the query-tracing processes of the $PBAC_C$ and $PBAC_B$ models. Specifically, $PBAC_B$ only allows the starting node, from which a regular path expression query can be traced, to be exclusively O. The model relaxes this restriction and allows the starting node to be any of the three vertex types (S, A, and O).

4.2.4 Policies and Access Evaluation

Based on the model components, access requests can be evaluated and decided following the access evaluation algorithm provided in Algorithm 4.2. Policies can be informally specified using a policy grammar that is modified and incrementally extended from the grammar provided in [67]. A policy specification provides a mechanism for matching a request to a corresponding policy rule-set. Each policy rule utilizes pairs of a dependency name construct, which can be applied with a function γ to arrive at base dependency path patterns, and an associated starting graph node to be applied with a function δ that traces through the provenance graph to obtain a set of resulting nodes. Another function, f, can then be applied on the resulting set of nodes to obtain values that can be used to assist the evaluation of a rule condition with a function rf. All rules decisions are then combined in accordance to a *RulesCombining* procedure that is specified in the policy body.

All the specific evaluation functions associated with a particular instance of request are obtained from the rule specification in the policy body. For example, consider the *sample policy 4* provided in the next section. In this policy, the function f is specified as the sum function and the rf function is specified as "$>=$" 3.

Algorithm 4.2 $AccessEvaluation(s, a, O)$

(Rule Collecting Phase)
$at \leftarrow a$'s action type
$p \leftarrow \gamma(at)$
$RULE \leftarrow$ access evaluation rules $Rule$ found in p
(Evaluation Phase)
for all $Rule$ in $RULE$ **do**
    Extract the path rule $(StartingNode, dname \in DNAME)$ from $Rule$
    Extract $type$ of $StartingNode$ as S, A, or O
    if $type$ is O **then**
        Extract $ObjRole$ of $StartingNode$
        Determine the object $o \in O$, whose role is $ObjRole$
        Set value of $StartingNode$ to o
    if $type$ is S **then**
        Set value of $StartingNode$ to s
    if $type$ is A **then**
        Set value of $StartingNode$ to a
    Extract dependency path expression $dpath \in DPATH_L$ from $dname$ using λ function
    Determine vertices $VSet$ by tracing provenance data PD through the paths expressed in $dpath$ that start from the vertex $StartingNode$ using δ function
    Extract $f \in F$ from $Rule$ and execute it on $VSet$
    Extract $rf \in RF$ from $Rule$ and execute it on $f(VSet)$
    Store $rf(f(VSet))$ in $RuleEval$
Execute $RulesCombining(RuleEval)$ and return the Boolean result

4.2.5 DSOD in $PBAC_C$ Model

This subsection illustrates how the $PBAC_C$ model can be utilized to support traditional DSOD policies. The subsection also identifies and discusses several limitations of current approaches as well as additional unique DSOD and some access control features that can be supported with the $PBAC_C$ model.

Traditional DSOD in $PBAC_C$ Model

This subsection provides a set of informal policies to express the various DSOD constraints in a provenance-aware system. In the following sample policies, several sample dependency names are assumed to be defined accordingly in the dependency list *DL* of the application system. These dependency names can be broken down to a dependency path of basic edge labels (i.e., u, g, c variations), as defined in the model specification, and then utilized in regular path queries. Furthermore, the readers should note that these informal policies can be easily translated to equivalent XACML policies such as the representation of the sample policy 2 as shown in Listing A.3.

Sample Policy 1: represents a *simple DSOD* concern in a single session only for simplicity. In particular, a subject should not be active with the role “Reviewer" to activate the role “Student".

$$\textit{allow(sub,activate,Student)} \Rightarrow$$

$$\textit{Reviewer} \notin \delta \textit{ (sub, performedActionsOf:hasAttributeOf(activeRoles))}$$

In this example, the dependency name $performedActionsOf$ is used to determine all the actions performed by the subject so the resulting action vertices can be used to identify all the active roles of the subject with $hasAttributeOf(activeRoles)$. Then the policy evaluates whether the “$Reviewer$" role is among the active roles found by the δ function.

Unlike the other DSOD variations that were identified earlier, simple DSOD does not require any historical information as it only utilize information of currently active roles of the user of a subject. However, it can be still achieved by using provenance data only even though it may not be ideal to exercise. To achieve this, as shown above, it is first necessary to find all the actions performed by the subject then further find all the active roles of the identified actions.

This approach can also address simple DSOD for multi-sessions where active roles of all active sessions of the same user need to be considered by identifying all the active subjects of the same user in provenance data. To achieve this, a provenance system needs to capture a user’s activities on subject creation and termination. This information can be used to identify all the active roles of currently active subjects of the same user. Another way of achieving this is by simply capturing all the activated roles and deactivated roles of a user then subtracting the deactivated roles from activated roles to get currently active roles of the user. This can be done by (1) identifying an acting user of a requesting subject, (2) finding all subjects that are currently active, (3) finding all roles activated by these subjects, (4) finding all roles deactivated by these subjects, (5) subtracting the deactivated roles from the activated roles, and (6) comparing the resulting roles with the requested role to see a conflict.

Sample Policy 2: represents an *ObjDSOD* concern that requires the requesting subject on replacing a homework document to be activated by the same acting user who activated the subject on uploading it.

$$allow(sub,replace,o) \Rightarrow$$
$$\delta(sub,hasPerformedActions{:}hasAttributeOf(actingUser)) \in \delta(o,wasUploadedBy) \wedge$$
$$count(\delta(o,wasSubmittedVof)) = 0$$

In this scenario, to allow a subject *sub* to perform *(action:replace)* on the object *(o)*, the acting user of *sub* should have performed *(action:upload)* on that particular object, either through the same or different subjects. The acting user of *sub* can be found by tracing provenance graph starting from o and following the path identified with the expression $hasPerformedActions : hasAttributeOf(actingUser)$. The user who uploaded o can be obtained by tracing through the provenance graph starting from o and following path as defined with the dependency name *wasUploadedBy*. In addition, the dependency name *wasSubmittedVof* is used in a separate rule to specify an additional rule that the homework object should not have been submitted as well.

Sample Policy 3: represents a *history-based DSOD* ($HDSOD$) concern by requiring that a request to review a homework document can only be allowed after the object is submitted and before it is graded.

$$allow(s,review,o) \Rightarrow$$
$$count(\delta(o,wasSubmittedVof)) \neq 0 \wedge$$
$$count(\delta(o,wasGradedOof^{-1})) = 0$$

Sample Policy 4: represents a *weighted-action DSOD* concern where the weight of each review process is summed up and the total is used to regulate the access request to grade an object.

$$allow(sub,grade,o) \Rightarrow$$
$$sum(\delta(o,previousReviewProcesses{:}hasAttributeOf(Weight))) >= 3.$$

Dependency names are utilized in all policies for specifying the policy. While careful selection of such named abstractions can convey the semantics of the provenance graph path, the dependency path expressions are ultimately responsible for describing a precise path and used for actual tracing process. The readers should also note that while a dependency path expression can be paired with

any types of starting vertex (either $subject$, $object$, or $action$), for efficiency and effectiveness, one type of starting vertex is preferable to another in different cases. The HWGS example can utilize dependency path expressions with $object$ starting vertices for addressing ObjDSOD and $subject$ starting vertices for simple DSOD, OpsDSOD and HDSOD issues.

Unique DSOD Features in $PBAC_C$ Model

The utilization of provenance data in the form of dependency paths supports the traditional DSOD variations effectively. Furthermore, it also supports novel features that have not been discussed much, if at all, in the literature. This subsection elaborates on these unique features that $PBAC_C$ realizes.

Feature 1: Awareness of Past-Action attribute. The context information of the system components provides insightful information on the current state of the system. The information is useful not only for access control mechanisms in the current state, but can also make an impact on access control usages in future states. $PBAC_C$ provides convenient mechanisms for storing and carrying the state attributes of a current action instance into the future, at which point they are past attributes of the current action instance. This novelty can be demonstrated with the weight-action example.

When an action transaction is executed in the system, it is assigned a weight as specified by the system policy or setting. That weight attribute is stored in the provenance data in association with that particular action instance. In some future state, the system may assign a different weight value to that action type. An access control policy based on weighted actions can base policy rules on either the current weight attribute state or the ones captured in the past action transitions. $PBAC_C$ facilitates this process and can support both type of mechanisms.

Feature 2: Dependency Path Pattern-based DSOD. Traditional DSOD approaches base control on some fixed and restrictive form of control unit, especially actions. With regular expression-based dependency paths, it is possible to achieve much more expressiveness in the specification of the control unit. In particular, it can be expressed a wide variety of path patterns which includes sequences, repetitions, existences, and any combinations of actions. Additionally, the intrinsic

characteristics of provenance data enable versioning of a data object and linkages of distinct but related data objects. This enables the grouping of objects or objects versions with meaningful semantics, on which DSOD constraints (e.g., conflicting actions on current and past object versions instead of a particular object), can be specified.

The identified features above allow a broader family of DSOD policies and constraints to be specified. This goes hand in hand with the complexity a provenance-aware environment and all its huge amount of data can produce. It seems natural that as the amount of data and complexity rises, so do concerns of DSOD involved in such environment. Utilizing provenance constructs such as dependency path patterns and associated dependency names can lay a strong foundation for addressing this phenomenon.

4.2.6 Pre-Enforcement Access Evaluation

Motivating Problems

When a user, represented by a subject, initiates a request for an action on a system resource, a typical procedure consists of several steps as follows:

1. The request is first evaluated for authorization decision. This is typically performed by the Policy Decision Point and requires several components for various information that is necessary for making the decision.

2. If the request is denied, the Policy Enforcement Point delivers the denial result to the user. Otherwise, the granted request action on resource is executed and subsequent outputs/results are also delivered to the user.

3. Once the requested action is successfully completed, corresponding provenance data are generated to capture all information regarding the executed action.

The readers should note that between each of the above procedure steps there exists a time delay. Depend on the nature of each step, such time delay may be small or costly and can heavily

impact the performance of any system that utilizes PBAC approach. Specifically, in a system, there can occur a scenario where subsequent initiated requests can depend on the result of prior requests evaluation and execution. To be more specific, if the authorization of both requests require the same provenance data set for evaluation purpose, then the later request is required to utilize the appropriate provenance data that is updated as result of granting the earlier request. Delaying the evaluation of a request in order to wait for completed enforcement of prior dependent request and related provenance data update can substantially degrade performance. Ideally, it is desirable for the evaluation of a request to be started once the evaluation of the prior request is completed. Waiting for steps 2 and 3 as described above can assure the safety property of the system, but introduces unnecessary overhead which needs to be reduced.

Solution Approaches

In order to allow PBAC evaluation of the next request to start after the previous request decision is returned, some modification to the existing PBACB model can be introduced. Essentially, a new type of provenance data that is temporary can be used. Temporary provenance data keeps track of all requests that have been evaluated but corresponding enforcements had not been executed. A new subtype of the "used" edge, "used(request)", can be used to indicate that a request for an action is instantiated and an attribute edge "hasAttribute(requestEval)" that connects an attribute node that has either "granted/denied" values to denote the request evaluation. Such an update to the provenance data set can incur minimal run-time cost and subsequently be used for evaluation. Once a granted request-action is enforced, a new action node instance can be generated that used the "used(input)" edge to denote successful execution. The temporary provenance data is then obsolete and can then be deleted for efficient space.

While this approach does not reduce the gap between the initial reception of the request to its time of evaluation completion, it does take away the gap between evaluation and enforcement completion. Typically for systems where actions can take a long time to complete execution, this approach does ameliorate such run-time delay.

Sample Scenario

Building on the HWGS scenario, this subsection proceeds to demonstrate the above concepts. Suppose for every submitted homework object, there can only be 2 reviews on that homework object. After 3 reviews processes have been performed, any subsequent review is denied. Suppose Alice submits a review to homework $hw1$. Since $hw1$ has not been reviewed before, access control decision is evaluated to be granted. This can generate the following triples in the provenance data set:

<**review1**><**used(request)**><**hw1**>

<**review1**><**wasControlledBy**><sub_{Alice}>

<**reviewedHw1-1**><**wasGeneratedBy(review)**><**review1**>

<**review1**><**used(hasAttribute(requestEval))**><**granted**>

Next, Bob and Charlie also submit their reviews for $hw1$. Authorization evaluation for Bob's request recognizes that only Alice has reviewed the $hw1$ object before and grants access to Bob. The access decision is stored in the provenance data set as above before the review action is executed by the system. Charlie's request is then processed by the authorization evaluation, which recognizes that Bob's request has been granted and therefore denies Charlie's request.

4.3 An XACML-based $PBAC$ Prototype

In order to demonstrate the feasibility of the PBAC models, a proof-of-concept prototype, which is based on the XACML architecture and associated implementation, is implemented. This section elaborates on the extended XACML architecture, the prototype implementation and evaluations. Insights on the experimental results are then provided.

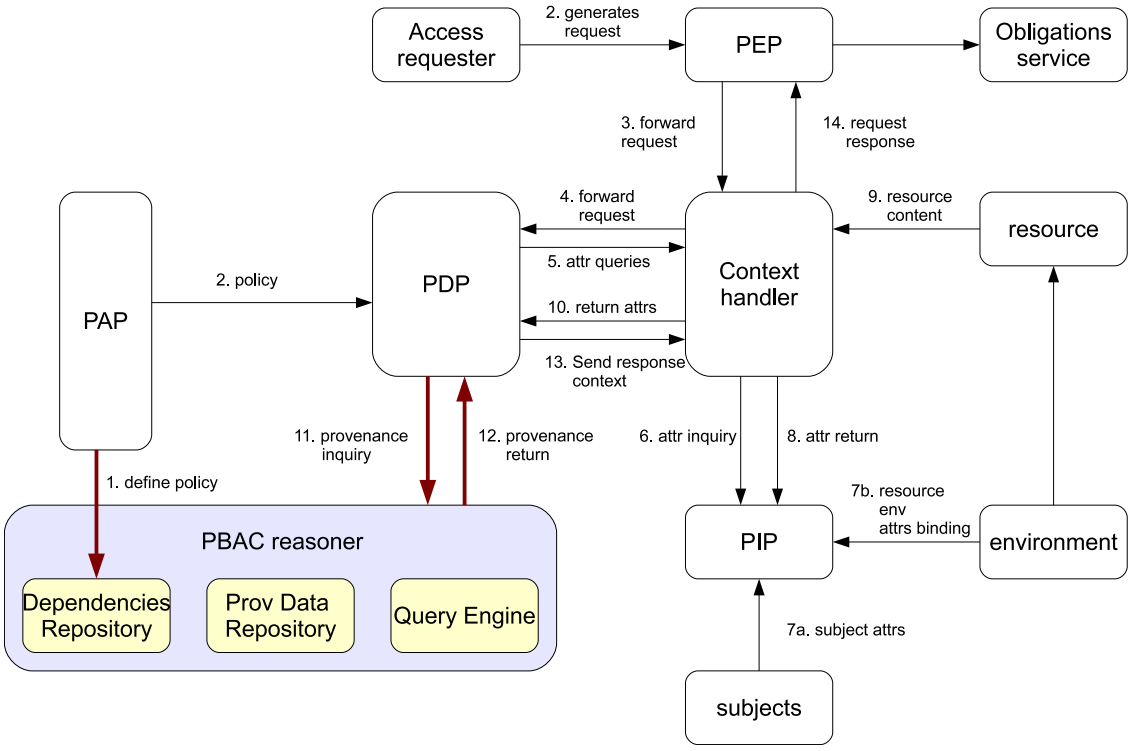

Figure 4.4: An Extended XACML Architecture (©2013 IEEE [63])

4.3.1 An Extended XACML Architecture

As discussed in Chapter 2, the XACML framework is a popular and industry-standard tool that is used in many commercial and research products. This subsection aims to extend the XACML framework with PBAC-enabled components.

PBAC Reasoner

In order to enable PBAC mechanisms in the XACML framework, an extension to the existing XACML architecture, as shown in Fig 4.3, is proposed. Specifically, a new component, the PBAC Reasoner, is introduced. The PBAC Reasoner communicates directly with the PDP and PAP components and is responsible for the specification of provenance-based access control policies. It also provides specific mechanisms for storing and extracting provenance data for access control request evaluation. The PBAC Reasoner component is further composed of three additional sub-

components:

- Dependencies Repository (DR): The dependencies repository component is responsible for storing application-specific dependency lists. This dissertation assumes only one set of dependency path patterns and associating dependency name constructs is at use, as only one system is in concern. However, it is possible to have multiple dependency lists when multiple systems are considered, as in a cloud or distributed environment. The dependency list is utilized by the PAP for policy specification purposes.

- Provenance Data Repository (PDR): The PDR component is responsible for storing captured provenance information. This can include both the base provenance as well as attributes data.

- Query Engine (QE): The query engine is mostly associated with the provenance database employed by the PDR. It is important that the QE is capable of performing regular path queries as provenance graph traversal is essential in the PBAC approach.

4.3.2 Prototype Implementation

The prototype employs various state-of-the-art tools available in the community. It uses Sun's XACML[3] implementation for policy specifications and the existing implementations of the XACML components. The PDR utilizes the Jena framework [21], where provenance graph is stored in RDF-triples [51] format. This dissertation uses Jena-2.7.4 and the corresponding ARQ package [4]. The query engine associating with Jena is ARQ, which was utilized to execute SPARQL [72] queries on the RDF-format provenance graph database. Dependency lists are simply implemented as array-lists of pairs of String values. To enable the communication between the existing components of the XACML framework, specifically the PDP and the PBAC Reasoner components, a specialized function that can be incorporated to the XACML framework at run-time was implemented. The function is essentially an extended class to the *FunctionBase* class available in Sun's framework. The function essentially performs the tasks equivalent to the functions δ elaborated in the model

[3]https://www.oasis-open.org/
[4]http://jena.apache.org/

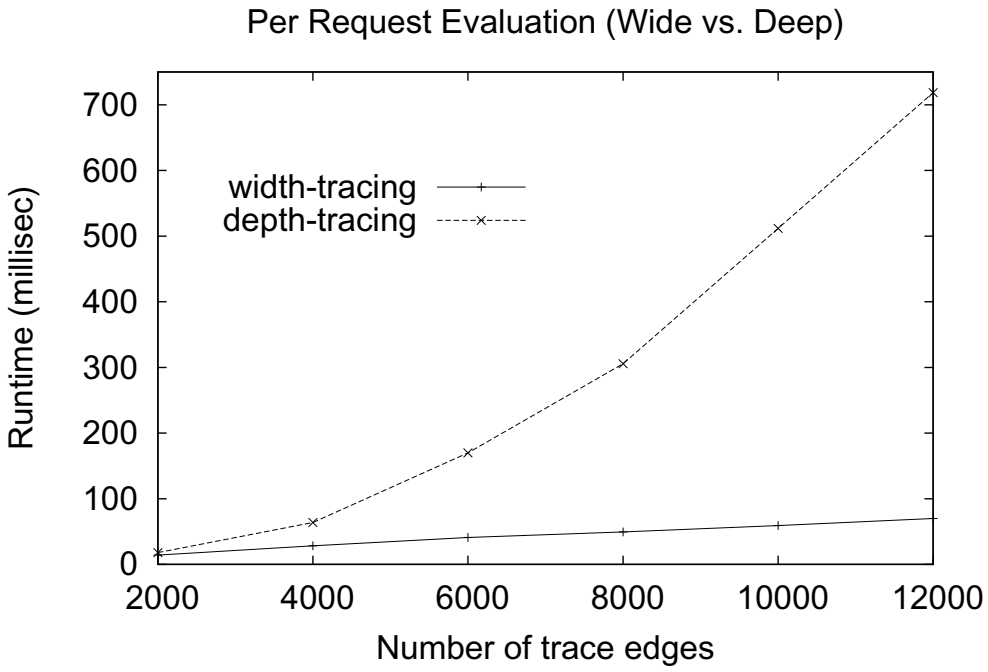

Figure 4.5: Performance of Single PBAC Requests (©2013 IEEE [63])

specification. It it assigned a NAME-ID, in this case "provenance-query-SPARQL", and then incorporated, after which it can be used in policy rule-sets, as shown in Listing A.3.

4.3.3 Experiments and Evaluation

Experiments

For the performance evaluation of the prototype, an experiment was conducted to test and capture the run-time execution of a request instance. The implemented prototype was deployed on a virtual machine instance that is launched with an Ubuntu 12.10 image with 4GB Memory and a 2.5 GHz quad-core CPU. Sample XACML policies and sample XACML requests, which would require the prototype to invoke the PBAC Reasoner components to gather necessary data for the access decisions, were designed. More specifically, it was measured the time it takes to complete the following operations flow:

- The access request is received by the PDP.

- An appropriate access policy is matched.
- Relevant parameters are extracted from the policy and passed to the PBAC Reasoner.
- SPARQL regular path queries are formed by utilizing the DR and QE configuration.
- Queries are executed against PDR and results are returned to the PDP.
- PDP performs evaluation process on the returned results and returns the final decision.

These experiments did not include the enforcement processes as they are application-dependent and do not impact the overall aspects of PBAC evaluations.

For the application domain, a simulation of the HWGS scenario, as described chapter 3, is done. Mock data of possible action transactions, which can occur within the system and captured them using the proposed provenance data model, was generated and then stored as RDF-format triples within an in-memory Jena model. A sample of this mock data is depicted in Listing A.4. Regular-path SPARQL queries were generated and executed against the Jena in-memory model using the ARQ engine. Sample queries are provided in Appendix A.3.

It is recognized that the bottleneck of the prototype lies in the execution of regular path queries. The performance of such query execution depends heavily on the shape of the provenance graph and the pattern after which the traversal needs to be done. Therefore, in generating the mock data, two types of shape a provenance graph takes can be considered. One requires wide and the other requires deep traversal. A wide provenance graph traversal is necessary to query a large amount of actions that all take a single object as input. For example, in the HWGS scenario, one submitted homework object can be reviewed by a multitude of reviewers. A query which searches for all reviewers of an object would have to traverse through all edges branching out from the object itself. A deep provenance graph traversal is necessary to query a path of cause and effect relations between different versions of an object. For example, an uploaded homework object can be replaced multiple times by its owner. To obtain the original version of a homework document, a query needs to traverse back through a large number of edges.

In order to test the flexibility of the framework, the performance of the framework against various quantities of edges that need to be traversed (the width and depth of a provenance graph) was evaluated. The experiment, where the prototype was validated against "extreme", yet reasonable, thresholds, was conducted. In particular, it was evaluated requests that would require incremental number of edges (to be traversed) in the quantities of 2000, 4000, 6000, 8000, 10000 and 12000. To produce precise results, requests of wide type would only require wide edges traversing. The same configuration applied to requests of deep type.

Various experiments were conducted to provide some insights into the scalability of the prototype and approach. Specifically, the throughput of 500 independent requests was evaluated. In other words, the evaluation of one request does not depend on the evaluation result of another request. The throughput results further validate the feasibility of the approach, as will be promptly elaborate after presenting the experimental results next.

Evaluation

The experimental results for single request evaluation are shown in Figure 4.5. The heaviest traversal query case obtains the result of 0.718 second per deep request and 0.069 second per wide request. At the same time, for the lightest traversal query, the result is 0.017 second per deep request and 0.014 second per wide request. The readers should note that for both types of queries, the resulting run-time demonstrate the feasibility of the prototype. While a query that requires a deep traversal shows increased run-time, such phenomenon is most likely because of the SPARQL implementation of query execution that utilizes recursive calls for each successive process step. Regardless, an observation is made that in a practical system deployment, the depth and width traversals of the associated provenance graph do not typically exceed such quantities. Furthermore, in certain special cases, while a provenance graph can grow extremely large in both depth and width, a practical application system and its associated provenance graphs are expected to resemble a large set of disconnected graph components with small depth and width values. Furthermore, it is perceived that in the case of a large provenance graph, various optimization approaches,

such as requests grouping and results caching, can be performed to improve the run-time results. The discussion of these concepts lies beyond the scope of this dissertation and belongs in future work. Nonetheless, although the obtained results are not optimal, they demonstrate the feasibility and potential enhancement of the prototype together with the $PBAC_C$ approach.

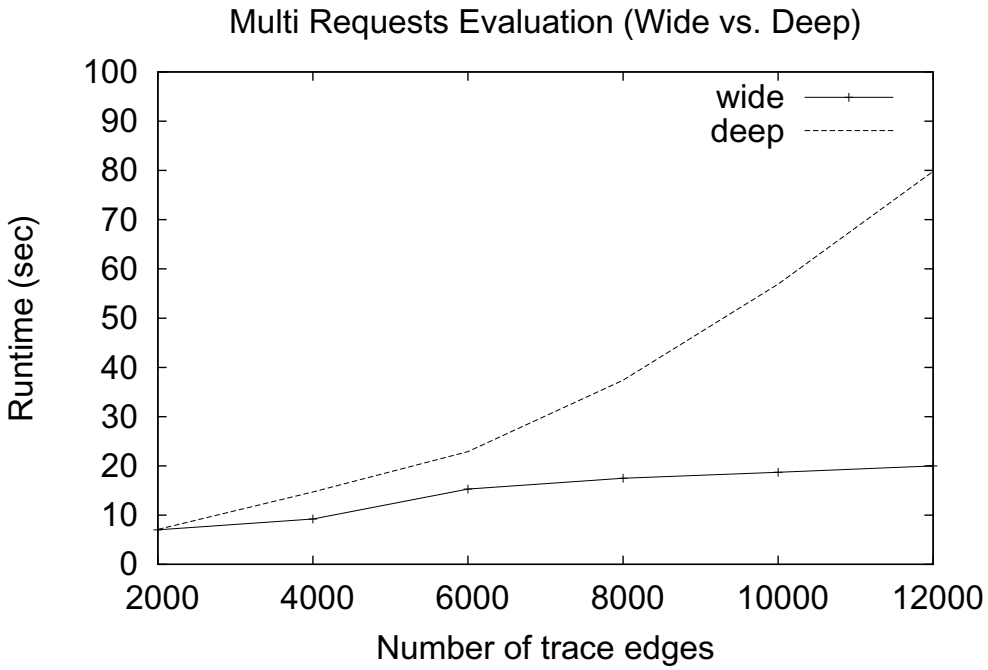

Figure 4.6: Throughput Evaluation per 500 PBAC Requests

For throughput evaluation, the results in Figure 4.6 show that in the width-tracing scenario, the performance overhead increases linearly with the increase in the number of traced edges. In the depth-tracing scenario, the performance overhead increases still linearly, but at a higher slope. In this case, the heaviest tracing query case obtains the result of 500 requests per 80 seconds (0.16 second per deep request) in depth-tracing scenario and 500 requests per 20 seconds (0.04 second per wide request) in width-tracing scenario. Simultaneously, for the lightest tracing query, the result is 500 requests per 7 seconds (0.014 second per deep request) and 500 requests per 7 seconds (0.014 second per wide request). This means that at the lightest tracing scenarios, the performance overhead of both wide-tracing and deep-tracing is very low and hardly differentiable. In fact, according to canonical data structure textbooks, the asymptotic time complexity of traversing a

graph is $O(n+e)$, where n is the number of nodes and e the number of edges in the traversed graph. The experiments consolidate the theoretical analysis. The heavier run-time increase of the depth-tracing query is a result of the SPARQL implementation of query execution that utilizes more recursive calls for each successive process step in depth-tracing scenario than that in width-tracing scenario.

Chapter 5: A PBAC ARCHITECTURE FOR CLOUD IAAS

Acknowledgment: The materials in this chapter are published in [64].

Provenance-based Access Control (PBAC) is an effective access control approach that can utilize readily provided history information of underlying systems to enhance various aspects of access control in a computing environment. The adoption of PBAC capabilities in the authorization engine of a multi-tenant cloud Infrastructure-as-a-Service (IaaS) such as OpenStack can enhance the access control capabilities of cloud systems. Toward this purpose, tenant-awareness is introduced to the $PBAC_C$ model by capturing tenant as contextual information in the attribute provenance data. Built on this model, a cloud service architecture, which provides PBAC authorization service and management, is presented. It is discussed in depth the variations of PBAC authorization deployment architecture within the OpenStack platform and implement a proof-of-concept prototype. Analysis of the initial experimental results and discussion of approaches for potential improvements follow.

5.1 Tenant-aware PBAC in Cloud IaaS

As depicted in Figure 5.1, the primary components of $PBAC_C$ can be briefly described as follows. **Subjects** represent human users interacting with a system. **Actions** represent the type of possible interaction a subject can perform in the system. **Objects** (or **Resources**) represent the type of data entities that exist within a system that require authorization protection for security goals. To interact with a system, a human user, through associated subjects, initiates **Requests** that are evaluated based on **Policies** to determine the access decision (granted or denied). **Provenance Data** contains information on past system events as results of granted access requests and includes two types. [1] *Base* provenance data captures primary component-information of granted and executed access requests while *Attribute* provenance data captures the contextual information associated with the

[1] While provenance data can capture access requests that are not granted, for simplicity, it is assumed that only granted accesses are stored in provenance data.

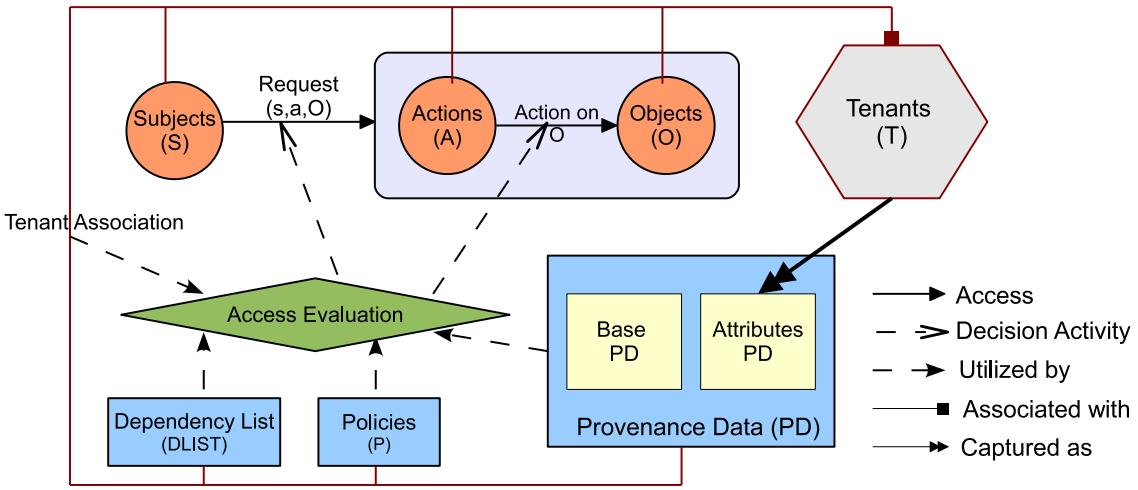

Figure 5.1: A Tenant-aware $PBAC_C$ Model

executed access requests.

In order to adopt tenant-awareness, it can be straightforward to view tenant as a special type of contextual information that can also be captured as attribute provenance data, as modeled in $PBAC_C$. It is then possible to use the relation type "associated with" to capture the semantic relations between tenants and other components such as: *Subjects*, *Actions*, *Objects*, *Dependency Lists*, *Policies*, and *Provenance Data*. Essentially, a set of atomic, or "base", application-specific and defined causality dependency edges between provenance graph vertices can be expressed with regular-expression based patterns. The graph vertices represent model components that constitute the primary entities of a system such as users or resources. This approach allows more expressive capture of relations between the model components. Meaningful combinations of dependency path expressions can be captured with abstract dependency-name constructs which represent more abstract application-specific semantics of the underlying system. An example is a dependency name "wasOriginallyUploadedBy" which captures any combination of dependency path expressions of actions, which can be multiple instances of modify or copy and ultimately upload, on a particular virtual image. Further application of attribute edges on the upload action instance can reveal the cloud user who originally uploaded the virtual image. These constructs can also be used for PBAC policy specifications. When an access request is generated, the *access evaluation* module

extracts the request information to locate the appropriate policies for evaluation. When an access request is granted, the current contextual information is stored as provenance data. This contextual information is uniquely anchored to the action instance of the access transaction in provenance data.

5.2 Provenance-aware Access Control Cloud Architecture

This section discusses a provenance-aware architecture that can enable PBAC capabilities in cloud environment. Specifically, it describes the main components and their interactions, and how the services can be deployed given various design criteria.

5.2.1 Architecture Overview

An overview of the architecture of the dissertation's approach is depicted in Figure 5.2. The three major types of services within this architecture are identified as follows:

Cloud Service (CS) essentially provides a particular IaaS service to client tenants. The types of services include computing (management of virtual resources), authorization, virtual networks, and so on. Examples of the services can be *Amazon Web Services Elastic Compute*, OpenStack *Nova*, etc. These services essentially provide the functionality of the cloud.

Provenance Services (PS) is an IaaS service that is proposed with the purpose of capturing and managing provenance data that can be generated from any other typical cloud service. The provenance data captures the history information of system events occurring within the cloud services and can be utilized for many purposes. This dissertation's focused usage is on PBAC.

PBAC-enabled Authorization Services (PBAS) is an IaaS service that is proposed with the purpose of providing authorization capabilities to all other cloud services that require authorization. The authorization service is capable of providing PBAC features, but at the same time it can also provide other forms of access control including Role-based Access Control, Attribute-based Access Control, etc. The focus of this dissertation is on the provision of PBAC capability for the authorization service.

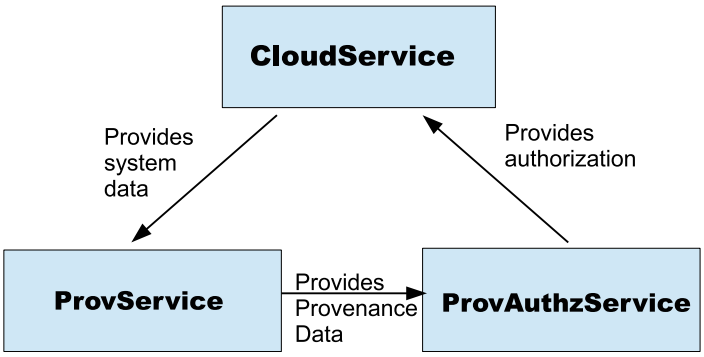

Figure 5.2: A Provenance-aware Cloud Architecture Overview

In summary, the three service types altogether establish an infrastructure that enables PBAC in an IaaS cloud. Specifically, the Cloud Service provides the PS with raw system events that PS selectively stores at provenance storage. The stored provenance data is then used to provide *PBAS* which enhances the security to the Cloud Service.

While this work mainly focuses on the scenario where access requests are granted, the readers should also note that it is possible to capture and store the information relating to access requests being denied. This information can allow additional control capability in a system. For example, if the provenance data of an object reflects that there exist three consecutive instances of request denial for a particular action type within certain recent request interval, it may lock the object from any future access or raise a flag indicating a potential threat or vulnerability within the system and request immediate attention with appropriate countermeasures. This dissertation does not consider denied events as part of provenance data for simplicity and leave it for future study.

5.2.2 Conceptual Architecture

This subsection, as shown in Figures 5.3 and 5.4, identifies and describes the interaction between the logical architectural components of the three service types. These components establish the fundamental and functional aspects of the architecture approach and can be applied to whichever deployment methods that are discussed in the chapter.

Components Any regular cloud service includes:

- Policy Enforcement Point (PEP): is responsible for receiving and enforcing an access request from a user. The enforcement is based on the evaluation results of that access request generated from the authorization service.

- A User is able to generate a request to the cloud service through any forms of interfaces such as web browsers (e.g., OpenStack Dashboard) or command-line interfaces (e.g., OpenStack Nova pythonclient).

The provenance service includes:

- Provenance Data Collector (PDC): is responsible for receiving raw system events data captured from a granted service action request being executed within individual services and potentially performing some filtering to select necessary data only.

- Provenance Data Manager (PDM): is responsible for transforming the collected raw data received from the PDC into provenance graph data format as well as managing the resulting provenance data. The management responsibilities include storing and loading provenance data in and from a database, as well as forming and executing provenance graph queries and formatting and returning query results thereafter. A sample query is included in Appendix A.3.

- Database (DB): represents persistent storage.

The PBAC-enabled authorization service includes:

- Policy Administration Point (PAP): is responsible for managing access control policies by enabling policies specification, storage and retrieval.

- Policy Information Point (PIP): is responsible for looking up relevant information that is necessary for making an access decision. In regard to PBAC, the PIP is tasked with delivering responses to provenance data requests to the relevant provenance service.

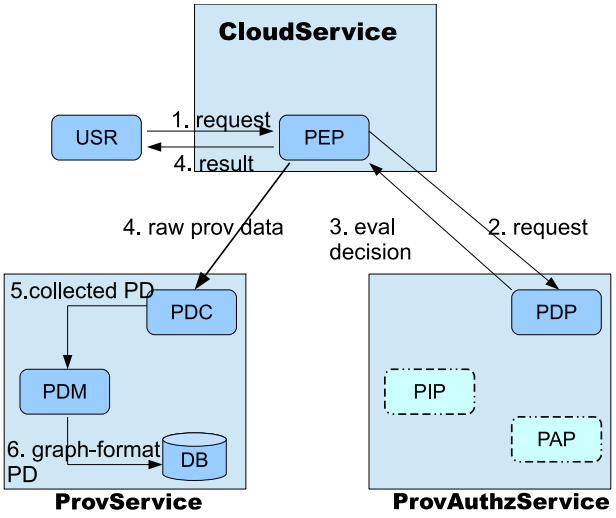

Figure 5.3: A Provenance Service for Cloud IaaS

- Policy Decision Point (PDP): serves as the main computing process in deciding how a request should be resolved. In particular, the PDP receives the requests from the PEP, looks up the policy from the PAP, and requests information from the PIP to make decisions, which are then returned to the PEP.

Interactions Next, it will be described how the services perform whenever an access request comes in, as illustrated in Figure 5.3 and Figure 5.4. When a request is initiated by a user through any user client interface, the PEP receives the request and proceeds to verify the request with the authorization service by forwarding the request with relevant content to the PDP that resides in the PBAS service. In Figure 5.3, this interaction is abstracted in steps 2 and 3. It is further elaborated in Figure 5.4 through steps 2-11. Figure 5.3 demonstrates what happens after the access request evaluation process is completed. Essentially, if a request is granted, the PEP enforces the execution of the requested action. The corresponding system event is then captured and sent to the PDC component of the provenance service where certain filtering can be performed to remove unnecessary data. The filtered data is then passed to the PDM for formatting into appropriate provenance data graphs for storage for later use. This completes a functional cycle in the context of an access request being directed at a cloud service.

In Figure 5.4, upon receiving the request from the PEP (2), the PDP proceeds to perform the

evaluation procedure which includes, in sequential order, retrieving the correct policies through the PAP (3,4), searching for information required for policy rules evaluation through the PIP (5,6,9,10), and computing the actual evaluation decisions and returning the final results back to the PEP for enforcement (11). In this architecture, the PIP is responsible for looking up relevant provenance information in the provenance service for carrying out PBAC-related policy rules. The PIP performs this task by communicating with the PDM component of the provenance service by sending query templates. The PDM loads provenance data from its storage, forms appropriate queries and executes them to extract necessary provenance information to return to the PIP.

The above process can be demonstrated with the following example. Suppose a user Alice requests to delete a particular virtual machine, "vm1", in a tenant. The policy states that only a user who creates and stops a virtual machine instance can delete it. The PEP receives the request from Alice and delivers necessary information to the PDP. The PDP parses the request information, matches the request to the correct policy through the PAP to extract appropriate rules for the action "delete". The PIP then sends information including "vm1" and dependency path patterns, e.g. "wasVMCreatedBy", that express the semantics of creating and stopping users of a virtual machine instance to the PDM. The PDM forms appropriate queries using the provided information, executes the queries and returns the results back to the PDP. As Alice is shown to be the user who created and stopped "vm1", the PDP sends the approval to the PEP which starts the enforcement of the action. Upon completion, the PEP sends the events information to the PDC. The PDM can then at least generate provenance data which captures the event where Alice performed "delete" on "vm1".

5.2.3 Deployment Architecture in OpenStack systems

Given the above logical architecture discussion, the focus is shifted to how the services can be deployed in a cloud IaaS OpenStack system.

Most extant OpenStack cloud services often embed their own authorization service compo-

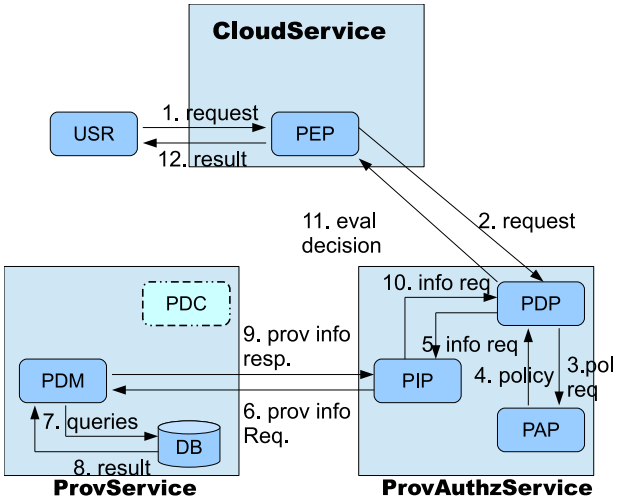

Figure 5.4: A PBAC-enabled Authorization Service for Cloud IaaS

nents that can enable authorization mechanisms including RBAC and ABAC. [2] In order to enable PBAC authorization mechanism for the extant OpenStack services, it is essential to identify several deployment architectures based on where PS and PBAS are implemented, which presents their own strengths and weaknesses.

First, similar to the current deployment of authorization components, these services can be integrated as structural components of an extant cloud service. In a cloud environment where sharing provenance data in multiple services is not a necessity, this integrated services deployment can significantly reduce the communication decision latency that takes place. Current standalone services, such as Nova and Glance, communicate over HTTP REST interface that can introduce expensive latency. Communication between components within the same service, as either inter-process or intra-process, is much less expensive in comparison. However, the extant cloud service may have more computing load to deal with as it is required to maintain its own PBAS and PS components. Essentially, the integrated service has to collect, store and manage its own provenance data. This can also reduce the ease of services integration as it becomes more difficult to update changes to any of the embedded services.

Furthermore, in a cloud environment where cross-service provenance data sharing is necessary

[2]The Swift component utilizes a different form of authorization than most other OpenStack services.

for purposes such as PBAC, a deployment of integrated PBAC-enabling services is required to employ provenance data sharing mechanisms. As each service stores and manages its own provenance data, PBAC decisions of a service require the provenance data of a different service. The requiring service has to initiate a request to the different service, therefore introduces communication decision latency over HTTP channels. In order to mitigate decision latency, each service can take the approach of maintaining duplicate provenance data of all relevant services. However, this introduces the necessity to synchronize all provenance data storage, and results in synchronization latency over HTTP channels. In scenarios where immediate synchronization is vital to the correctness of a PBAC decision, synchronization latency can affect decision runtime even if the evaluation process is done locally to the extant service. In scenarios where periodic synchronization is acceptable, optimal decision latency can be achieved.

Individual cloud service management of locally maintained provenance data can be complicated. The complications can be alleviated with the standalone deployment method with the cost of communication latency. Essentially, a standalone provenance service enables central provenance data storage and management, which facilitates duplication and synchronization.

In addition, it is possible to employ several varieties of these two deployment methods, which can be termed hybrid deployment, to alleviate some of the issues faced by the above two deployment approach. For example, since not all provenance data is required for PBAC uses, only PBAC-relevant provenance data is necessarily duplicated in individual regular cloud services. Other provenance data, which can be used for auditing, can be stored and managed by standalone provenance service. This dissertation uses the standalone architecture for the OpenStack implementation and experiments.

5.3 An OpenStack Implementation

This section emphasizes the application of the approach on the open-source cloud management platform of OpenStack.

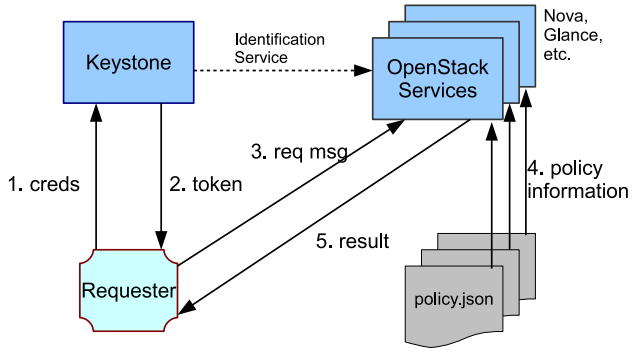

Figure 5.5: An Overview of OpenStack Authorization

5.3.1 Overview of OpenStack Authorization Architecture

At the IaaS layer, OpenStack comprises several components that provide services to enable a fully functional cloud platform. Each of these components controls access to specific resources through locally maintained JSON policy files. At the IaaS layer, the resources to be protected are composed of API functions and virtual resources such as virtual machine images and instances.

Figure 5.5 captures a simplified view of the authorization as similarly performed by most OpenStack components. While the solid arrows denote information flow, the fine-dashed arrow indicates the Keystone component is responsible for providing identity service to other OpenStack components. This also provides authorization-required information indirectly using a token.

1. When an access request is made, authentication credentials need to be submitted to Keystone for validation.

2. Once validated, Keystone returns a token which contains necessary authorization information such as roles.

3. The token is then included in the request that is sent to the specific service component.

4. Authorization information is extracted from the token and used in evaluating the rules specified in the policy file native to that service.

5. The final evaluation and/or enforcement result is then returned to the requester.

Policy rules can be specified as individual rules of each criteria or a combination of rules. For Grizzly release, OpenStack authorization engine supports two types of rules: RBAC [77] where decisions are based on role field, and ABAC [46] where decisions are either based on the value of a specific field or the comparison of multiple fields' values. A sample PBAC policy for Nova is depicted in Listing A.6 and for Glance is depicted in Listing A.7.

The authorization engine of OpenStack is evolving as additional blue-prints and feature proposals are raised and delivered by the open-source community on a daily basis. Currently, OpenStack does not possess or support any variations of PBAC in its authorization schemes. As the demonstration and discussion of PBAC's usefulness in a multi-tenant cloud IaaS exhibit, it is useful to incorporate PBAC mechanisms into the OpenStack authorization platform for history-based, dynamic and finer-grained access controls.

5.3.2 Prototype, Experiment and Evaluation

This subsection describes and discusses the implementation of a proof-of-concept prototype that realizes the above proposed architecture for enabling a PBAC-enabled authorization service within the OpenStack platform. Specifically, this subsection demonstrates how the OpenStack Computing (Nova) service can utilize the PBAC-enabled authorization for making access control decisions in addition to the current authorization schemes Nova is employing. A similar process can be applied on the other services in OpenStack. This dissertation implements the solution for Nova and Glance components and evaluates the performance runtime for each component under different experiments. Afterward, an analysis of the runtime results is provided.

OpenStack Nova Architecture

First, this subsection describes the current implementation details of the OpenStack Nova architecture. As depicted in Figure 5.6, the Nova components include:[3]

1. *Web Dashboard* is the potential external component that talks to the API, which is the com-

[3]This is a partial list of Nova components.

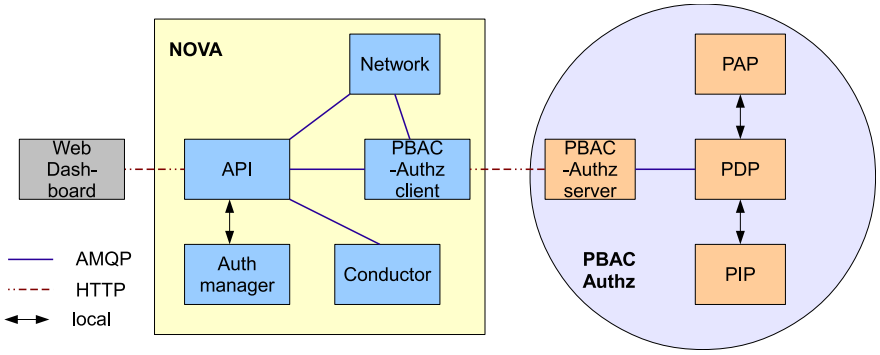

Figure 5.6: Nova Implementation Architecture with PBAS Service

ponent that receives HTTP requests, converts commands and communicates with other components via the queue or HTTP.

2. *Auth Manager* is a Python class that is responsible for users, projects and roles. It is used by most components in the system for authentication purposes.

3. *Network* is responsible for the virtual networking resources.

4. *Conductor* is responsible for manage database operations.

There are two methods of communication between the service components: intra-service communication is done via AMQP mechanisms while inter-service communication is done via *HTTP REST* mechanisms. These are represented in Figure 5.6 as continuous lines and dashed lines respectively. Communication between sub-components of the same service can be done locally, such as invocation of the *Auth Manager*.

OpenStack Glance Architecture

This subsection provides a similar description about the current implementation details of the OpenStack Glance architecture. As depicted in Figure 5.7, the Glance components include:[4]

1. Swift-api: Similar to the Nova API component, the Swift-api is responsible for receiving HTTP requests from external Web Dashboard, converting commands and communicating

[4]This is a partial list of Glance components.

with other components via the queue or HTTP.

2. Glance Registry: stores and manages metadata about images.

The communication between the service components is similar to the Nova communication.

PBAC-enabled authorization implementation

In order to incorporate and enforce PBAC-enabled authorization service, the following components were implemented to extend Nova. Similar approach can be applied to Glance.

- *PBAC Authorization Client*: a Python class that implements an authorization method that can be invoked whenever an API/ Scheduler/ Network/ Compute method is invoked. The client sends HTTP requests to the PBAC authorization server.
- *PBAC Authorization Server*: a Python class that resides on the PBAC-enabled authorization service. Receives HTTP requests from the PBAC authorization client, forwards the requests to and receives the decisions from the PDP, and returns the decisions to the PBAC authorization client.
- PDP, PIP, and PAP Python implementations that correspond to the associated architecture components.

Since all access control policies are specified in JSON, a policy parser class, which can interpret policy statements specifying PBAC rules in JSON, was implemented. A sample PBAC policy is depicted in Listing A.6 (Nova policy) and in Listing A.7 (Glance policy).

Experiments

In order to evaluate the proof-of-concept prototype, the provenance service and the PBAC-enabled authorization service were created and deployed on a Devstack installation of the OpenStack platform. [5] The Devstack is under the OpenStack Grizzly release and is deployed on a virtual machine

[5]Note that this dissertation does not provide experiments for measuring provenance data update processes after granted action request enforced. Having such results can further enrich the insights and belongs in future line of work.

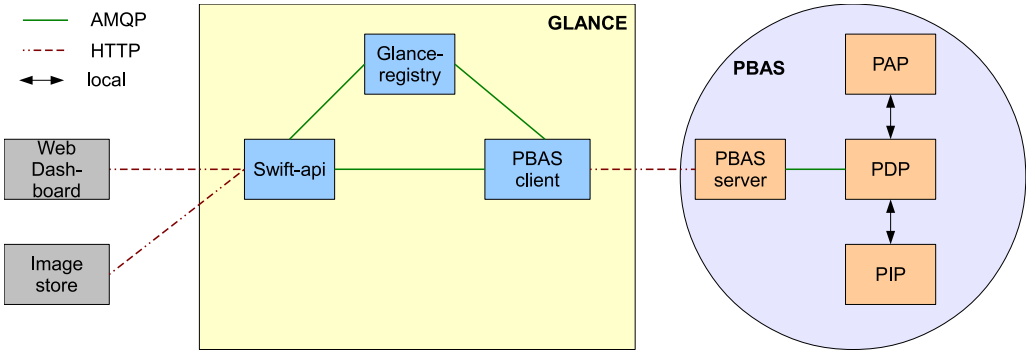

Figure 5.7: A PBAC-enabled Authorization Service for Glance Architecture

that has 4GB of memory and runs on Ubuntu 12.04 OS installation. In addition, provenance data was generated to simulate life cycles of VM images and instances across tenants and stored in Resource Description Format (RDF) [51] with the Python RDFlib library [7]. Measurements were conducted on the execution performance of the following experiments.

Experiment 1 (e1): The execution time of a Glance command and a Nova command that require checking the associated policy for RBAC requirements (the original DevStack system).

Experiment 2 (e2): The execution time of a Glance command and a Nova command that require checking PBAC policy rules in addition to regular RBAC policy rules. In this experiment, the PS and PBAS are deployed as integrated components of the Nova and Glance services.

Experiment 3 (e3): The execution time of the commands with the presence of an authorization service which evaluates the RBAC policy the service maintains and additionally PBAC policy where the authorization service also manage provenance service operations.

Experiment 4 (e4): The execution time of the commands with the independent presence of both a provenance service and a PBAC-enabled authorization service. The PBAC-enabled authorization service performs both normal RBAC requirements as well as PBAC requirements, where

Table 5.1: Evaluation of Glance Experiments (secs)

Traversal Distance	Glance(e1)	Glance(e2)	Glance(e3)	Glance (e4)
No PBAC	0.55	-	-	-
20 Edges	-	0.575	0.607	0.642
1000 Edges	-	0.612	0.788	0.852

necessary provenance information is obtained from the provenance service.

For each experiment and each command, 10 runs were performed and the average run-time was taken. As noted in [63], the size and shape of the underlying provenance graph pose significant impact on query run-time. This dissertation evaluated PBAC queries that require depth traversal. Specifically, the experiments were conducted to evaluate the two scenarios where graph traversal takes distances of 20 and 1000 edges. These edge parameters were selected to respectively reflect a normal and an extreme use case of a VM image and instance within a cloud IaaS environment. The mock provenance data captures a VM image that is uploaded and modified multiple times and used to create a VM instance, which is suspended, resumed, and taken as a snapshot by multiple cloud users. A sample of this mock provenance data is depicted in Listing A.5. The policy is specified following the informal grammar provided in Chapter 4.1.2. A sample policy rule can specify that a user is only allowed to resume a VM instance if and only if he suspended that instance or a user is only allowed take a snapshot of a VM instance if he uploaded the VM image that instance is created from. A JSON equivalent of this policy is depicted in Listing A.8. The performance results are given in Table 5.1 and Table 5.2.

Evaluation and Discussion

Based on the experimental results, the following observations are made. First, compared to the regular execution (**e1** approach) of Glance or Nova commands, the incorporation of PBAC services (either **e2**, **e3** or **e4** approaches) introduces some overhead for traversal distance of 20 edges, specifically between the 10 to 40 percent range. It is also observed that the deployment of separate PBAS and Provenance services also introduces some overhead, specifically between the 5 and 18 percent range, as a result of the additional communication time between provenance service and

Table 5.2: Evaluation of Nova Experiments (secs)

Traversal Distance	Nova(e1)	Nova(e2)	Nova(e3)	Nova(e4)
No PBAC	0.75	-	-	-
20 Edges	-	0.84	0.902	1.062
1000 Edges	-	2.292	3.620	4.102

PBAC service. It is observed that for the case of 1000 edges distance, the additional overhead is as expected as depth traversals require recursive implementation. However, the overhead is much more expensive for the Nova command in comparison to the Glance command. The reason for this lies in the authorization implementation of the Nova command. Specifically, the execution of the Nova command generates several authorization calls in contrast to only one from the Glance command. As the number of edges increases, this additional cost increases exponentially.

This subsection identifies two potential approaches to improve on these performance results. Firstly, it is possible to reduce the performance cost associated with the increase in traversed edges by using meaningful, abstract edges that can equivalently capture the semantics of many base edges. This can help reduce the number of edges and thus produce, for example, a 20 edges run-time for a 1000 edges case. Secondly, in cases where numerous authorization calls are required such as the case of the Nova command, it is possible to employ some caching mechanism to store the result of the first authorization call and thus reduce the frequency of needed authorization calls. This can potentially reduce the run-time of the Nova command to a similar run-time of the Glance command. Overall, this approach produces feasible results for typical cases, and for extreme cases some optimization approaches can achieve acceptable results.

Chapter 6: PROVENANCE DATA SHARING FOR PBAC IN MULTI-ORGANIZATION

Acknowledgment: The materials in this chapter are published in [62, 66].

In previous chapters, a framework of PBAC models, architecture and implementations were presented for enhanced access control approaches in single systems and multi-tenant single-cloud systems. PBAC approaches can also be applied in distributed systems or multi-cloud systems and enable similar security benefits in the underlying systems. However, for PBAC to be deployed in such systems, the concerns on provenance data sharing need to be addressed. This chapter provides the preliminary works that have been done toward this goal and presents the concepts through a group-centric collaboration environment involving multiple independent organizations.

6.1 Group-centric Secure Collaboration

Collaboration comes in many different forms and sizes. To facilitate scenarios where a well-defined collaboration group exists, the concept of a Group-Centric sharing framework was recently introduced [52,53]. In this inter-organizational collaboration framework, the participating organizations collaborate through an agreed structure defined as a group. In a collaboration group, organizations share resources, which are termed objects. A version control system is applied on these shared

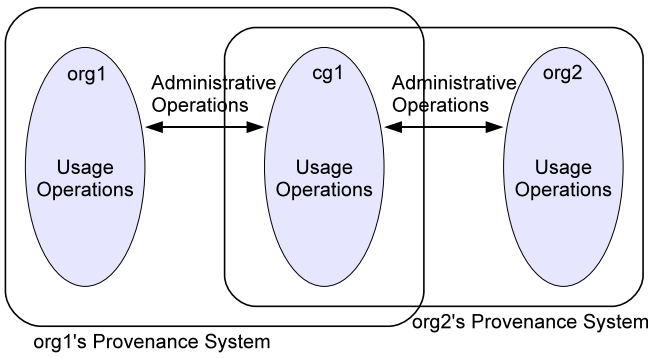

Figure 6.1: A Conceptual View of Provenance Systems in A Group-centric Collaboration Environment (©2011 IEEE [66])

objects. Users, who are granted access, can perform collaborative work on these objects. Organizations can create as many collaboration groups as necessary.

In [52], the administrative and operational aspects of the framework are addressed separately with two component sub-models. The models are specified following the attribute-based UCON model for usage control. The administrative sub-model is responsible for the management of groups as well as users and objects in the collaboration groups. The set of corresponding operations include: Establish/Disband for managing the group, Join/Disband/Substitute for managing users/admins, and Add/Remove/Export/Import/Merge for managing objects that are shared or locally created within the groups. The usage or operational sub-model, in contrast, is concerned with the management of users' activities within the collaboration groups as well as the respective organizations. The set of operations corresponding to these group-centric entities include: CreateRO/CreateRW/Kill for data flow control, Read/Update/Create for usage of objects/versions, and Suspend/Resume for controlling usage of objects/versions.

6.1.1 A Group Collaboration Environment for Data Provenance

In group-centric collaboration, in general there could be multiple organizations and these organizations could establish multiple collaboration groups for different purposes. For simplicity, it is assumed that two participating organizations $org1$ and $org2$ have established one collaboration group $cg1$. As identified in [52], there are two types of operations. Administrative operations are performed to establish/disband groups together with group administrators, substitute group administrators, join/leave group members in a group, add/remove organization data to and from a group, etc. This means provenance data includes operations not only on shared data but also on groups and users. Usage operations are performed against data objects accessed via either an organization or a collaboration group. Also, there are two types of data objects in a collaboration group. Firstly there are pre-existing data objects shared by organizations in the collaboration group, and secondly there are data objects that are natively created in the collaboration group.

It is assumed, as shown in Figure 6.1, that conceptually each organization facilitates its own

provenance system which captures provenance data for usage operations against data objects managed within the organization and shared by the organization in collaboration group as well as data objects that are natively created in the group. The provenance system also captures provenance data for administrative operations against the collaboration group, group members and data objects in the organization and in the collaboration group.[1] If mutually agreed, the participating organizations can query the other organization's provenance data using the overlapping provenance data as connectors.

6.1.2 Data Object Versioning Model

This dissertation uses the terms objects, versions and copies. It is assumed that one object can have multiple versions, and each version can have multiple identical copies. The versions of an object form a rooted tree structure, relating a parent version to its immediate children versions. For provenance purposes each copy (identical in content) is considered as a separate "object."

6.2 Provenance Data for Group-centric Secure Collaboration

In order to discuss utility of provenance data, it is necessary to identify operations that can be performed on data objects and dependency of the data objects that are formed as a result of an operation or a set of operations. This dissertation utilizes OPM notations to show these operations and data object dependencies.

As mentioned earlier, [52] identified various administrative operations on groups, administrators, regular users, and data objects as well as regular users' usage operations on data objects in a group. It is not necessary to capture all these operations in provenance data. Many of these

[1] There can be several different ways to structure the overall provenance system in a group collaboration environment. For example, it is possible that $org2$ is only allowed to capture provenance data for its own user's operations or operations on their local data objects while $org1$ captures as discussed above. This could make sense, for example, in case a government organization collaborates with a contracting company where the contracting company's access to provenance data is restricted by the government. Another example could be that each participating organization and shared group maintains its own provenance system. In this case, the provenance data captured and maintained by collaboration group may need to be accessible by the participating organizations even after the collaboration group is disbanded.

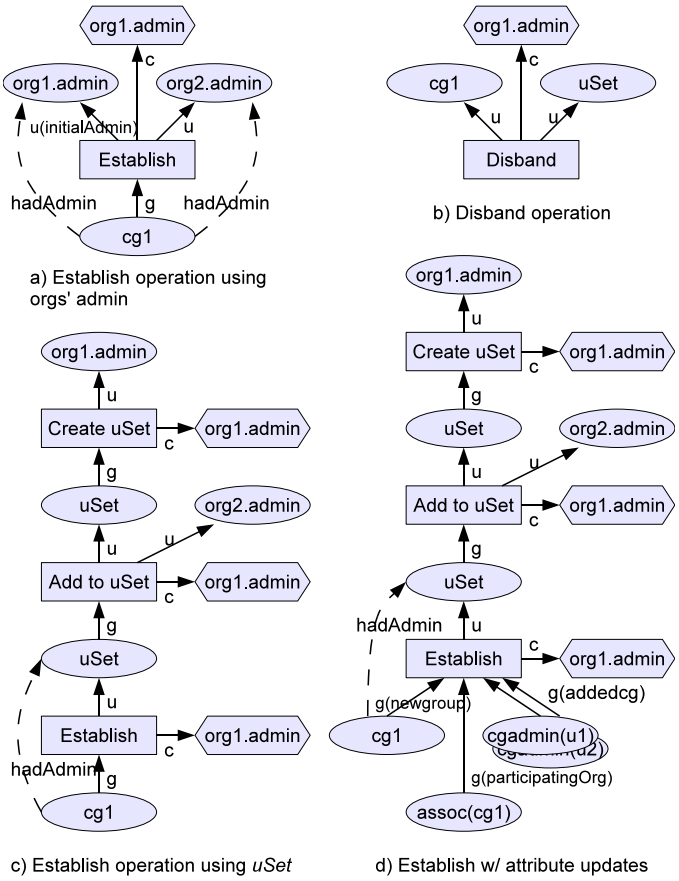

Figure 6.2: OPM Diagrams for Establish/Disband Operations (©2011 IEEE [66])

operations are for authorization purpose and are not meaningful for provenance. Note that how much information of an operation can be or should be captured in provenance data depends on the participating organizations' agreement and provenance system design details. Hence, there could be many variations of the general theme of this section. Also, in the system of Figure 6.1 the provenance data captured in $org1$ could be different from those captured in $org2$ for the same operation. Further, there could be other operations (e.g., object duplication and deletion) or the existing operations could be refined to capture richer semantics (e.g., update operation can integrate some content of another data object into the updating object). Here the main focus is on the operations identified in [52].

6.2.1 Provenance Data of Administrative Operations

This subsection discusses how administrative operations identified in [53] can be expressed in OPM.

Establish($uSet, cg$): Establish collaboration group cg. In general, collaboration group is established together with a set of administrative users who represent their own organizations. While there could be multiple ways to do this depending on how participating organizations agree, it is assumed that one of these administrative users establishes collaboration group on behalf of other users.[2] Figure 6.2a) illustrates that the $establish$ process "$wasControlledBy$" (shown as an arrow labeled "c") $org1.admin$ and "$used$" (shown as arrows labeled "u") $org1.admin$ and $org2.admin$. The artifact $cg1$ "$wasGeneratedBy$" (shown as an arrow labeled "g") the $establish$ process. In other words, an administrator of an organization $org1.admin$ established collaboration group $cg1$ together with two group administrators $org1.admin$ and $org2.admin$. Here, there is a subtype of $wasDerivedFrom$ (shown in dashed arrows) named as $hadAdmin$ to show more meaningful dependency of provenance data artifacts. The provenance data of the $establish$ operation can be also captured in a way discussed in [52]. This is shown in Figure 6.2c). Here, $org1.admin$ created a $uSet$, added a set of administrative users to the $uSet$ and then used it to establish collaboration group $cg1$. In addition, [52] discussed that, as shown in Figure 6.2d), firstly $assoc$ attribute of cg1 was created/updated to include the participating organizations (all organizations found in $uSet$) and secondly participating users' $cgadmin$ attributes were updated to include $cg1$ as part of the groups they administer.

While these additional updates on related attributes are discussed in [52], these activities may not need to be captured in provenance data. This is because capturing the "establish" operation as shown Figure 6.2a) might be enough to provide sufficient provenance utility. Capturing additional details of creation/update activities on attributes may not provide any additional significantt prove-

[2]This dissertation does not attempt to identify an exhaustive list of the possible scenarios for establishing a collaboration group. Rather the dissertation shows a couple of possible ways how provenance data for collaboration group establishment can be expressed using OPM and further discuss the captured provenance data. This also applies to the other operations discussed here.

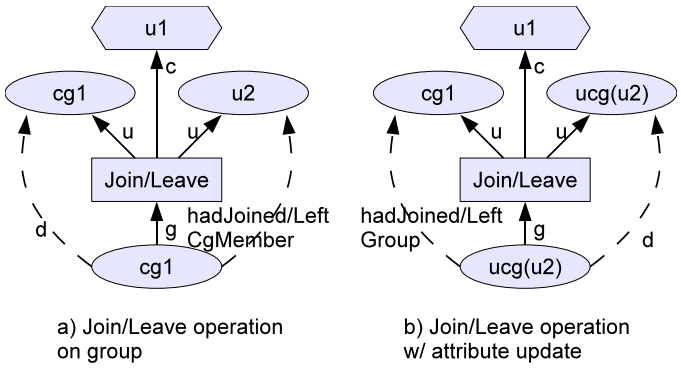

Figure 6.3: OPM Diagrams for Join/Leave Operations (©2011 IEEE [66])

nance utility. Specifically, Figure 6.2a) is essentially enough to identify who created the group or who were the participating organizations or administrative users of the group. Capturing how $uSet$, $accoc$ or $cgadmin$ attributes were created/updated can be useful only if it is required to verify some specific aspects such as who added a certain user in $uSet$, which administrative user is added first, etc. At the same time, attribute updates shown in Figure 6.2c) and d) are just one way of conducting the details of the operation and can be subsumed in the approach shown in Figure 6.2a).

Disband$(uSet, cg)$: Disband group. The provenance data of this operation allows users to query who disbanded the collaboration group. While [52] requires agreement of all administrative users for this operation, provenance data only captures who conducted the operation and does not reflect the authorization processes. Figure 6.2b) shows an administrative user $org1.admin$ who disbanded collaboration group $cg1$ and a set of administrative users $uSet$. Capturing provenance data of the $establish$ and $disband$ operations allows users' to query pedigree and disposition of the collaboration group. This also means that the group is considered an OPM artifact.

Join/Leave$(u1, u2, cg)$: Join/Leave user to/from group.[3] Suppose an administrative user $u1$ from a participating organization included a user $u2$ as a member of collaboration group $cg1$. The provenance data of this operation can be expressed in OPM as shown in Figure 6.3a). Here,

[3]Although $join$ and $leave$ operations are shown in a single process in Figure 6.3 for convenience, they are two separate operations and occurrence of each operation should be captured by a separate process.

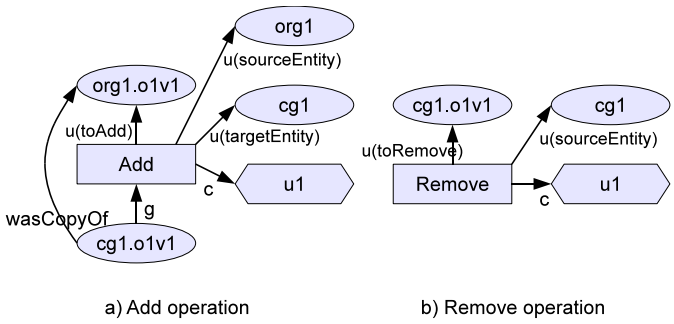

Figure 6.4: OPM Diagrams for Add/Remove Operations (©2011 IEEE [66])

"$join/leave$" operation processes were controlled by $u1$ and used $u2$ and $cg1$, and a new $cg1$ was generated from the "$join/leave$" processes. In [52], a necessary update activity on the attribute ucg of $u2$ is captured to reflect that the user is now a member of $cg1$ (see Figure 6.3b)). However, as similarly discussed in the $establish$ operation above, this update can be seen as one of multiple ways of performing the "$join/leave$" operations. For example, instead of using the ucg attribute of a user, the ucg' attribute of $cg1$ can be used to capture all the group members. Therefore, Figure 6.3b) can be subsumed in a more general operation description shown in Figure 6.3a). In the Figure 6.3a), two subtypes of "$wasDerivedFrom$" named "$hadJoinedCgMember$" and "$hadLeftCgMember$" were introduced to capture the dependencies of provenance data artifacts.

Add$(u, o, v, org, cg,)$: Add object version from org to group. The add operation creates a copy of an object version from an organization to a collaboration group. In Figure 6.4a), $u1$ added a copy of an object $org1.o1v1$ from an organization $org1$ to a group $cg1$. A subtype of "$wasDerivedFrom$" named "$wasCopyOf$" is identified to show a node dependency. Here, $org1$ is used as a source entity and $cg1$ is used as a target entity. While both source and target entities are captured here, if this provenance data is captured by $org1$, the source entity information may not need to be captured since it is always $org1$. However, if this provenance data is captured by organizations other than the source entity, say $org2$, the provenance data in $org2$ needs to include both the source and target entities information. While the source organization information could be found in source data object, this dissertation does not assume that this is always the case. Hence

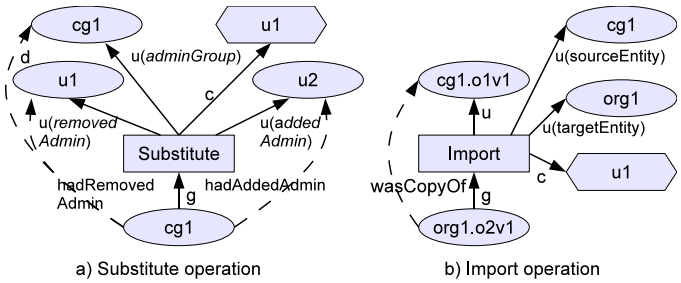

Figure 6.5: OPM Diagrams for Substitute/Import Operations (©2011 IEEE [66])

the source entity information is shown explicitly in the OPM diagram.

Remove(u, o, v, cg): Remove object version from group. The $remove$ operation deletes a copy of an object version from the entity where it is located. In Figure 6.4b), $u1$ removed $cg1.o1v1$ from $cg1$.

Substitute$(u1, u2, cg)$: Substitute group admin. The $substitute$ operation removes an existing administrative user and add another administrative user in a collaboration group. In Figure 6.5a), $u1$ substituted herself with $u2$ as an administrative user in $cg1$. The roles of these $used$ edges are captured in $u(role)$ format. In Figure 6.5a), $cg1, u1$ and $u2$ are used with roles $adminGroup$, $removedAdmin$ and $addedAdmin$, respectively. Two subtypes of "$wasDerivedFrom$" named "$hadRemovedAdmin$" and "$hadAddedAdmin$" are identified to show the node dependencies. The OPM diagram also shows a generic "$wasDerivedFrom$" arrow to capture dependency between the previous and current state of $cg1$.

Import$(u, o1, v1, o2, cg, org)$: Import a version from group to organization. The $import$ operation copies a version of an object that was natively created in collaboration group into an organization. In Figure 6.5b), an object version $cg1.o1v1$ was copied from $cg1$ to an organization $org1$ and named as $org1.o2v1$. While $org1.o2v1$ is an exact copy of $cg1.o1v1$, $org1.o2v1$ is treated as a new object. The "$wasCopyOf$" shown in Figure 6.5b) shows the dependency of the two data objects. While these two copies are considered different objects and cannot be connected in a version control system, using the dependency arrow $wasCopyOf$, users in collaboration group or in an organization can trace the usage information of a particular object version that are imported

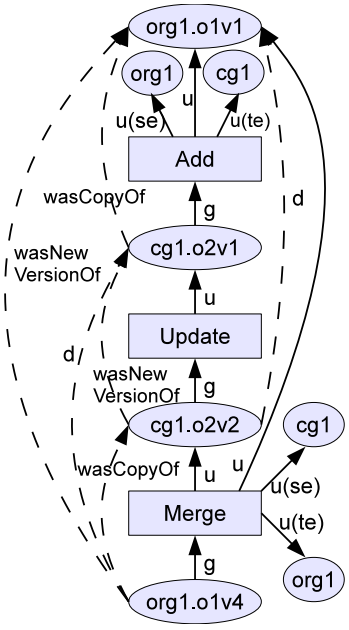

Figure 6.6: OPM Diagrams for Merge Operation (©2011 IEEE [66])

to another organization even if the user does not belong to the organization. While [52] discusses "$export$" operation to capture the fact that all the administrative users should agree to make an object version exportable, this operation is identified for authorization purpose, hence not included in this dissertation.

Similar to the add operation, provenance data for the $import$ operation may or may not include the source entity information depending on whether the organization of the provenance system is the one who performed the operation or not.

Merge$(uSet, cg, o, v, org)$: Merge a version from group to organization. The $merge$ operation creates a newer object version of an existing object in an organization. This new version created in the organization is a copy of an object version that is created in collaboration group as a result of the $update$ operation on a version of the object that is previously added from the organization to the group. (Additional details on the $update$ operation are discussed in the next subsection.)

The $merge$ operation needs some precedent operations that should have occurred in advance. At least one add operation and then one $update$ operation on the added version are necessary to perform a $merge$ operation on the updated version. In Figure 6.6, $org1.o1v1$ was added to $cg1$ then

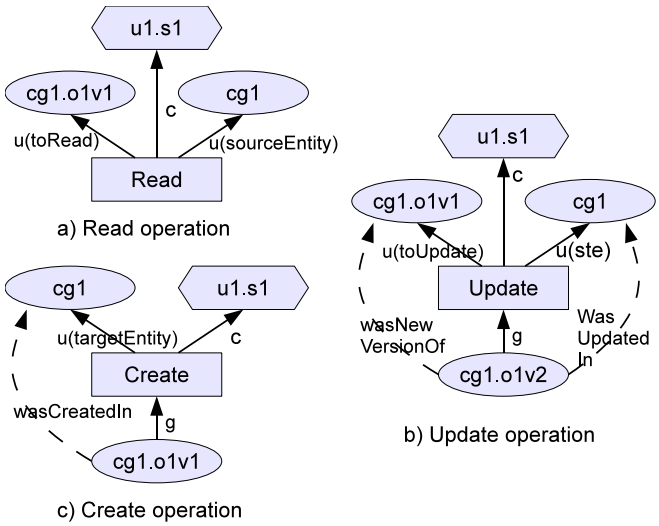

Figure 6.7: OPM Diagrams for Read/Update/Create Operations (©2011 IEEE [66])

the added version $cg1.o2v1$ was updated in $cg1.o2v2$ which then is merged back into the original organization $org1$ as a new version of $org1.o1v1$ shown as $org1.o1v4$. Here, the new version $org1.o1v4$ is an exact same copy of $cg1.o2v2$.[4] Figure 6.6 shows that in the $merge$ operation, $cg1$ was used as a source entity (shown as $u(se)$) and $org1$ was used as a target entity (shown as $u(te)$). Two subtypes of "$wasDerivedFrom$" named "$wasCopyOf$" and "$wasNewVersionOf$" are used to show the dependencies of object artifacts. Having the dependency of data objects allows users to trace information flow and usage history on the various versions of a particular object as well as copies of the versions. For simplicity, agent nodes for add, $update$ and $merge$ operation processes are omitted in Figure 6.6 though every process needs an agent.

6.2.2 Provenance Data of User's Usage Operations

This section discusses provenance data of user's usage operations identified in [52]. [52] assumes that a user represents a human who creates a subject in a system and a subject exercises operations

[4]Note that the $merge$ operation creates an exact copy of an object version in the collaboration group into an existing version tree in the organization where the original version in the collaboration group is $added$ from. This is different from merging two versions found in a same version tree of an object within an organization. While the latter could be useful, this dissertation does not consider this kind of "content merge" operation. For example, if $org1.o1v1$ was updated within $org1$ after it was added to $cg1$, $org1.o1v4$ is still a new version of $org1.o1v1$ but not a new version of the updated version of $org1.o1v1$.

on behalf of the user. While user-subject distinction is critical for information flow control in group collaboration setting, provenance data only capture operation events that already occurred in a system and do not worry about how the operations are authorized. Therefore, as shown in Figure 6.7, subjects are used as agents who controlled operations. This is not necessarily critical in this dissertation.

Read$(s, o, v, entity)$: Read an object version. The $read$ operation occurred on an object version by a subject in an entity. Entity information is captured if an object version is read in collaboration group since there could be multiple groups in a single provenance system. Provenance data of $read$ operation against an imported or merged object in an organization is not likely to be captured by a provenance system of another organization. However it may need to be traced by another organization since the data object may have been used or updated earlier by the tracing organization. For this, provenance data needs to include source entity information. This applies to both $update$ and $create$ operations discussed below.

Update$(s, o, v, entity)$: Update an object version. Similar to the $read$ operation, the $update$ operation occurred on an object version by a subject in an entity but creates a new version. In Figure 6.7b), $cg1.o1v1$ was updated and a new version $cg1.o1v2$ was created. In the diagram, two subtypes of "$wasDerivedFrom$" named $wasNewVersionOf$ and $wasUpdatedIn$ were identified to show the node dependencies. Note that, in the $merge$ operation diagram (Figure 6.6), the $entity$ node and $wasUpdatedIn$ arrow are not shown for simplicity.

Create$(s, o, entity)$: Create an object. The $create$ operation creates a data object in an entity. This is an initial version of the object. In Figure 6.7c), $cg1.o1v1$ was created in $cg1$ hence $cg1.o1v1$ has a $wasCreatedIn$ edge to $cg1$.

In addition to these three operations, [52] identified $createRO$ and $createRW$ operations as well as $kill$, $suspend$ and $resume$ operations. These operations are not discussed in this dissertation since they are identified mainly for authorization and information flow control purposes.

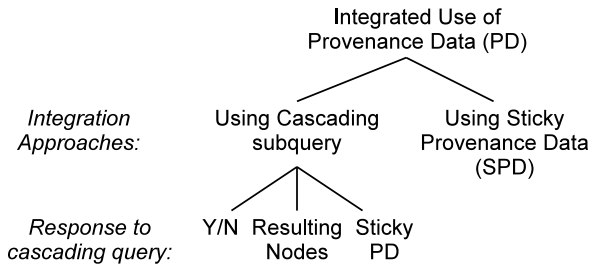

Figure 6.8: A Taxonomy of Provenance Data Integration in Multi-Provenance Systems (©2012 IEEE [62])

6.3 Provenance Data Sharing Approaches

In a provenance-aware environment, as data objects are created and used, the transaction information is captured as provenance data. Provenance-based access control utilizes the captured provenance information to control access to the underlying data. In a group-centric collaboration environment, data objects are shared and modified by multiple organizations/systems while the relevant provenance data are captured and stored in the local systems. While captured provenance data are readily available for access control within the local system, provenance-based access control in a group-centric collaboration environment requires integrated use of provenance data from other collaborating systems for effective access control. However, some provenance information maintained by a system may be too sensitive to be directly viewed or used by other systems. This dissertation demonstrates and discusses the issue relating the incorporation of an access control model in the context of group-centric secure collaboration environment, where multiple provenance systems act independently and therefore require some form of provenance data sharing for access control purposes.

For seamless use of provenance data in multiple provenance systems, there potentially are at least two approaches [62]. As shown in Figure 6.8, one approach is by utilizing cascading subqueries. The other approach is by utilizing sticky provenance data which are transmitted together with an associated data when the associated data is added/moved to another collaborating entity (organization or collaboration group).

Using Cascading Subquery

When an access request is parsed and a corresponding query is generated, the query is executed and evaluated against the local provenance system. Certain path patterns, however, would lead to objects that are added copies from different system entities. The local provenance system then does not contain sufficient information to make an access decision. It becomes necessary to ask for provenance information in another provenance system entity. In other words, subqueries need to be generated and evaluated against the provenance graphs maintained by other systems. These queries are generated from the original query with modifications of the path patterns to remove those already traversed. As a provenance data traversal can potentially span across many system entities, different subqueries may become necessary every time such a cross-system transition occurs. This type of query is labeled a *cascading subquery*.

In a uni-provenance setting, evaluating a query returns a set of resulting nodes upon which access control decisions are made. In a multi-provenance setting, once a provenance system receives a cascading subquery, there are at least three different ways such a subquery can be handled. For each case, different granularity of additional information is required from the requesting entity. Specifically, the receiving system can utilize its provenance data to return

- Y or N: ($startingNode_{new}$,$dPath_{new}$,$rule_{new}$) must be transmitted.
- Resulting Nodes: ($startingNode_{new}$,$dPath_{new}$) must be transmitted.
- Provenance Data Set: ($startingNode_{new}$) must be transmitted.

In the first case, the receiving provenance system is asked to make the access control decision. This could be the case when either the requesting entity does not possess enough computation resources to perform the evaluation itself or the receiving entity does not allow direct access to its provenance data. One such an example could be found in group collaboration between a government agency and a contracting company where the agency is not allowed to reveal the details of its provenance data. For evaluating such a subquery, the receiving provenance system requires

additional information. In particular, it requires the recomputed $startingNode_{new}$, $dPath_{new}$, and $rule_{new}$. Here, the $rule_{new}$ is necessary for the receiving provenance system to be able to make a decision for the requesting entity.

In the second case, the receiving provenance system requires the recomputed $startingNode_{new}$ and $dPath_{new}$ to evaluate the subquery but does not need make any access control decision for the requesting entity. This could be the case of a collaboration between a government agency and a contract company where the agency (requesting system) requires the contract company (receiving system) to reveal its provenance data as the agency may not want to reveal (part of) its access control policy.

In the third case, the receiving provenance system receives only the recomputed $startingNode_{new}$. Here, the system returns all provenance entities reachable by a recomputed $startingNode_{new}$. Similar to the second case, access control decisions are made by the requesting system.

Sticky Provenance Data

Another approach for resolving the stated issue is through the use of sticky provenance data. Discussion of this approach is explained with the example scenario as described in details in [62]

With sticky provenance data available in the local system, a cascading query may not be necessary to obtain provenance information necessary in making access control decisions. Rather, a locally generated query can be evaluated and completely executed against the local provenance graph similar to how a query under a uni-provenance system is treated. However this may not always be the case. The sticky provenance data of an object/version contains all the provenance information of that object/version up to the point in time when the information flow takes place. This means the transactions on the source data ($Org1.o1v2$ in the example) that occur after the information flow (phase 2 in the example) are not captured in the sticky provenance data unless the sticky provenance is constantly modified to reflect all the transactions that occur on all the previous versions ($Org1.o1v1$ in the example) of the added object ($Org1.o1v2$) and the added object itself.

Benefits of Sticky Provenance Data

There are multiple advantages associated with using sticky provenance data. One main advantage is the elimination of repeating computational efforts required from the source system. More specifically, once an organization data object is added to a group, policy rules for subsequent access request may base the decision evaluation on the provenance information of the organization data object. As discussed above, if no stick provenance data is used, some forms of cascading subqueries may be passed to the organization asking for results. The organization then would have to spend its computation resources in satisfying such requests. By sending sticky provenance data together with the data object copy, the requests and queries can be evaluated and answered locally.

Issues of Sticky Provenance Data

There are also multiple issues related to the static nature of sticky provenance data. Essentially, once sticky provenance data is moved to a new system entity together with the copy of an associating data object, the sticky provenance data only contains provenance information of the copied data object up to the point in time when such information flow occurs. Any processes occurring upon that object version thereafter are not captured by the sent sticky provenance data.

While this should be fine for the queries that need only backward traversal of provenance data, if a query requires checking all the transactions occurred against all of the related objects, forward traversal of provenance data may be necessary and the available sticky provenance data are not likely to be enough for access control decision. For example, suppose a policy rule which dictates that no $Update$ actions can be performed on a group version if the original organization object version had been updated in some ways (application-specific context). In the context of the scenario depicted in Figure 6.9, granting access to a request to update $CG1.o2v2$ requires no modifications had been done on $Org1.o1v2$. If the query only checks provenance data obtained in the sticky provenance data of $(CG1.o2v1)$ stored in $CG1$, then the access request is granted. However, as shown in $Org1$, some modifications had been done on $Org1.o1v2$ during phase 2 and

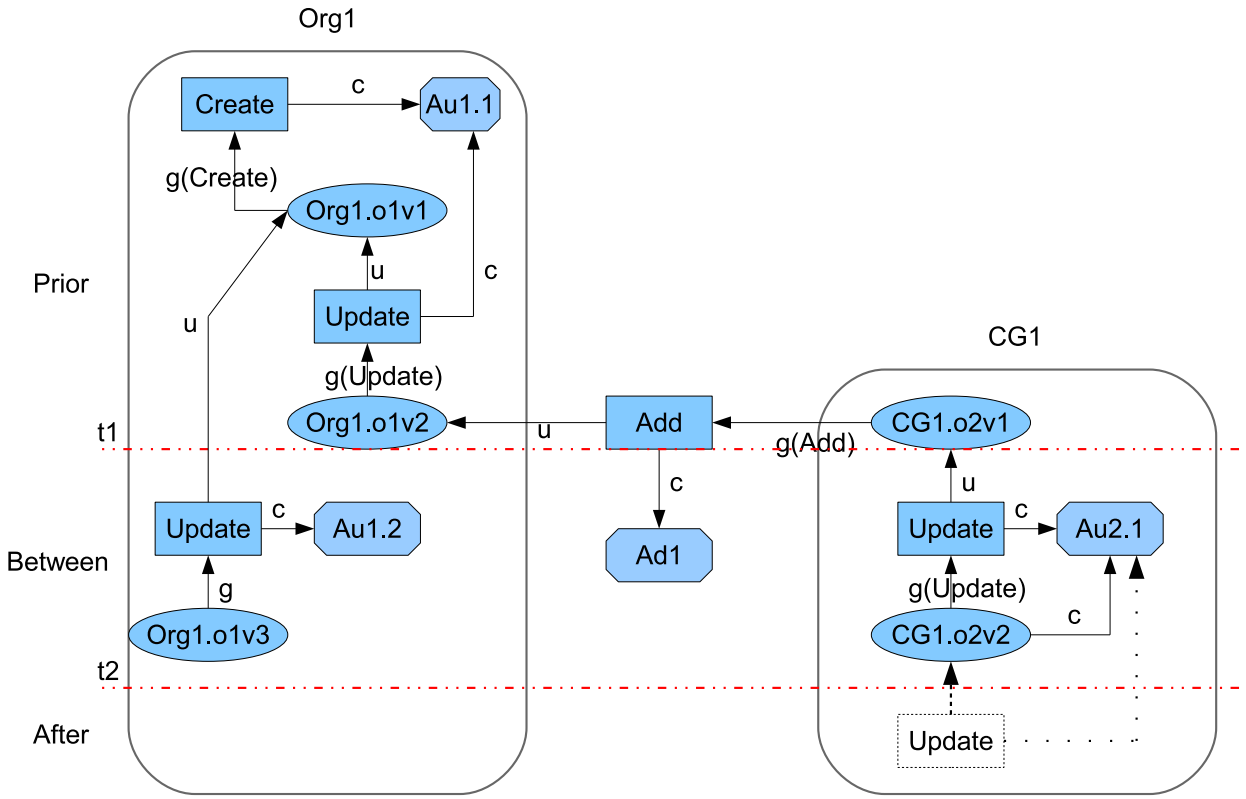

Figure 6.9: Sticky Provenance Data in Simplified Scenario (©2012 IEEE Nguyen2012IRI)

generated $Org1.o1v3$ at the current point in time. Such provenance information is not captured in the sticky provenance data of $(Org1.o1v2)$ because at the point in time when sticky provenance data of $(Org1.o1v2)$ was sent to $CG1$, such process had not occurred.

For sticky provenance data to be useful and reliable in such use cases, there require some mechanisms to keep the sticky provenance data up-to-date at all points in time or require additional traversal of provenance data that are not reflected in sticky provenance data. In practice, keeping up-to-date sticky provenance data could be quite costly or even unrealistic. This is further complicated as the complexity of the group-centric collaboration environment becomes larger.

Consider the scenario in Figure 6.10 which depicts information flow across three provenance systems where real-time updates of sticky provenance data are not available. Here, an object $o1v2$ is added to the collaboration group. Within the group, the added copy $o2v1$ is further updated and eventually generates $o2v2$, which is then added to a different organization $Org2$. In this information flow scenario, when $o1v2$ is added to $CG1$, all provenance data of $o1v2$ and the add

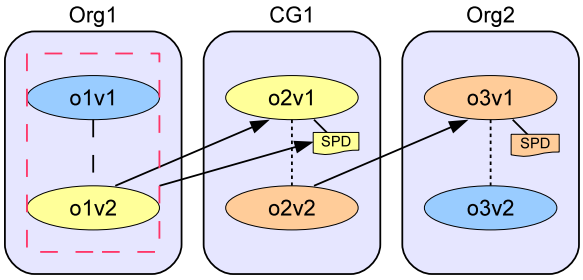

Figure 6.10: A “Sticky" Multi-Provenance Scenario (©2012 IEEE Nguyen2012IRI)

transaction information are transmitted to $cg1$ together with $o1v2$ and stored in sticky provenance data of $o2v1$. When $o2v2$ is added to $Org2$, it seems logical to combine sticky provenance data of o2v1 to sticky provenance data of o3v1. However, this is largely dependent on the application-specific context and may not be the case. In other words, sticky provenance data of $o3v1$ may or may not contain sticky provenance data of $o2v1$. For example, while a government agency $Org1$ may trust $CG1$ and shares its provenance data with $CG1$ using sticky provenance data, it may not allow further dissemination of the shared sticky provenance data to a contract company $Org2$. Furthermore, as the number of local provenance systems increases, the configuration and efforts required for updating sticky provenance data can grow exponentially in complexity.

Regardless, sticky provenance data can prove to be useful in some cases where forward traversal is not necessary.

Chapter 7: CONCLUSION AND FUTURE WORK

7.1 Summary

This dissertation proposed the design and demonstrated the capabilities of a framework of Provenance-based Access Control models to enhance the security of any provenance-aware computing environment. Specifically, the models make extensive use of a provenance data model that expresses the causality dependency relationships between primary provenance data entities in directed-acyclic graph form. Rich expressions of dependency path patterns and associated abstracted names constructs enable the specification of richer policy with utilization of provenance-aware rules in access control management. This allows the achievement of traditional as well as enhanced classes of Dynamic Separation of Duties. Furthermore, PBAC policies can naturally conform and be adapted into industry-compliant XACML policies. The implementation and evaluation of the PBAC models, through a proof-of-concept prototype, are done in the context of a hypothetical homework grading system. The results showed that with further work the approach can be feasible in a practical system deployment.

There also has been progress in adapting PBAC into a cloud Infrastructure-as-a-Service platform. This dissertation identified various architecture that enable provenance-awareness and PBAC capabilities within a single, multi-tenant cloud. The initial proof-of-concept implementation and evaluation demonstrated reasonable feasibility. There were also investigations on the aspects of deploying PBAC in a multi-system or multi-cloud environment. In this dissertation, the initial progress in this direction is discussed.

7.2 Future Work

This section describes and discusses several directions for future work.

7.2.1 Extended PBAC Models

The proposed base model is a foundational model for provenance-based access control and can be further extended for additional security enhancements. A crucial extension that enhances the base model is allowing **user-declared provenance data** in addition to the system-computed base provenance data. This means a user can declare a specific dependency between entities using a dependency name that is predefined by the system and available to the user. The user-declared dependency may cause conflict with the dependencies with the same name that are computed using base provenance data. For example, in Figure 3.2, a user au_2 may declare another user, say au_6 as an actual reviewer who generated a review o_{2v1} for the object o_{1v3} while the system-computed dependency points to au_2 as a reviewer based on the base provenance data. One approach to address this issue is by explicitly identifying the intentions of the declaration. For this, there are at least three types of intentions: ***inclusive, exclusive*** and ***denying*** types. The inclusive-intent dependency means, for example, both au_2 and au_6 are considered as reviewers who created o_{2v1}. If au_2 declared au_6 as an exclusive reviewer, the system computed dependency is voided. In addition, if allowed, au_2 may deny that he is not the actual reviewer. Furthermore, this extended model also needs to resolve an authorization issue of who can declare what kind of dependency intentions under which circumstance. The proposed base model can provide a concrete foundation for this extension and further pursue in this direction can further enrich the PBAC models and enhance overall system security.

7.2.2 Implementation of Provenance Capture Approaches for Provenance Service

In this dissertation, while a major emphasis is placed on the utilization of provenance data for access control purposes, the issue on how provenance data can be collected, captured and managed is also a significant aspect of provenance-aware systems.

There are several implementation frameworks for capturing provenance data with the intent of security in mind. The PASS system [58] aims to capture provenance information at the file level.

The PLUS system [22] captures provenance at the application level and use the information for taint analysis to handle insider threat [10]. In order to retrieve information from the provenance storage, many implementation of query languages are available. Park et al [66] employed SPARQL [40] with GLEEN [33] in a group collaboration environment. PQL [6] is a language in development that would provide useful functionality for provenance data queries. Other works [37, 80] focuses on provenance collection for distributed systems.

Provenance service prototype for OpenStack

This subsection identifies and describes three different approaches toward collecting provenance data in a cloud service:

- *Logs extraction*: as all events are collected and stored in a logging system, these data can be extracted and filtered into provenance data. This approach, however, is limited by the logging mechanisms and can introduce large overhead.

- *Database I/O instrumentation*: as all executed action instances ultimately involve a database read/write operations, it is possible to get according provenance data by collecting useful context information whenever a database read/write operations succeed. In Nova, it is possible to instrument the component responsible for these database operations, the *Conductor* component.

- *Communication queue*: As all communication between services in Nova is done via the use of a messaging queue (RabbitMQ technology), messages containing useful provenance information can be extracted from the queue for conversion into provenance data.

The components that can be implemented for the provenance service include:

- *ProvService Client*: a Python class resides within Nova architecture that has capabilities to perform the second and third approach to collect system events as outlined above. This component is also responsible for communicating with the ProvService server

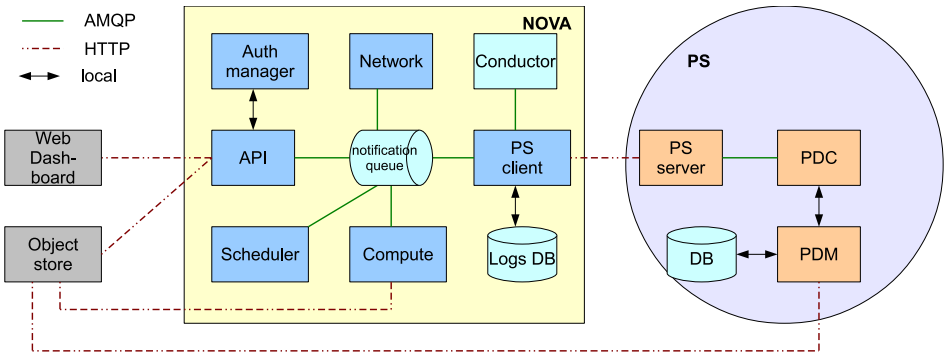

Figure 7.1: A Provenance Service for Nova Architecture

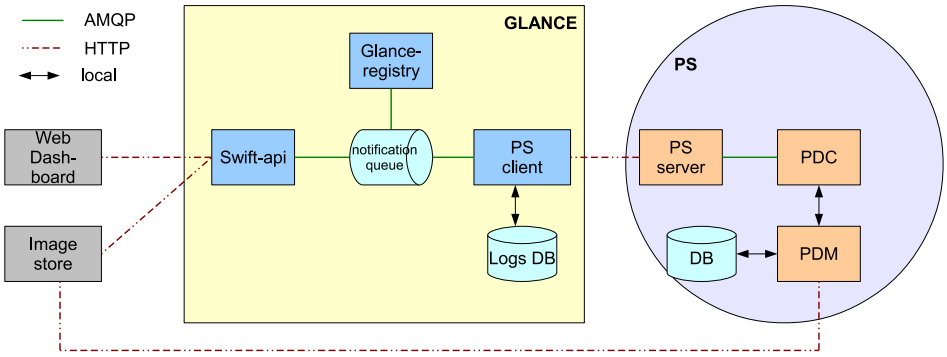

Figure 7.2: A Provenance Service for Glance Architecture

- *ProvService Server*: a component that resides within the provenance service, receives and handles HTTP requests from ProvService clients.

- PDC and PDM Python implementations that correspond to the associated architecture components.

Figures 7.1 and 7.2 depict potential implementation architecture for the Nova and Glance services in OpenStack. Designing and running more experiments can further consolidate the feasibility of the approach and prototype.

Appendix A: ADDITIONAL EXAMPLES

A.1 XACML Examples

Listing A.1: Sample XACML request

```
<Request>
  <Subject SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="http://www
        .w3.org/2001/XMLSchema#string"><AttributeValue>au4</AttributeValue></Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://
        www.w3.org/2001/XMLSchema#string"><AttributeValue>o2v0</AttributeValue></Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.
        w3.org/2001/XMLSchema#string"><AttributeValue>review</AttributeValue></Attribute>
  </Action>
</Request>
```

Listing A.2: Sample XACML response

```
<Response xmlns="urn:oasis:names:tc:xacml:1.0:context" . >
        <Result>
                <Decision>
                        Permit
                </Decision>
                <Status>
                        <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
                </Status>
        </Result>
</Response>
```

Listing A.3: Sample XACML policy

```
<Policy PolicyId="replacePolicy"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:ordered-permit-
    overrides">
<Target>
...
```

```
<Actions>
<Action>
<ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">replace</AttributeValue>
<ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
</ActionMatch>
</Action>
</Actions>
</Target>

<Rule RuleId="ReplaceRule" Effect="Permit">
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">

<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
<Apply FunctionId="provenance-query-SPARQL">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
</Apply>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
    hasPerformedActions:hasAttributeOf(actingUser)
</AttributeValue>
</Apply>
<Apply FunctionId="provenance-query-SPARQL">
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string" />
</Apply>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">wasUploadedBy
</AttributeValue>
</Apply>
</Apply>

</Rule>

<Rule RuleId="FinalRule" Effect="Deny" />
</Policy>
```

A.2 Provenance Data Samples in RDF-XML format

Listing A.4: Sample HWS Provenance Data in RDF-XML format

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://peac/hwgs#" >
  <rdf:Description rdf:about="http://peac/hwgs#replace99154">
    <j.0:wasControlledBy rdf:resource="http://peac/hwgs#au1"/>
    <j.0:usedInput rdf:resource="http://peac/hwgs#o1v99153"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://peac/hwgs#replace18721">
    <j.0:wasControlledBy rdf:resource="http://peac/hwgs#au1"/>
    <j.0:usedInput rdf:resource="http://peac/hwgs#o1v18720"/>
  </rdf:Description>
                ...
  <rdf:Description rdf:about="http://peac/hwgs#o3905v0">
    <j.0:wasGeneratedByReview rdf:resource="http://peac/hwgs#review3905"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://peac/hwgs#o1v71556">
    <j.0:wasGeneratedByReplace rdf:resource="http://peac/hwgs#replace71556"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://peac/hwgs#o1v52045">
    <j.0:wasGeneratedByReplace rdf:resource="http://peac/hwgs#replace52045"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://peac/hwgs#o1v49959">
    <j.0:wasGeneratedByReplace rdf:resource="http://peac/hwgs#replace49959"/>
  </rdf:Description>
                ...
</rdf:RDF>
```

Listing A.5: Sample Cloud IaaS Provenance Data in RDF-XML format

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://peac/osic#" >
  <rdf:Description rdf:about="http://peac/osic#modify_image2818">
    <j.0:wasControlledBy rdf:resource="http://peac/osic#au1"/>
    <j.0:usedInput rdf:resource="http://peac/osic#o1v2817"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://peac/osic#modify_image3339">
```

```
    <j.0:wasControlledBy rdf:resource="http://peac/osic#au1"/>
    <j.0:usedInput rdf:resource="http://peac/osic#o1v3338"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://peac/osic#get_image2383">
    <j.0:wasControlledBy rdf:resource="http://peac/osic#au1"/>
    <j.0:wasControlledBy rdf:resource="http://peac/osic#au2"/>
    <j.0:wasControlledBy rdf:resource="http://peac/osic#au3"/>
    <j.0:usedInput rdf:resource="http://peac/osic#vmi2v0"/>
  </rdf:Description>
              ...
              ...
</rdf:RDF>
```

A.3 Sample SPARQL Queries

Query (1): searches for all acting users who had replaced any previous version of $o1v50$.

```
PREFIX hw: <http://peac/hwgs#>
SELECT ?actingUser
WHERE {
  hw:o1v50
  ((hw:wasGeneratedByReplace)
  /(hw:usedInput/hw:wasGeneratedByReplace)*
  /hw:wasControlledBy)
  ?actingUser.
}
```

Query (2): searches for all objects/versions that were submitted by $hw : au2$.

```
PREFIX hw: <http://peac/hwgs#>
SELECT ?object
WHERE {
  ?object
```

```
    (hw:wasGeneratedBySubmit*
    /hw:wasControlledBy)
    hw:au2.
  }
```

Query (3): searches for all virtual machine objects/versions that were modified in the *Testing* tenant.

```
  PREFIX hw: <http://peac/osis#>
  SELECT ?vmobject
  WHERE {
    ?vmobject
    (osis:wasGeneratedByModify*
    /osis:hasAttribute(tenant))
    osis:Testing.
  }
```

A.4 Sample JSON Policies

A.4.1 Regular OpenStack Policies

Listing A.6: Default Nova policy in JSON

```
1 {
2         "context_is_admin":  [["role:admin"]],
3         "admin_or_owner":  [["is_admin:True"], ["project_id:\%(
              project_id)s"]], [1]
4         "default": [["rule:admin_or_owner"]], [2]
```

```
                ...
        "compute_extension:flavormanage": [["rule:admin_api"]],
           [3]
}
```

Listing A.7: Default Glance policy in JSON

```
{
    "context_is_admin":  "role:admin",
    "default": "",

    "add_image": "",
    "delete_image": "",
    "get_image": "",
    "get_images": "",
    "modify_image": "",
    "publicize_image": "role:admin",
    "copy_from": "",

    "download_image": "",
    "upload_image": "",
        ...
}
```

A.4.2 PBAC-enabled OpenStack Policies

Listing A.8: PBAC Nova policy in JSON

```

```

```
"compute.get_all":{
    "Rules":[
    {
        "id":"rule1",
        "Effect":"Allow",
        "Conditions":[
        {
            "cond1":[
            {
                "in": ["\%(sub_id)s"]
            }
            {
                "provquery":["\%(vm_id)s","wasCreatedBy"]
            },
        ]}]
    }]
}
```

BIBLIOGRAPHY

[1] Amazon Web Services. http://aws.amazon.com/.

[2] Google Compute Engine.
https://cloud.google.com/products/compute-engine.

[3] Infrastructuce Services Window Azures.
http://www.windowsazure.com/en-us/solutions/infrastructure/.

[4] OASIS, Extensible access control markup language (XACML), v2.0 (2005).

[5] OpenStack Open Source Cloud Computing Software. http://www.openstack.org/.

[6] PQL-path query language. http://www.eecs.harvard.edu/syrah/pql/. Accessed: 03/31/2012.

[7] RDFlib.
https://code.google.com/p/rdflib/.

[8] Cloud computing: An overview. *ACM Queue*, 7(5):2:3–2:4, Jun. 2009.

[9] Martin Abadi and Cedric Fournet. Access control based on execution history. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, pages 107–121, 2003.

[10] M.D. Allen, A. Chapman, L. Seligman, and B. Blaustein. Provenance for collaboration: Detecting suspicious behaviors and assessing trust in information. In *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 342 –351, Oct. 2011.

[11] Anindya Banerjee and David A. Naumann. History-based access control and secure information flow. In *Construction and Analysis of Safe, Secure, and Interoperable Smart Devices,*

International Workshop (CASSIS 2004), Revised Selected Papers, volume 3362 of Lecture Notes in Computer Science, pages 27–48. Springer-Verlag, 2005.

[12] Alex Berson and Stephen J. Smith. *Data Warehousing, Data Mining, and OLAP*. McGraw-Hill, Inc., New York, NY, USA, 1st edition, 1997.

[13] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. A formal model for isolation management in cloud infrastructure-as-a-service. In *Proceedings of the 8th International Conference on Network and System Security (NSS14)*, Oct. 2014.

[14] K.Z. Bijon, R. Krishnan, and R. Sandhu. A framework for risk-aware role based access control. In *2013 IEEE Conference on Communications and Network Security (CNS)*, pages 462–469, Oct 2013.

[15] K.Z. Bijon, R. Krishnan, and R. Sandhu. Towards an Attribute Based Constraints Specification Language. In *2013 International Conference on Social Computing (SocialCom)*, pages 108–113, Sept 2013.

[16] Barbara Blaustein, Adriane Chapman, Len Seligman, M. David Allen, and Arnon Rosenthal. Surrogate parenthood: protected and informative graphs. *Proc. VLDB Endow.*, 4(8):518–525, May. 2011.

[17] Uri Braun, Avraham Shinnar, and Margo Seltzer. Securing provenance. In *The 3rd USENIX Workshop on Hot Topics in Security*, USENIX HotSec, pages 1–5, Berkeley, CA, USA, Jul. 2008. USENIX Association.

[18] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *Proceedings of the international conference on Management of data*, SIGMOD, pages 539–550. ACM, 2006.

[19] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani Thuraisingham. A language for provenance access control. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 133–144. ACM, 2011.

[20] Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu, and Bhavani Thuraisingham. Transforming provenance using redaction. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 93–102. ACM, 2011.

[21] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, pages 74–83, New York, NY, USA, 2004. ACM.

[22] A. Chapman, B.T. Blaustein, L. Seligman, and M.D. Allen. Plus: A provenance manager for integrated information. In *2011 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 269 –275, Aug. 2011.

[23] Adriane P. Chapman, H. V. Jagadish, and Prakash Ramanan. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 993–1006, New York, NY, USA, 2008. ACM.

[24] Liang Chen and Jason Crampton. Risk-Aware Role-based Access Control. In *Proceedings of the 7th International Conference on Security and Trust Management*, STM'11, pages 140–156, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] J. Cheney. A formal framework for provenance security. In *IEEE 24th Computer Security Foundations Symposium (CSF)*, pages 281 –293, Jun. 2011.

[26] James Cheney. Causality and the semantics of provenance. *CoRR*, abs/1004.3241, 2010.

[27] James Cheney, Amal Ahmed, and Umut A. Acar. Provenance as dependency analysis. In *Proceedings of the 11th international conference on Database programming languages*, DBPL'07, pages 138–152, Berlin, Heidelberg, 2007. Springer-Verlag.

[28] James Cheney, Stephen Chong, Nate Foster, Margo Seltzer, and Stijn Vansummeren. Provenance: a future history. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, OOPSLA '09, pages 957–964, New York, NY, USA, 2009. ACM.

[29] Yuan Cheng. *Access Control for Online Social Networks Using Relationship Type Patterns*. PhD thesis, University of Texas at San Antonio, San Antonio, TX, USA, 2014.

[30] Yuan Cheng, Jaehong Park, and Ravi Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *PASSAT 2012*, pages 646–655. IEEE, 2012.

[31] Yuan Cheng, Jaehong Park, and Ravi Sandhu. A user-to-user relationship-based access control model for online social networks. In *Data and Applications Security and Privacy XXVI*, pages 8–24. Springer, 2012.

[32] Y. Cui and J. Widom. Lineage Tracing for General Data Warehouse Transformations. *The VLDB Journal*, 12(1):41–58, May. 2003.

[33] L.t. Detwiler, D. Suciu, and J.F. Brinkley. Regular paths in SPARQL: Querying the NCI thesaurus. In *AMIA Annual Symposium Proceedings*. American Medical Informatics Association, 2008.

[34] Guy Edjlali, Anurag Acharya, and Vipin Chaudhary. History-based access control for mobile code. In *ACM Conference on Computer and Communications Security*, pages 38–48. ACM Press, 1998.

[35] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, Aug. 2001.

[36] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, Aug. 2001.

[37] Ashish Gehani and Dawood Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In *Proceedings of the 13th International Middleware Conference*, Middleware '12, pages 101–120, New York, NY, USA, 2012. Springer-Verlag New York, Inc.

[38] V.D. Gligor, S.I. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 172 –183, May. 1998.

[39] Nabil I. Hachem, Ke Qiu, Michael A. Gennert, and Matthew O. Ward. Managing derived data in the Gaea scientific DBMS. In *Proceedings of the 19th International Conference on Very Large Data Bases*, VLDB '93, pages 1–12, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[40] S. Harris and A. Seaborne. SPARQL 1.1 Query language W3C working draft, Jan 2012. `http://www.w3.org/TR/sparql11-query/`. Accessed: 03/31/2012.

[41] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, StorageSS '07, pages 13–18, New York, NY, USA, 2007. ACM.

[42] Ragib Hasan, Radu Sion, and Marianne Winslett. Preventing history forgery with secure provenance. *ACM Transactions on Storage (TOS)*, 5(4):12:1–12:43, Dec. 2009.

[43] Thomas Heinis and Gustavo Alonso. Efficient lineage tracking for scientific workflows. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1007–1018, New York, NY, USA, 2008. ACM.

[44] Vincent C. Hu, David F. Ferraiolo, and et al. Guide to Attribute Based Access Control (ABAC) Definitions and Considerations. *NIST Special Publications 800-162*, Jan. 2014.

[45] Jing Jin and Gail-Joon Ahn. Role-based access management for ad-hoc collaborative sharing. In *Proceedings of the eleventh ACM symposium on Access control models and technologies (SACMAT)*, pages 200–209. ACM, 2006.

[46] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy*, DBSec'12, pages 41–55, Berlin, Heidelberg, 2012. Springer-Verlag.

[47] Xin Jin, Ram Krishnan, and Ravi Sandhu. Reachability Analysis for Role-based Administration of Attributes. In *Proceedings of the 2013 ACM Workshop on Digital Identity Management*, DIM '13, pages 73–84, New York, NY, USA, 2013. ACM.

[48] Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: Role-centric Attribute-based Access Control. In *Proceedings of the 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security: Computer Network Security*, MMM-ACNS'12, pages 84–96, Berlin, Heidelberg, 2012. Springer-Verlag.

[49] Florian Kelbert and Alexander Pretschner. Towards a Policy Enforcement Infrastructure for Distributed Usage Control. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 119–122, New York, NY, USA, 2012. ACM.

[50] Florian Kelbert and Alexander Pretschner. Data Usage Control Enforcement in Distributed Systems. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 71–82, New York, NY, USA, 2013. ACM.

[51] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and abstract syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, Feb. 2004.

[52] R. Krishnan, R. Sandhu, Jianwei Niu, and W. Winsborough. Towards a framework for group-centric secure collaboration. In *5th International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009*, pages 1 –10, Nov. 2009.

[53] Ram Krishnan, Ravi Sandhu, Jianwei Niu, and William H. Winsborough. Foundations for group-centric secure information sharing models. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 115–124, New York, NY, USA, 2009. ACM.

[54] Katja Losemann and Wim Martens. The complexity of evaluating path expressions in SPARQL. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 101–112, New York, NY, USA, 2012. ACM.

[55] Frank Manola and Eric Miller, editors. *RDF Primer*. W3C Recommendation. World Wide Web Consortium, Feb. 2004.

[56] Peter Mell and Timothy Grance. The NIST definition of cloud computing. Special Publication 800-145, 2011.

[57] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743 – 756, 2011.

[58] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proceedings of the annual conference on USENIX '06*

Annual Technical Conference, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[59] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. Making a Cloud Provenance-aware. In *First Workshop on on Theory and Practice of Provenance*, TAPP'09, pages 12:1–12:10, Berkeley, CA, USA, 2009. USENIX Association.

[60] Kiran-Kumar Muniswamy-Reddy, Peter Macko, and Margo Seltzer. Provenance for the cloud. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, pages 15–14, Berkeley, CA, USA, 2010. USENIX Association.

[61] Dang Nguyen, Jaehong Park, and R. Sandhu. Dependency path patterns as the foundation of access control in provenance-aware systems. In *4th USENIX Workshop on the Theory and Practice of Provenance*, TaPP'12. USENIX Association, Jun. 2012.

[62] Dang Nguyen, Jaehong Park, and R. Sandhu. Integrated provenance data for access control in group-centric collaboration. In *2012 IEEE 13th International Conference on Information Reuse and Integration (IRI)*, pages 255–262, 2012.

[63] Dang Nguyen, Jaehong Park, and R. Sandhu. A provenance-based access control model for dynamic separation of duties. In *11th Annual Conference on Privacy, Security and Trust*, PST 2013. IEEE, Jul. 2013.

[64] Dang Nguyen, Jaehong Park, and Ravi Sandhu. Adopting provenance-based access control in OpenStack cloud IaaS. In *Proceedings of the 8th International Conference on Network and System Security (NSS14)*, Oct. 2014.

[65] Qun Ni, Shouhuai Xu, Elisa Bertino, Ravi Sandhu, and Weili Han. An access control language for a general provenance model. In *Proceedings of the 6th VLDB Workshop on Secure Data Management*, SDM '09, pages 68–88, Berlin, Heidelberg, 2009. Springer-Verlag.

[66] Jaehong Park, Dang Nguyen, and R. Sandhu. On data provenance in group-centric secure collaboration. In *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 221 –230, Oct. 2011.

[67] Jaehong Park, Dang Nguyen, and R. Sandhu. A provenance-based access control model. In *10th Annual Conference on Privacy, Security and Trust*, PST 2012. IEEE, Jul. 2012.

[68] Jaehong Park and Ravi Sandhu. Towards Usage Control Models: Beyond Traditional Access Control. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, SACMAT '02, pages 57–64, New York, NY, USA, 2002. ACM.

[69] Jaehong Park and Ravi Sandhu. The $UCON_{ABC}$ usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, Feb. 2004.

[70] Jaehong Park, Ravi Sandhu, and Yuan Cheng. ACON: Activity-centric access control for social computing. In *2011 Sixth International Conference on Availability, Reliability and Security (ARES)*, pages 242–247. IEEE, 2011.

[71] Jaehong Park, Ravi Sandhu, and Yuan Cheng. A user-activity-centric framework for access control in online social networks. *Internet Computing, IEEE*, 15(5):62–65, 2011.

[72] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2006.

[73] Ravi Sandhu. Transaction control expressions for separation of duties. In *Proc. of the Fourth Computer Security Applications Conference*, pages 282–286, 1988.

[74] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based user-role assignment. In *DBSec*, pages 262–275, 1997.

[75] Ravi Sandhu and Qamar Munawer. The ARBAC99 model for administration of roles. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC)*, pages 229–238. IEEE, 1999.

[76] Ravi S. Sandhu. Lattice-Based Access Control Models. *Computer*, 26(11):9–19, Nov. 1993.

[77] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[78] Ravi S. Sandhu and Pierangela Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32:40–48, 1994.

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

[80] Chen Shou, Dongfang Zhao, Tanu Malik, and Ioan Raicu. Towards a provenance-aware distributed filesystem, 2013.

[81] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, Sep. 2005.

[82] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance techniques. Technical report, 2005.

[83] R.T. Simon and M.E. Zurko. Separation of duty in role-based environments. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 183 –194, Jun. 1997.

[84] Victor Tan, Paul Groth, Simon Miles, Sheng Jiang, Steve Munroe, and Luc Moreau. Security issues in a SOA-based provenance system. In *Proceedings of the International Provenance and Annotation Workshop*. Springer, 2006.

[85] Bo Tang, Qi Li, and Ravi Sandhu. A Multi-Tenant RBAC Model for Collaborative Cloud Services. In *Proceedings of the 11th IEEE Conference on Privacy, Security and Trust (PST)*, 2013.

[86] Bo Tang and Ravi Sandhu. Cross-Tenant Trust Models in Cloud Computing. In *Proceedings of the 14th IEEE Conference on Information Reuse and Integration (IRI)*, 2013.

[87] Bo Tang and Ravi Sandhu. Extending OpenStack access control with domain trust. In *Proceedings of the 8th International Conference on Network and System Security (NSS14)*, Oct. 2014.

[88] Bo Tang, Ravi Sandhu, and Qi Li. Multi-Tenancy Authorization Models for Collaborative Cloud Services. In *Proceedings of the 14th International Conference on Collaboration Technologies and Systems (CTS)*, 2013.

[89] William Tolone, Gail-Joon Ahn, Tanusree Pai, and Seng-Phil Hong. Access control in collaborative systems. *ACM Computing Surveys (CSUR)*, 37(1):29–41, 2005.

[90] Xinwen Zhang, Sejong Oh, and Ravi Sandhu. PBDM: a flexible delegation model in RBAC. In *Proceedings of the 8th ACM symposium on Access control models and technologies (SACMAT)*, pages 149–157. ACM, 2003.

VITA

Dang Nguyen was born in Ho Chi Minh City, Vietnam. He received his B.S. in 2009, subsequently entered the doctoral program in 2010, and received his interim M.S. in Computer Science in 2013, from the University of Texas at San Antonio. At the university, he joined the Institute for Cyber Security and has been working closely under the supervision of Dr. Ravi Sandhu and Dr. Jaehong Park. His research interests focus on the theoretical implication and practical utilization of provenance data for various aspects of access control. The applied domains of his work include legacy systems and cloud computing platforms.