DIGITAL PROVENANCE TECHNIQUES AND APPLICATIONS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Amani Abu Jabal

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Elisa Bertino, Chair

College of Science

Dr. Clifton Bingham

College of Science

Dr. Sorin Adam Matei

Brian Lamb School of Communication

Dr. Sonia Fahmy

College of Science

Approved by:

Dr. Clifton Bingham

Head of the School Graduate Program

To my beloved parents, sisters, and brothers.

ACKNOWLEDGMENTS

First and foremost, all praise and gratitude to Allah (God) for granting me the strength and determination to overcome all difficulties throughout my graduate study. I want to express my sincere appreciation for those who helped me to reach this point and those that without them being by my side, I will never be able to survive this long journey of Ph.D.

I want to thank my advisor Professor Elisa Bertino for her continuous support during my Ph.D. study and for giving me the opportunity to learn so much from her knowledge and experience. I have been lucky to work with Prof. Sorin Matei, Prof. Jorge Lobo, Prof. Alessandro Russo, Dr. Seraphin Calo, Dr. Dinesh Verma, Dr. Geeth De Mel, and Dr. Christian Makaya as mentors for various research projects. I would also like to thank my dissertation committee members: Prof. Sonia Fahmy, Prof. Clifton Bingham, and Prof. Sorin Matei, for their valuable comments, suggestions, and guidance. I want to thank my friends Pinar Yanardag, Daren Fadolalkarimen, Ruby Tahboub, Hasini Gunasinghe, and Maryam Davari, for their support and help. Special thanks to my friends Pinar and Daren, for their continuous support and being beside me whenever I needed their advice and help.

Finally, but not least, I would like to thank my mother, father, sisters, and brothers who gave me all the endless love and support to encourage me to complete my long journey to accomplish my dream.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

ABSTRACT

Abu Jabal, Amani Ph.D., Purdue University, August 2020. Digital Provenance Techniques and Applications. Major Professor: Elisa Bertino Professor.

This thesis describes a data provenance framework and other associated frameworks for utilizing provenance for data quality and reproducibility. We first identify the requirements for the design of a comprehensive provenance framework which can be applicable to various applications, supports a rich set of provenance metadata, and is interoperable with other provenance management systems. We then design and develop a provenance framework, called SimP, addressing such requirements. Next, we present four prominent applications and investigate how provenance data can be beneficial to such applications. The first application is the quality assessment of access control policies. Towards this, we design and implement the ProFact framework which uses provenance techniques for collecting comprehensive data about actions which were either triggered due to a network context or a user (i.e., a human or a device) action. Provenance data are used to determine whether the policies meet the quality requirements. ProFact includes two approaches for policy analysis: structure-based and classification-based. For the structure-based approach, we design tree structures to organize and assess the policy set efficiently. For the classification-based approach, we employ several classification techniques to learn the characteristics of policies and predict their quality. In addition, ProFact supports policy evolution and the assessment of its impact on the policy quality. The second application is workflow reproducibility. Towards this, we implement ProWS which is a provenance-based architecture for retrieving workflows. Specifically, ProWS transforms data provenance into workflows and then organizes data into a set of indexes to support efficient querying mechanisms. ProWS supports composite queries on three

types of search criteria: keywords of workflow tasks, patterns of workflow structure, and metadata about workflows (e.g., how often a workflow was used). The third application is the access control policy reproducibility. Towards this, we propose a novel framework, Polisma, which generates attribute-based access control policies from data, namely from logs of historical access requests and their corresponding decisions. Polisma combines data mining, statistical, and machine learning techniques, and capitalizes on potential context information obtained from external sources (e.g., LDAP directories) to enhance the learning process. The fourth application is the policy reproducibility by utilizing knowledge and experience transferability. Towards this, we propose a novel framework, FLAP, which transfer attribute-based access control policies between different parties in a collaborative environment, while considering the challenges of minimal sharing of data and support policy adaptation to address conflict. All frameworks are evaluated with respect to performance and accuracy.

1. INTRODUCTION

Data provenance is a set of metadata which captures information about a data object starting from its origin until its current state including all the activities and data objects that were parts of the transformation process. The term data object refers to data in any format (e.g., files, database records, or workflow templates). Data provenance is crucial for several purposes including detecting sources of errors and anomalies [1, 2], providing an audit trail for regulatory purposes, assessing data trustworthiness [3,4], evaluating data quality [5], and supporting reproducibility [6,7].

The use of data provenance requires a provenance management system which is capable of capturing, storing and querying sets of data provenance. Despite a large number of research efforts devoted to provenance management, only a few provenance infrastructures have been proposed. Chimera [8], myGrid [9], and Karma [10] are examples of provenance systems. However, the provenance models of these systems are tailored to their specific applications and therefore are not general enough. PASS [11] is a provenance management system for file systems; it provides a custom query tool, but it does not support security and different granularity levels for provenance metadata. In addition, there are two standard provenance models: Open Provenance Model (OPM) [12] and PROV [13]. These two models are interoperable and generic so that they are able to represent provenance for different systems and applications. However, their major limitation is that they are not able to represent metadata about access control policies. Ni et al. [14] have proposed a provenance model that focuses on access control policies for provenance. However, Ni's model is not able to support different granularity levels. Therefore there is the need for a comprehensive provenance management framework which addresses the limitations of existing frameworks. Sultana and Bertino [15] have defined such a framework that fulfills four requirements: a) including an expressive provenance model able to cap-

ture data objects at different granularities (e.g., a table record in a DBMS level or a file in an OS level), b) providing a query language which is independent of the provenance model representation, c) enabling interoperability with other provenance systems adopting other provenance models, and d) supporting a security mechanism by capturing the permissions granted to users at the time of data manipulation. Based on such initial definition we design and implement the first comprehensive provenance infrastructure (named *Secure Interoperable Multi-Granular Provenance Framework* (SimP) [16]) addressing these requirements.

Through our research work, we first investigate the use of provenance for assessing the quality of access control policies. Since provenance can be used for auditing and capturing all system transactions including access requests and their corresponding responses, such data can be used to analyze the system behavior and evaluate it in correspondence of the implemented access control system. Access control is a fundamental building block for secure information sharing [17]. It has been widely investigated and several access control models (e.g., the role-based access control (RBAC) model [18] and the XACML attribute-based access control model [19]) have been proposed. To date several approaches (e.g., see [20–24]) for policy analysis have been proposed (surveyed by Abu Jabal *et al.* [25]). However such previous approaches have two major drawbacks: they focus on assessing a single quality requirement (e.g., consistency), and require as input the specification of all possible access control requests. In particular, the latter drawback makes such previous methods not suitable for systems in which it is not often possible to determine in advance all actions that will be executed. Therefore it is not possible to statically analyze the quality of the policies. We need an approach by which policies are analyzed at "run-time" based on information on the actual behavior of subjects in the system. Therefore, we propose a framework, referred to as ProFact [26, 27], which utilizes provenance metadata to aggregate information about system behavior at execution time, and is thus able to address all quality requirements of access control policies both statistically and dynamically.

Another emerging application of data provenance is reproducibility. An example of reproducibility is to automatically generate a workflow of repetitive tasks and re-produce data. Thus, workflows have gained popularity in various domains (e.g., scientific discovery [28], business processes [29], and software modeling [30]) where users deal with data-intensive and sophisticated procedures, e.g., scientific experiments with workflow systems such as CRIS [31], Kepler [32], Taverna [33], VisTrails [34], and myGrid [35]. As a consequence of the widespread use of scientific workflows, online social websites such as myExperiments [36] have recently emerged aiming at supporting collaborative development, sharing and reuse of workflows. An important requirement towards such goals is the availability of rich and flexible workflow retrieval capabilities by which users can retrieve and compare workflows of interest for their research. The design and efficient implementation of a workflow retrieval tool require addressing several challenges. First, a unified and rich workflow model is required onto which one can map workflows expressed according to the models of the different scientific workflow management systems. Second, workflows should provide rich information to support the high-fidelity reproduction of the corresponding experiments. Scientists who design a workflow might not provide sufficiently detailed information about the necessary operations to re-execute an experiment. Hence, we need mechanisms that provide accurate and detailed information about workflows. Third, researchers are often interested in descriptive metadata and statistical information related to scientific workflows. Such statistical information includes usage percentage and users that contributed to the workflows. Hence, we propose an architecture, referred to as ProWS [6], to address such challenges based on the use of data provenance due to its ability to address all these challenges since it accurately records all the necessary information for reproducing workflows.

Another example of reproducibility is to generate access control policies (ACP) from logs (known as policy learning). However, given the challenges that are encountered when using the RBAC model including the inability of RBAC to handle the situations where some context and environmental information are essential for

making decisions regarding access requests, in addition to the inability of an RBAC policy to control the access of multiple objects, most modern access control systems have moved to another access control policy model, that is capable of overcoming the early mentioned challenges, which is the attribute-based access control model (ABAC) [17]. In ABAC, user requests to protected resources are granted or denied based on discretionary attributes of the users, the resources, and the environmental conditions [37]. However, a major challenge in using an ABAC model is the manual specification of the ABAC policies that represent one of the inputs for the enforcement engine. Such a specification requires detailed knowledge about properties of users, resources, actions, and environments in the domain of interest [38,39]. One approach to address this challenge is to take advantage of the many data sources that are today available in organizations, and use machine learning techniques to automatically learn ABAC policies from data. Suitable data for learning ABAC policies could be access requests and corresponding access control responses (i.e., *access control decisions*) in addition to other data sources including user directories (e.g., LDAP directories), organizational charts, workflows, and security tracking's logs (e.g., SIEM). Access control decisions are captured by a data provenance mechanism. For example, an organization may log past decisions taken by human administrators [40], or may have automated access control mechanisms based on low-level models, e.g., models that do not support ABAC. If interested in adopting a richer access control model, such an organization could, in principle, use these data to automatically generate access control policies, e.g., logs of past decisions taken by the low-level mechanism could be used as labeled examples for a supervised machine learning algorithm[1].

Furthermore, another track that complies with the reproducibility goals is to perform knowledge and experience transferability especially in coalition environments where a coalition member can obtain a log from another coalition member, and use

[1]Learning policies using logs of access requests and corresponding control responses does not necessarily that there is an existing access control system nor that the goal of policy learning is to re-produce the policies or validate them. Such logs may consist of examples of access control decisions provided by a human expert.

this log to learn new policies for similar missions. Technology advances in areas such as sensors, IoT, and robotics, enable new collaborative applications (e.g., autonomous devices). A primary requirement for such collaborations is to have a secure system which enables information sharing and information flow protection. Policy-based management system is a key mechanism for secure selective sharing of protected resources. However, policies in each party of such a collaborative environment cannot be static as they have to adapt to different contexts and situations. However one advantage of collaborative applications is that each party in the collaboration can take advantage of knowledge of the other parties for learning or enhancing its own policies. We refer to this learning mechanism as *policy transfer*. The design of a policy transfer framework has challenges, including policy conflicts and privacy issues. Policy conflicts typically arise because of differences in the obligations of the parties, whereas privacy issues result because of data sharing constraints for sensitive data. Hence, the policy transfer framework should be able to tackle such challenges by considering minimal sharing of data and support policy adaptation to address conflict.

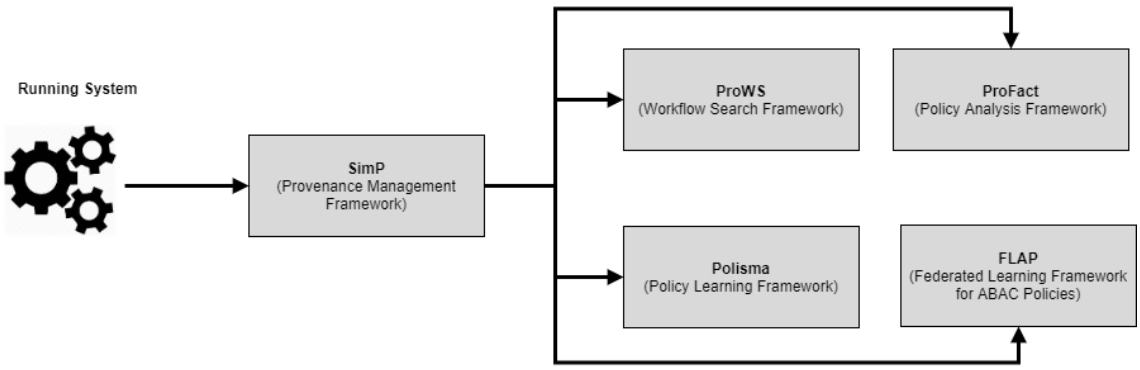


Fig. 1.1.: The Integration of SimP, ProFact, ProWS, Polisma, and FLAP frameworks

In what follows, we first introduce the SimP provenance framework. Next, we introduce four frameworks as applications for provenance: ProFact for quality assessment of access control policies, ProWS for workflows reproducibility, Polisma for policies reproducibility, and FLAP for policy transfer and thus reproducibility. There-

after, we outline the organization of this thesis. Fig. 1.1 shows the integration of the five proposed frameworks. The detailed architectures of the five frameworks will be described in the next subsections.

1.1 SimP: Secure Interoperable Multi-Granular Provenance Framework

Building a comprehensive provenance infrastructure involves addressing the following requirements ([15]):

- **multi-granular provenance model**: the infrastructure should provide an expressive provenance model able to capture data objects with different forms (e.g., file, database record, data in a workflow);
- **provenance query language**: the infrastructure should provide a query language which is independent of the provenance model representation;
- **security mechanism**: the infrastructure should capture the permissions granted to users at the time of data manipulation; the infrastructure should also restrict access to provenance storage as it might contain sensitive data;
- **interoperability services**: the infrastructure should be interoperable with other systems which adopt other provenance models.

We thus design and implement the first comprehensive provenance infrastructure addressing the four requirements discussed earlier. Our framework, Secure Interoperable Multi-Granular Provenance Framework (SimP) [16], is an extension of the framework on [15]. SimP includes the following contributions:

- A data provenance model extended from the provenance model proposed by Sultana and Bertino [14]. We provide specifications of this model according to a relational and graph model;
- A mapping ontology to support interoperation of our provenance model with both OPM and PROV;

- The integration of our provenance framework with the Computational Research Infrastructure for Science (CRIS) [31]. CRIS is widely used at Purdue University for managing scientific data from many different research areas, including biology, biochemistry, water management, and social sciences;
- A design, implementation and evaluation of a query language for SimP [16, 41].

1.2 ProFact: A Provenance-based Analytics Framework for Access Control Policies

Advances of technology in areas such as sensors, IoT, and robotics enable new collaborative applications. Such applications involve not only humans but also autonomous devices (e.g., drones, robots) [42]. A key requirement for such collaborations is represented by secure information sharing or information flow protection. Access control is a primary mechanism for selectively controlling accesses to a set of protected information.

An access control system decides, based on a set of access control policies, whether a subject (e.g., user, process, device, application) can access a specific information resource (e.g., files) for performing a certain action (e.g., read, write). A large number of research efforts have been devoted to defining access control models (surveyed by Bertino et al. [17]) including RBAC [18] and XACML [19].

For an access control system to be effective and efficient, it is critical that policies be of "good quality" in order to make sure that the appropriate access control decisions are taken. Towards this, we introduce a set of quality requirements and propose a framework for policy analysis which utilizes provenance metadata to assess the quality of policy sets [26].

As shown in Fig. 1.2, ProFact is based on three main phases: data collection, policy analysis, and policy evolution. The data collection phase uses a provenance logging component which captures all actions executed in the system in addition to all changes made on the access control policy set and stores them in the provenance

repository. To support the data collection phase, the framework maintains the tree structures (i.e., policy and transaction trees) which abstract the necessary information for the analysis phase. The analysis phase includes two types of analysis approaches: structure-bases and classification-based. The results of the analysis approaches are aggregated into a separate repository referred to as *policy analysis repository*. Moreover, the framework provides automatic support for policy evolutions based on the results of policy analysis. Based on the policy analysis, our evolution tool handles the modifications to "low quality" policies and re-analyzes them. The overall novel framework, which we refer to *ProFact* (standing for *Provenance-based Analytics Framework for Access Control Policies*), has been implemented and experimental results are reported from the implemented prototype.

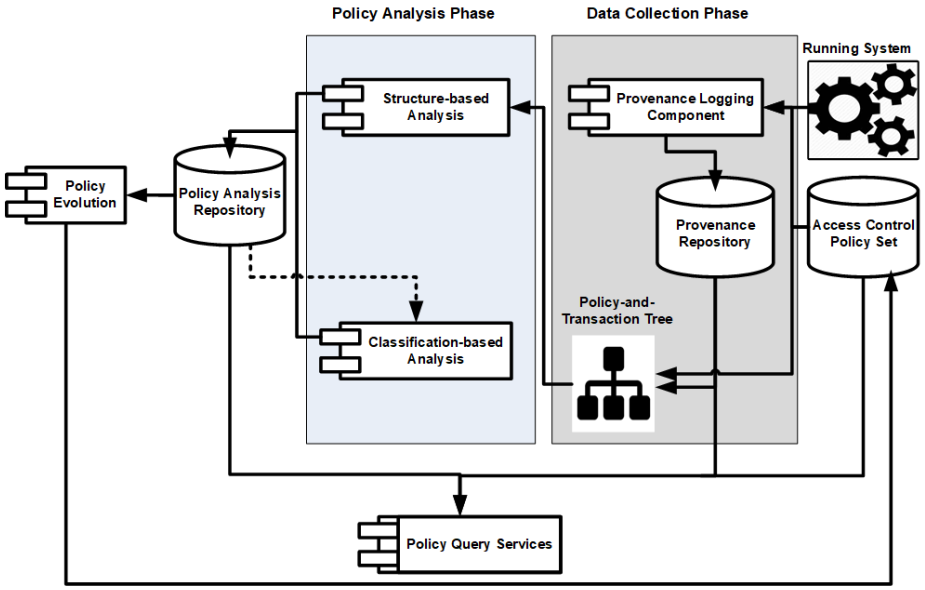


Fig. 1.2.: The Infrastructure of the ProFact Framework

While implementing the framework, we were not able to obtain a large set of access control policies in a real system. Alternatively, we generated a synthesized dataset for experimental purposes. Moreover, machine learning algorithm (particularly classification techniques) sufficient dataset to efficiently learning the characteristics of the dataset. In our framework, we addressed the challenges of classification techniques

at two levels: data level by sampling more dataset only in the learning phase and approach level by devising a classification scheme that combines the classification results of multiple well-known classifiers.

1.3 ProWS: Provenance-based Scientific Workflow Search

Workflows enable automating repetitive tasks performed by experts and scientists in research systems. Thus, it is very crucial to benefit from workflows and utilize the experience of other scientists. However, the key factor to facilitate workflows reproducibility is to have a mechanism which accurately records all the detailed activities comprising a workflow. The SimP framework allows one to capture detailed information related to scientific experiments and then use this information to create workflow repositories. By using SimP, we can address the three challenges discussed earlier: a) scientific workflows can be represented by our provenance model independently from underlying systems storing and managing the workflows; b) our provenance model is able to accurately represent workflows by capturing all transactions related to workflow tasks; and c) provenance captures all metadata related to workflows which can be aggregated for statistical and metadata queries.

We thus design and implement ProWS which consists of four-sequential stages. In the first stage, the workflow management system is integrated with a provenance mechanism to capture and store provenance information in the provenance repository. Then, provenance metadata is transformed into workflows. Workflows are modeled by directed graphs with nodes and edges; nodes represent functional modules while edges represent the dependencies between modules. The workflows graph representation is derived from the provenance model. In the third stage, several index structures are allocated which organize workflow data according to different representations supporting efficient workflow retrieval. The architecture of our framework is shown in Fig. 1.3.

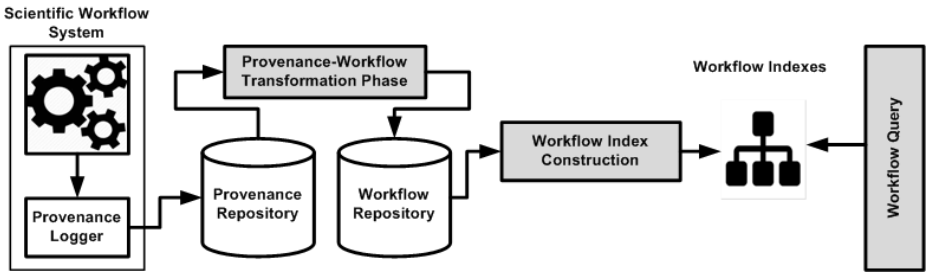


Fig. 1.3.: Provenance-based Scientific Workflow Search Architecture

Since workflows are represented as graphs, searching workflows can be expensive. For example, supporting pattern matching against a workflow is an NP-complete problem [43]. Therefore, answering workflow queries by sequentially scanning all workflows in the workflow repository is not a suitable query processing strategy. To address such issue, we have developed a set of index structures to reduce the search space and thus reduce the query processing time.

1.4 Polisma: A Framework for Learning Attribute-based Access Control Policies

Most modern access control systems are based on the attribute-based access control model (ABAC) [17]. In ABAC, user requests to protected resources are granted or denied based on discretionary attributes of users, resources, and environmental conditions [37]. ABAC has several advantages. It allows one to specify access control policies in terms of domain-meaningful properties of users, resources, and environments. It also simplifies access control policy administration by allowing access decisions to change between requests by simply changing attribute values, without changing the user/resource relationships underlying the rule sets [37]. As a result, access control decisions automatically adapt to changes in environments, and in user and resource populations. Because of its relevancy for enterprise security, ABAC has been standardized by NIST and an XML-based specification, known as XACML, has been developed by OASIS [44]. There are several XACML enforcement engines, some

of which are publicly available (e.g., AuthZForce [45] and Balana [46]). Recently a JSON profile for XACML has been proposed to address the verbosity of the XML notation.

When addressing the challenge of learning ABAC policies from logs, a few key requirements must be satisfied. The learned policies must be *correct* and *complete*. Informally, an ABAC policy set is correct if it is able to make the correct decision for any access request. It is complete if there are no access requests for which the policy set is not able to make a decision. Such a case may happen when the attributes provided with a request do not satisfy the conditions of any policy in the set.

To meet these requirements, the following issues need to be taken into account when learning ABAC policies:

- *Noisy examples.* The log of examples might contain decisions which are erroneous or inconsistent. The learning process needs to be robust to noise to avoid learning incorrect policies.

- *Overfitting.* This is a problem associated with machine learning [47] which happens when the learned outcomes are good only at explaining data given as examples. In this case, learned ABAC policies would be appropriate only for access requests observed during the learning process and fail to control any other potential access request, so causing the learned policy set to be incomplete. The learning process needs to generalize from past decisions.

- *Unsafe generalization.* Generalization is critical to address overfitting. But at the same time generalization should be *safe*, that is, it should not result in learning policies that may have unintended consequences, thus leading to learned policies that are unsound. The learning process has to balance the trade-off between overfitting and safe generalization.

Thus, we investigate the problem of learning ABAC policies, and proposes a learning framework that addresses the above issues. Our framework learns from logs of

access requests and corresponding access control decisions and, when available, context information provided by external sources (e.g., LDAP directories). We refer to our framework as *Polisma* to indicate that our ABAC **po**licy **l**earner uses m**i**ning, **s**tatistical, and **ma**chine learning techniques. The use of multiple techniques enables extracting different types of knowledge that complement each other to improve the learning process. One technique captures data patterns by considering the frequency and another one exploits statistics and context information. Furthermore, another technique exploits data similarity. The assembly of these techniques in our framework enables better learning for ABAC policies compared to the other state-of-the-art approaches.

Polisma consists of four steps. In the first step, a data mining technique is used to infer associations between users and resources included in the set of decision examples and based on these associations a set of rules is generated. In the second step, each constructed rule is generalized based on statistically significant attributes and context information. In the third step, authorization domains for users and resources (e.g., which resources were accessed using which operations by a specific user) are considered in order to augment the set of generalized rules with "restriction rules" for restricting access of users to resources by taking into account their authorization domain. Policies learned by those three stages are safe generalizations with limited overfitting. To improve the completeness of the learned set, *Polisma* applies a machine learning (ML) classifier on requests not covered by the learned set of policies and uses the result of the classification to label these data and generate additional rules in an "ad-hoc" manner.

1.5 Federated Learning for Attribute-based Access Control Policies

Recent policy-based management systems are attribute-based (AB). In particular, policy rules are expressed as conditions against domain-meaningful properties of coalition parties, resources, actions, and environments. This approach simplifies pol-

icy administration as policy decisions automatically adapt between requests based on changes of attribute values. Such a capability is critical to enhancing the autonomy of coalition parties in the era of multi-domain operation (MDO) [48] involving coalitions. In coalition MDO, coalition parties operating in the land, air, sea, or cyber will come together to achieve collective goals by sharing multiple viewpoints about emerging situations. Since coalition MDO contains multiple parties and types of resources, approaches to simplify policy specifications and a systematic approach to autonomously adapt policies according to the context will be critical. A major challenge is the specification of the AB policies representing the key input for policy enforcement. Since in MDO, we may typically deal with local contexts and situations, the needed detailed knowledge may be lacking. Addressing this challenge requires a distributed intelligence approach for policy learning that is able to: (i) combine datasets available at coalition parties (e.g., directories, organizational charts, logs, and existing local policies); and (ii) use machine learning (ML) to infer AB policies from these combined data.

In coalitions, parties can each have their own datasets, and combining these datasets can enhance the learning outcomes. In some cases, coalition members may only share their own local policies but not the data they used to learn their policies. In practice, a combination of those cases (i.e., sharing datasets, sharing policies) may occur. A federated approach is thus required for learning policies from a broad variety of data and knowledge, including raw data, policies expressed as rules, and ML models. It is, therefore, critical to develop an AB policy learning framework able to learn from multiple data sources while at the same time assuring that each party can generate accurate policies. To the best of our knowledge, there are no existing frameworks addressing such requirement. Recently, we have proposed a learning framework [49] to learn policies of interest to a single party that uses only the data of that party. Therefore, such a framework needs to be extended in a way that enables its utilization in such a federated environment.

Towards enabling the learning framework to accommodate the differences that might be encountered in a federated environment, one main issue is the conflicts that might arise in the process of interchanging the policies between different coalition parties. Those conflicts are expected as a result of the regulation differences and security specifications. Thus, to address this issue, a similarity analysis, as well as qualitative analysis, should be performed by the target party to ensure the correctness and accuracy of the learning and transfer process. Another issue is the timeline and degree of the interaction between the policies of a pair of parties to perform a better learning and interoperability process. Therefore, we propose four approaches with different levels of interaction aiming to find the best learning output.

1.6 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 introduces the SimP provenance framework. Chapter 3 introduces the ProFact provenance-based policy analysis framework and Chapter 4 introduces our workflow searching framework. Chapter 5 presents a learning framework for ABAC policies. Chapter 6 presents a federated learning framework for transferring ABAC policies among multiple parties in a coalition environment. Chapter 7 concludes the thesis and outlines directions for future work.

2. SIMP: SECURE INTEROPERABLE MULTI-GRANULAR PROVENANCE FRAMEWORK

The SimP provenance framework as shown in Fig. 2.1 is composed of the following components: provenance model, provenance capturing component, interoperability component, storage, and query component.

2.1 Provenance Model

In our provenance model, called SimP, the provenance of a data object records the history of its input *data*, *processes*, *operations*, *communications*, *actors*, *environments*, and *access controls*.

A *process* manipulates input data objects by performing a sequence of operations to generate other data objects. A process might be a service in a user application (e.g., a workflow in a scientific experiment application) or an operating system level process (e.g., executing a script by shell command in UNIX). At a more detailed level, an *operation* executes a task which is part of process execution. The operations may generate/modify persistent data or intermediate results. In provenance, it is crucial

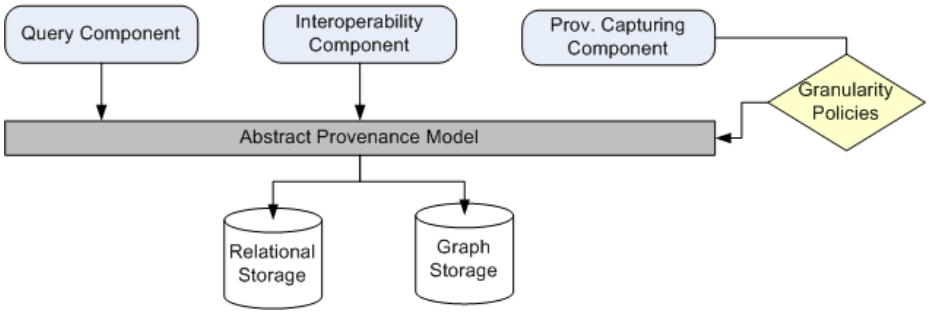


Fig. 2.1.: SimP Framework

to capture the origin of the generated data. Such information is captured by a *data lineage* entity. In data lineage, data is described by the other data and the operations used to derive it. Lineage helps in producing the data dependency graph of a data object and implicitly describing the process dependency.

The operations in the same process or in two different processes interact by real or virtual messages. In our model, such an interaction is referred to as *communication*. There are two types of communication between operations. The first type refers to the completion of operation followed by the start of another operation. The second type refers to an operation executed (i.e., started and completed) within the execution of another operation. We refer to the first type of communication as *sequential* and to the second as *composition*. A communication may involve data passing if the operation which initiated the communication generates data. Examples of communications include data flow, copy-paste in UNIX. On the other hand, a process may initialize another process to be executed. The process which invokes the initialization is the parent process and the newly created process is the child process.

Processes (including its operations) and data are manipulated by *actors* which can be people or workflows. Capturing information about actors, who actuate the activities changing data objects, helps in detecting intrusion or system changes. A user may authorize other users to perform certain activities on his behalf. In this model, *data* objects are attributed to actors to identify users who inserted input data or generated output by executing a process. Processes also have a context that affects their execution and output. Such a context is represented by the *environment* which refers to a set of parameters, and system configurations. Environment information helps in understanding the system context and performance in which processes were executed and data output were generated.

Our model is security-aware. Following the model by Ni *et al.* [14], our model is able to represent the security access policies of actors at the time of data manipulation by the actors. Information about access control policies includes which actors are authorized to utilize which processes and operations on which data. Such infor-

mation is modeled in *Access Control Policy* entity which includes actor and policy information. The policy object might refer to processes, or operations specified by the policy subject.

All fundamental provenance entities contain a domain attribute which might be used to specify the scope of provenance information (e.g., where processes and data manipulations executed), especially when having a provenance storage for different systems. In addition, the domain value might include more detailed scope information (e.g., a particular application or workflow). The domain attribute is essential for providing an abstract domain view of the provenance graph.

Our framework provides specifications of the provenance model in two representations: relational and graphs.

2.1.1 Relational Representation

In this section, we introduce a relational model representation of SimP which is shown in Fig. 2.2. Based on the abstract description of our model, the fundamental entities are stored in six fundamental tables (i.e., *Data*, *Processes*, *Operations*, *Actors*, *Environments*, and *Access Control Policies*); in addition, there are tables maintaining many-to-many relationships among the fundamental tables (i.e., *Lineages*, *Communications*, *Process Input Data*, *Process Output Data*, *Operation Input Data*, *Operation Output Data*, and *Delegations*). These tables are connected through a set of referencing relations (i.e., foreign key constrains). Each table has a unique identifier and consists of several attributes.

Each data record contains a description, value, and Actor ID. An example of a data object is a file. The actual data object identifier is different from data record identifier in the provenance storage. Let us assume the data object is a script file named f_i. This file might be edited by different users at different times. Thus, the provenance storage contains multiple records for f_i and each record is identified by different Data ID, but the description attribute of these records are similar (i.e., f_i).

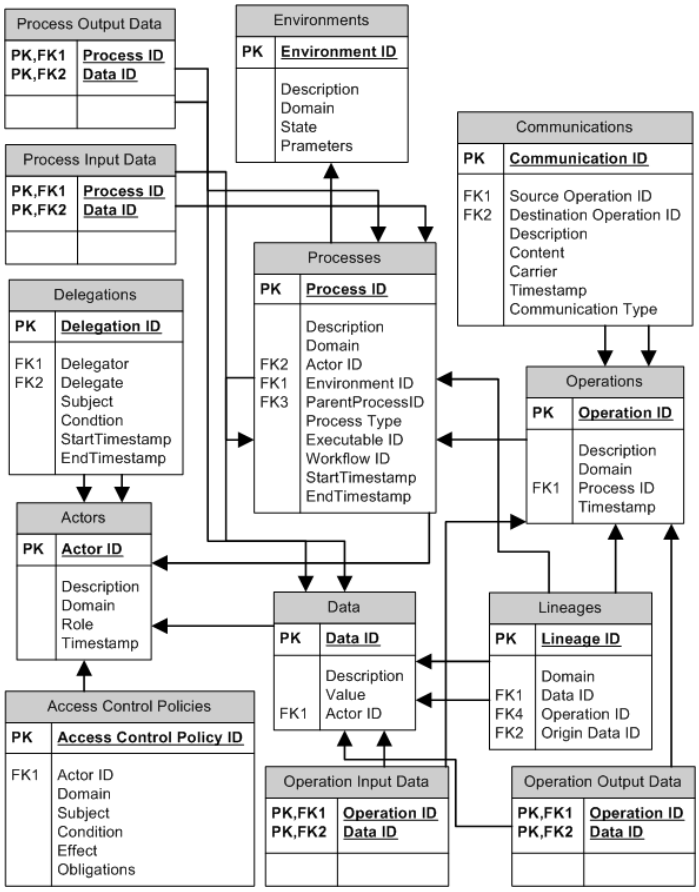


Fig. 2.2.: SimP Relational Model

In our model, every data object is attributed to an actor to trace who created the object or generated it by executing a process.

Each process record has a unique ID and is executed by an actor in a certain environment. A process manipulates certain data and may generate output data, so process input and output data are recorded in the Process Input Data and Process Output Data tables. The process is categorized into two types: application process or system process. If the process belongs to an application (e.g., a scientific workflow), it contains a workflow ID. Otherwise, it is a system process. In the case of system process, the workflow ID attribute refers to the workflow hosting the application process which encapsulates the system process. Each process has a description which is the actual identifier of the process (e.g., executing the script file (f_i)). A process

might be executed multiple times so to capture the provenance about the same process multiple times we have an automatic generated ID to distinguish process records. A process might be initialized (i.e., forked) by another process, so we store such information in the parent process attribute.

Operation record attributes include ID, description, and process ID. Depending on applications, the operation description attribute might contain different definitions (e.g., a function name or a block of source code statements). The output of an operation might be intermediate or persistent. The output is intermediate if it is used as input for another operation while a persistent output is final. The persistent output of operations is considered also as the output for the container process. On the other hand, all input data of a process can be input data for all its contained operations.

Communication record attributes include a description, carrier, the source operation ID, the destination operation ID, and type of communication. The detail level of the communication description depends on the applications. The communication channel (e.g., HTTP or SOAP) is described by the carrier attribute. The type of communication between two operations might be *sequential* or *composition*.

Lineage record attributes include lineage ID, data ID, and operation ID that produced this data operating on a certain input data IDs. A data item has multiple lineage records if it is generated by an operation which took multiple input data.

The attributes of an actor record include actor ID, description, and role. A role is a job of the actor. For an actor, the most recent record contains its current job while its previous records might include its previous jobs. An actor can delegate another actor to execute a process. Such information is stored in a Delegation record which contains a delegator, delegate, subject, condition, start timestamp and end timestamp. The subject might refer to processes or operations while the condition refers to the identifier of the subject.

The content of an Environment record includes domain, state and parameters attributes. In our model, each process belongs to one environment while an environ-

ment contains multiple processes. An environment timestamp is determined by its parent process record (i.e., the first process executed within the environment).

The attributes of an access control policy record include access policy ID, actor ID, subject, condition, effect, obligations. Such records capture the security access policies of actors at the time of manipulating data of interest. The actor ID records the actor privileged to perform certain operations or processes. The policy subject might refer to processes, operations, or communications while the condition refers to the identifier (e.g., process or operation description) of the subject.

2.1.2 Graph Representation

The relational model is suitable for storing provenance in a relational database. However, as the standard provenance models (i.e., OPM and PROV) are modeled according to a graph-based format, developing an interoperability ontology mapping from these models onto our model requires a corresponding graph representation of our model. Thus, we also introduce a graph model specification of SimP.

As shown in Fig. 2.3, our graph model consists of six nodes and twelve types of edges. Our graph specifications follow the standard graph models provided by OPM and PROV.

The graph nodes include *Data*, *Process*, *Operation*, *Actor*, *Environment*, and *Access Control policy*. Each graph node has a set of attributes similar to its corresponding entity table described in the relational model.

An edge in the SimP model is a relationship between the source of the edge and the destination of the edge. The kinds of the relationship (and thus the meaning of the edges) are specified by the types of the nodes that are the end-points of the edges. These types of relations are defined as follows:

- An edge of type *used* connects a source Process or Operation to a destination Data. It indicates that the process/operation required the availability of certain data to be able to complete its execution.

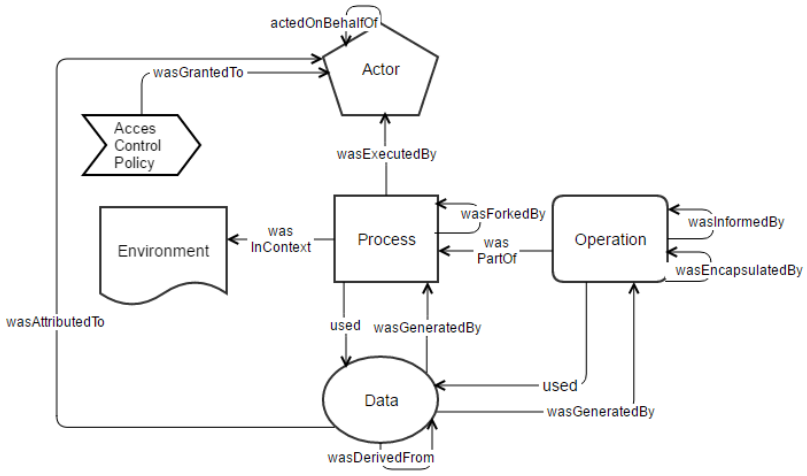


Fig. 2.3.: SimP Provenance Model

- An edge of type *wasGeneratedBy* connects a source Data to a destination Process or Operation. It indicates that the process/operation was required to initiate its execution for the data to have been generated.

- An edge of type *wasDerivedFrom* connects a source Data to a destination Data. It indicates that the destination data needs to have been generated for the source data to be generated.

- An edge of type *wasExecutedBy* connects a source Process to a destination Actor. It indicates that a process with all its operations was executed by the actor.

- An edge of type *wasInformedBy* connects a source Operation to a destination Operation. It indicates that the execution of the destination operation was followed by the execution of the source operation.

- An edge of type *wasEncapsulatedBy* connects a source Operation to a destination Operation. It indicates that the execution of the source operation is part of the execution of the destination operation.

- An edge of type *wasPartOf* connects a source Operation to a destination Process. It indicates that the execution of the source operation is part of the execution of the destination process.

- An edge of type *wasForkedBy* connects a source Process to a destination Process. It indicates that the execution of the source operation is initiated by the destination process.

- An edge of type *wasInContext* connects a source Process to a destination Environment. It indicates that the execution of the source process was in the context of the destination Environment.

- An edge of type *wasGrantedTo* connects a source Access Policy to a destination Actor. It indicates that the destination actor had the source access control policy at the time of capturing the provenance.

- An edge of type *actedOnBehalfOf* connects a source Actor to a destination Actor. It indicates that the action performed by the source actor was granted by the destination actor.

- An edge of type *wasAttributedTo* connects a source Data to a destination Actor. It indicates that the source data was manipulated by the destination actor.

Based on the proposed provenance specifications for relational and graph-based representations, our framework supports two types of storage: relational database (i.e., MySQL) or graph database (i.e., Neo4J) for storing provenance metadata. For this sake, our framework has an abstract storage interface. This abstract interface communicates with either MySQL adapter or Neo4J adaptor. The default storage is Neo4J and the framework can be configured to change to the second database. The administrator is able to change the target storage type at any time, but he should first import the database from Neo4J to MySQL (or vice versa). We will show in section III some insights on the choice of storage type based on provenance query types.

2.2 Interoperability With Other Models

Provenance interoperability means that provenance framework is able to integrate and convert provenance information represented by different provenance models in order to facilitate provenance data interchange. Our model supports interoperability with two well-known standard provenance models: OPM [12] and PROV [13]. Below we provide some background about OPM and PROV model ontologies and address the interoperation challenge by defining a mapping ontology that maps provenance information expressed in each of those two standard models into provenance information expressed in SimP.

2.2.1 Mapping from the OPM Standard Model

OPM [12] represents provenance information in a directed graph which consists of nodes and edges. The nodes comprise three types of entities; *Artifacts* (i.e., data) which represent resources, *Processes* which represent actions or steps of action performed on artifacts, and *Agents* (i.e., actuators) which control the processes. The edges represent the dependencies and relations among the entities. There are five types of edges: *used*, *wasControlledBy*, *wasGeneratedBy*, *wasDerivedFrom*, and *wasTriggeredBy*. Each edge is distinguished by its source and destination: a process used an artifact, a process was controlled by an agent, an artifact was generated by a process, an artifact was derived from another artifact, and a process was triggered by another process. OPM also introduced the concept of *account* to represent a particular view of the provenance.

The conversion from OPM to our provenance model is straightforward because of the rich vocabularies in our model. Table 2.1 shows the mapping from OPM graph to our provenance model graph. Besides the intuitive mapping provided in Table 2.1, we need to consider the following:

- OPM does not have a finer granularity level of description for a process as a set of operations. When converting a process from OPM to SimP, we create a

process and operation and link them by a WasPartOf edge. Mapping an OPM process into a SimP operation assumes that the relations connected with OPM process are mapped onto relations with SimP operation. So the purpose of creating a dummy SimP process is only to host the created SimP operation.

- In OPM, the intercommunication between a process and another process is identified by one edge type, that is, WasTriggeredBy, which does not capture the exact meaning of process-process relations (i.e., communication relationship or parent-child relationship). In our model, these relations are represented by three types of edges (WasForkedBy, WasEncapsulatedBy, and WasInformedBy). WasForkedBy represents the parent-child relation between two processes. On the other hand, WasInformedBy and WasEncapsulatedBy represent communications between two operations (in the same process or different processes). More specifically, a WasInformedBy represents a sequential communication and a WasEncapsulatedBy represents a composition communication. When performing a conversion, we map the OPM WasTriggeredBy edge to WasInformedBy in our model under the assumption that all OPM processes are related with sequential communication.

- OPM supports a particular view of provenance by including the "account" attribute of the graph nodes. By contrast, in our model, we support such a view of provenance using the environment node. So each node in OPM that has an account attribute is mapped onto a WasInContext edge connecting the mapped node with the environment node.

2.2.2 Mapping from the PROV Standard Model

A W3C provenance standard model called PROV [13] was published in 2013 based on a revision of OPM. PROV graph contains three types of nodes: Entities (i.e., data) to represent resources, Activities to refer to actions performed on entities and Agents

Table 2.1.: Mapping From OPM To SimP

	OPM	SimP
Nodes	Process	Process, Operation, WasPartOf
	Artifact	Data
	Agent	Actor
Edges	Used	Used
	WasGeneratedBy	WasGeneratedBy
	WasDerivedFrom	WasDerivedFrom
	WasControlledBy	WasExecutedBy
	WasTriggeredBy	WasInformedBy

to model parties responsible for activities. Additionally, PROV includes seven types of edges: used (some entity used by an activity), wasAssociatedWith (an agent was engaged in some activity), wasGeneratedBy (an entity generated by an activity), wasDerivedFrom (an entity derived from another entity), wasAttributedTo (an agent used an entity), actedOnBehalfOf (an agent acted on behalf of another agent) and wasInformedBy (an activity sent its result data to another activity).

Table 2.2 explains the mapping from a PROV graph to a SimP graph. Besides the straightforward mapping provided in Table 2.2, we need to consider the following:

- When converting an activity in PROV to SimP, we create a process and operation and link them by a WasPartOf edge. Mapping a PROV process into a SimP operation assumes the relations connected with PROV process are mapped to relations with SimP operation.

- As in OPM, PROV does not support an articulated specification of different types of relations between activities. When converting, we map the PROV

Table 2.2.: Mapping From PROV To SimP

	PROV	SimP
Nodes	Activity	Process, Operation, WasPartOf
	Entity	Data
	Agent	Actor
Edges	Used	Used
	WasGeneratedBy	WasGeneratedBy
	WasDerivedFrom	WasDerivedFrom
	WasAssociatedWith	WasExecutedBy
	WasInformedBy	WasInformedBy
	WasAttributedTo	WasAttributedTo
	ActedOnBehalfOf	ActedOnBehalfOf

WasInformedBy to SimP WasInformedBy by assuming PROV activities are related with sequential communication relation.

By following the mapping ontology described earlier, we implemented a conversion tool that facilitates the conversion from the standard models (OPM, and PROV) to SimP model. The input of the tool is an XML -formatted file containing data provenance encoded according to the OPM model or the PROV model. The conversion tool stores the converted provenance data into our provenance storage, that is, MySQL or Neo4J.

2.3 Provenance Query Language

Our framework provides a unified provenance query language independent of the provenance representation (relational or graph-based) and is referred to as QL-SimP.

The language provides expressions and constructs which facilitate the specifications of SimP queries. Using QL-SimP, we propose a set of provenance queries.

2.3.1 QL-SimP Grammar

QL-SimP is derived from the Versioning Query Language (VQuel) [50]. Similar to VQuel, a QL-SimP query has two main clauses: iterator and retrieval.

- Iterators specify the source entity from which results are fetched. The iterator clause has the following format: RangeOf $\langle iterator - variable \rangle$ is $\langle entity \rangle$.

- The retrieval clause is used to select entity attributes. The retrieval might be associated with a predicate, sorting, limit, or group-by clause. The predicate is used to specify a condition which should be satisfied by the retrieved entity records. The sorting clause re-orders the results based on a certain attribute(s) either ascending or descending. Furthermore, the limit clause constraint the number of retrieved records. QL-SimP also supports the $Group - by$ clause to group the result based on a set of attributes. Besides that, it has aggregate functions (i.e., sum, avg, $count$, min, and max) which are usually associated with the $Group - by$ clause. Subsequently, the retrieval clause is formatted as shown in definition 2.3.1.

  Definition 2.3.1 (Retrieval Clause:) *The retrieval clause has the following format* $retrieve[\langle iterator - list \rangle].\langle attribute - list \rangle$ $[where \langle predicate \rangle]$ $[Group - by \langle attribute - list \rangle]$ $[sort - by \langle attribute(s) \rangle [asc/desc]]$ $[limit \langle positive - integer \rangle]$

For referencing attributes, QL-SimP uses the iterator reference (e.g., Process.description). The keyword *all* is also used to retrieve all attributes from the corresponding entity (e.g., $Process.all$). QL-SimP supports different types of relational comparators (i.e., $=$, $\neq$, $<$, $\leq$, $>$, and $\geq$) and logical connectives (i.e., *and*,

or, and *not*). Combining similar structured results from two different queries is supported by the *Union*, *UnionAll*, and *Intersect* clauses. Both *Union* and *UnionAll* are used to merge the result between two queries; *Union* returns distinct values while *UnionAll* allows duplicate values. *Intersect* returns the common results between two queries. The following example shows a query which combines the results retrieved from two different entities which have the description attribute:

Query Example:

RangeOf *P is Process* ***retrieve*** *P.description*

Union — Union All — Intersect

RangeOf *O is Operation* ***retrieve*** *O.description*

QL-SimP has four constructs (*Ancestor*, *Successor*, *Neighbor*, and *Origin*) to facilitate traversing the entities which have recursive relations (e.g., WasDerivedFrom relates Data entities or WasForkedBy relates Process entities).

- $\langle iterator \rangle$.**Ancestor**($\langle integer \rangle | \langle min, max \rangle [, \langle relation - type \rangle]$): It is used to return a set of ancestors of the specified $\langle Iterator \rangle$ until a number of levels in the provenance graph is reached. The number of levels can be specified by a number or range. The $relation - type$ is an optional input which specifies the relation connecting an iterator with its ancestor.

- $\langle iterator \rangle$.**Successor**($\langle integer \rangle | \langle min, max \rangle [, \langle relation - type \rangle]$): It is similar to the Ancestor construct, but it returns the successors for a certain iterator.

- $\langle iterator \rangle$.**Neighbor**($\langle integer \rangle | \langle min, max \rangle [, \langle relation - type \rangle]$): It is a relaxed version of the Ancestor and Successor constructs. The distance between two instances of an iterator is an integer number of levels regardless of the traversal graph direction (i.e., forward or backward).

- $\langle iterator \rangle$.**Origin**($[relation - type]$): It is similar to the Ancestor construct without any limitation on the number of ancestor levels for returning the origin(s) of a certain iterator.

In addition to the recursive-relation traversal constructs, the *Relation* construct enables traversing non-recursive relations on the provenance graph. It relates two iterator variables with a certain relation. It may also contain a predicate which is a short-hand form of the *Where* clause. It has the form:

$Relation(source - iterator - variable,\ destination - iterator - variable,\ type\ [,\ predicate])$.

Representing a pattern-based query by using the simple clauses introduced so far is challenging. So we include in our provenance language an additional construct to extract a subgraph using a pattern. The pattern is represented by a sequence of relations. Every two consecutive relations must have a common iterator. The output of the construct is the source iterator of the first relation. The pattern-based subgraph has the form:

sub-graph$(\langle relation - sequence \rangle)$.

The following example shows a pattern composed of three relations. Every relation might contain a predicate to reduce the size of the extracted sub-graph.

$sub - graph(Relation(I_1\ \text{is}\ entity_1,\ I_2\ \text{is}\ entity_2,\ edge - type),\ Relation(I_2\ \text{is}\ entity_2,\ I_3\ \text{is}\ entity_3,\ edge - type),\ Relation(I_4\ \text{is}\ entity_4,\ I_3\ \text{is}\ entity_3,\ edge - type))$.

QL-SimP supports another construct which checks the connectivity between two entities in the provenance graph without providing a pattern or a relation through the *Reachable* construct which has the form:

Reachable$(source - iterator - variable, destination - iterator - variable)$

It returns true if the provenance graph contains a path connecting the two iterators; otherwise, it returns false. The reachable construct corresponds to a reachability query [51].

Besides that, a provenance query might require finding the minimum distance between two iterator variables. The $Minimum - Distanceconstruct$, which supports this functionality, follows the form:

$Minimum - Distance(source - iterator - variable,\ destination - iterator - variable[,\ predicate])$.

Table 2.3.: Attribute-based Query Examples

$aQ-P$	**RangeOf** P is Process **Retrieve** P.all **Where** P.domain = "*Workflow*"
$aQ-O$	**RangeOf** P is Process, O is Operation **Retrieve** O.all **Where** Relation(O,P,wasPartOf) and P.description = "*Create Workflow*"
$aQ-A1$	**RangeOf** A is Actor **Retrieve** A.all **Where** A.domain = "*Workflow*"
$aQ-A2$	**RangeOf** P is Process, A is Actor **Retrieve** P. description **Where** Relation(P,A,wasExecutedBy) and A.description = "*Peter*" **Union** **RangeOf** D is Data, A is Actor **Retrieve** D. description **Where** Relation(D,A,wasAttributedTo) and A.description = "*Peter*"

It returns a distance which represents the minimum number of relations relating two iterator variables. It may also contain a predicate which is a short-hand form of the $Where$ clause.

2.3.2 Provenance Queries

Attribute-based Queries: In these queries, the main entities of the provenance model are retrieved by applying filters on their attributes. Examples of such queries (shown in Table 2.3) are: find processes by domain (this query is referenced as $aQ-P$), find all the operations belonging to a process (referenced as $aQ-O$); find actors by domain (referenced as $aQ-A1$); and find a list of processes and data for a certain actor (referenced as $aQ-A2$).

Invocation-based Queries: These queries retrieve the set of data objects manipulated by a selected process or operation. Examples of these queries (shown in Table 2.4) are to search for data objects generated by processes that satisfy certain criteria (e.g. process domain, actor or time) (referenced as $iQ-Out-P$) and to find data objects which were input for such processes (referenced as $iQ-In-P$). These

Table 2.4.: Invocation-based Query Examples

$iQ{-}Out{-}P$	**RangeOf** P is Process, D is Data **Retrieve** D.all **Where** Relation(D, P, wasGeneratedBy) and P.domain = "*Workflow*" and P.startTimestamp = "$2015-09-02$"
$iQ-In-P$	**RangeOf** P is Process, D is Data, A is Actor **Retrieve** D.all **Where** Relation(P, D, used) and Relation(P, A, wasExecutedBy) and P.domain = "*Workflow*"
$iQ{-}Out{-}O$	**RangeOf** P is Process, O is Operation, D is Data **Retrieve** O.Description, D.all **Where** Relation(O, P, wasPartOf), Relation(D, O, wasGeneratedBy) and P.description = "*Create Workflow*" and P.startTimestamp = "$2015-09-02$"
$iQ-In-O$	**RangeOf** P is Process, O is Operation, D is Data **Retrieve** O.Description, D.all **Where** Relation(O, P, wasPartOf), Relation(D, O, used) and P.description = "*Create Workflow*" and P.startTimestamp = "$2015-09-02$"

queries facilitate using provenance for reproducing a data object, tracking system changes, etc.

Lineage-based Queries: Data object dependencies are traversed either backward or forward. With the backward traversal, we retrieve the historical dependency of a data object while with the forward traversal we trace the usages of a data object. An example of the backward data dependency query is to find the ancestor's data objects to data d (referenced as $lQ-S-A$ and $lQ-S-O$ in Table 2.5), and an example of the forward data dependency query is to find the successors of the data object (referenced as $lQ-S-S$ in Table 2.5). The combined results of forward and backward data dependency queries can be retrieved using the Neighbor construct (referenced as $lQ-S-N$ in Table 2.5). Furthermore, an example of finding the

Table 2.5.: Lineage-based Query Examples

$lQ-S-A$	**RangeOf** D is Data, D_3 is D.Ancestor(3) **Retrieve** D_3.all **Where** D.description = "*Anatomy image*"
$lQ-S-S$	**RangeOf** D is Data, S_3 is D.Successor(3) **Retrieve** S_3.all **Where** D.description = "*Anatomy image*"
$lQ-S-N$	**RangeOf** D is Data, N_3 is D.Neighbor(3) **Retrieve** N_3.all **Where** D.description = "*Anatomy image*"
$lQ-S-O$	**RangeOf** D is Data, O is D.Origin() **Retrieve** O.all **Where** D.description = "*Anatomy image*"
$lQ-S-D$	**RangeOf** D_1 is Data, D_2 is Data **Retrieve** Minimum-Distance(D_1, D_2) **Where** D_1.description = "*Anatomy image*" and D_2.description = "*Atlas image*"
$lQ-P-S1$	**RangeOf** D is Data, G_O is sub-graph(¡Relation(O_1 is Operation, P_1 is Process, wasPartOf), Relation(P_1 is Process, P_2 is Process, wasForkedBy), Relation(O_2 is Operation, P_2 is Process, wasPartOf, O_2.description = "*execute experiment*")) **Retrieve** D.all **Where** Relation(D, G_O, wasGeneratedBy)
$lQ-P-S2$	**RangeOf** O is Operation, D is Data, G_O is sub-graph(¡Relation(O_1 is Operation, P_1 is Process, wasPartOf), Relation(P_1 is Process, P_2 is Process, wasForkedBy), Relation(O_2 is Operation, P_2 is Process, wasPartOf, O_2.description = "*execute experiment*")) **Retrieve** O.all **Where** Relation(O, D, used) and Relation(D, G_O, wasGeneratedBy)

minimum number of changes on data d_i to be transformed into another form d_j is shown in example $lQ-S-D$ in Table 2.5. These types of query can utilize the lineage entity in our provenance model.

In addition, data object dependency queries (i.e., lineage queries) may include patterns, so we refer to them as pattern-based lineage queries. The traversal of a data object dependency graph involves not only utilizing lineage entities but also other entities such as processes, operations, and communications. The basic approach for traversal is to match a set of relations involving fundamental provenance entities. An example of pattern-based forward data dependency query is the query that finds data objects produced by a specific flow pattern (referenced as $lQ-P-S1$ in Table 2.5). An example of the backward data dependency query is the query that finds the data objects used in a specific flow pattern. Another complex example of the forward data dependency query is the query that finds all operations which consumed data objects that have been generated by a certain flow pattern (referenced as $lQ-P-2$ in Table 2.5). The flow pattern used in the examples $lQ-P-S1$ and $lQ-P-S2$ is "*operations are contained in a process which was formed by another process, and the later process contains a certain operation.*".

Communication-based Queries: These queries retrieve a set of operations /processes that communicated with other operations/processes according to certain types of communications such as sequential or composition. An example is a query that finds a list of operations/processes which communicate with a certain process by a composition relation. This type of queries may also be aggregated such as a query finding the top N processes communicating together within the same environment in a certain domain (referenced as $cQ-A-Intra-P$ in Table 2.6) or finding the top N processes communicating together between two different environments in a certain domain (referenced as $cQ-A-Inter-P$ in Table 2.6). This type of queries utilizes the communication entity primarily.

Access-based Queries: The main purpose of these queries is to audit access policies for users (i.e., actors) and how the policies can evolve in the future. These queries can also be used to detect manipulations that did not follow the security access policies. Examples of these queries include: find access policies for a certain actor in a specific timeframe (referenced as $acQ-P$ in Table 2.7); find operations

Table 2.6.: Communication-based Query Examples

$cQ-A-Intra-P$	**RangeOf** P_1 is Process, P_2 is Process, O_1 is Operation, O_2 is Operation, E is Environment **Retrieve** P_1. description, P_2. description, count(*) **Where** P_1.description $\neq$ P_2. Description and Relation(O_1,P_1,wasPartOf) and Relation(O_2,P_2,wasPartOf) and Relation(O_2,O_1,wasInformedBy—wasEncapsulatedBy) and Relation(P_1,E,wasInContext) and Relation(P_2,E,wasInContext) and P_1.domain = "*workflow domain*" and P_2.domain = "*workflow domain*" **Group-by** P_1. description, P_2. description **Sort-by** count(*) desc **Limit** n
$cQ-A-Inter-P$	**RangeOf** P_1 is Process, P_2 is Process, O_1 is Operation, O_2 is Operation, E_1 is Environment, E_2 is Environment **Retrieve** P_1. description, P_2. description, count(*) **Where** P_1.description != P_2. Description and Relation(O_1,P_1,wasPartOf) and Relation(O_2,P_2,wasPartOf) and Relation(O_2,O_1,wasInformedBy—wasEncapsulatedBy) and P_1.domain = "*workflow domain*" and P_2.domain = "*workflow domain*" and Relation(P_1,E_1,wasInContext) and Relation(P_2,E_2,wasInContext) and E_1.description $\neq$ $E-2$.description *and* P_1.domain = "*workflow domain*" and $P-2$.domain = "*workflow domain*" **Group-by** P_1. description, P_2. Description **Sort-by** count(*) desc **Limit** n

Table 2.7.: Access-based Query Examples

$acQ-P$	**RangeOf** A is Actor, P is Process, AC is AccessPolicy **Retrieve** AC.all **Where** Relation(AC, A, wasGrantedTo) and A.description = "*Peter*" and AC.Subject = "*Process*" and AC.Condition = P.description and P.startTimestamp $\geq$ "$2015-09-02$" and P.endTimestamp $\leq$ "$2015-10-02$"
$acQ-O$	**RangeOf** O is Operation, A is Actor, P is Process, AC is AccessPolicy **Retrieve** O.all **Where** Relation(O, P, wasPartOf) and Relation(P, A, wasExecutedBy) and Not(Relation(AC, A, wasGrantedTo)) and AC.Subject = "*Process*" and AC.Condition = P.description
$acQ-H$	**RangeOf** P is Process, AC is AccessPolicy, A is Actor **Retrieve** AC.Subject, AC.Condition, AC.Effect, AC.Obligations **Where** AC.Subject = "*Process*" and AC.Condition = P.description and Relation(P, A, wasExecutedBy) and P.description = "*Create Workflow*" and A.description = "*Peter*" **Group-by** AC.Subject, AC.Condition, AC.Effect, AC.Obligations

which were actuated by actors who are not privileged to execute them (referenced as $acQ-O$ in Table 2.7); and find how a policy evolved for a certain process and actor (referenced as $acQ-H$ in Table 2.7).

Workflow-based Queries: The main purpose of these queries is to utilize provenance data for retrieving the various versions of a workflow executed within a time window. The applications of these queries include: analyzing a workflow evolution over time through comparing the historical versions of the workflow; comparing the behavior of different users when executing a workflow; and generating an abstract description of a workflow across several executed versions by different users. Examples of these queries include: find the list of operations that compose the tasks of a specific workflow in a certain period (referenced as $wQ-OT$ in Table 2.8); find the list of data objects that were part of (either manipulated or produced by) the tasks of a specific workflow in a certain period (referenced as $wQ-DT$ in Table 2.8); find the operations that compose the tasks of a specific workflow executed multiple times by a certain actor (referenced as $wQ-OA$ in Table 2.8); find the list of data objects that were involved in the tasks of a specific workflow executed multiple times by a certain actor (referenced as $wQ-DA$ in Table 2.8); and find the list of common operations in a workflow executed by two different actors (referenced as $wQ-IA$).

Transaction-based Queries: In general, at both application and operating system level a transaction can be defined abstractly by three attributes: action, object, and user. The transaction attributes can be extracted from our provenance model based on the mapping ontology in Table 2.9. When mapping the used or $wasExecutedBy$ edges, we map the edge to the object which is the destination of the edge. Transaction-based queries enable the retrieval of executed transactions in the system supported by our provenance model. From a security perspective, these types of query allow one to audit and monitor the system of interest at the transaction level. Examples of these queries include: find the transactions executed by a certain user (referenced as $tQ-A$ in Table 2.10); and find the transactions that accessed specific objects (referenced as $tQ-D$ in Table 2.10).

Table 2.8.: Workflow-based Query Examples

$wQ-OT$	**RangeOf** P is Process, O is Operation, E is Environment **Retrieve** O.description, P.description, E.description, O.domain **Where** P.workflowID = "20" and P.startTimestamp $\geq$ "$2015-09-02$" and P.endTimestamp $\leq$ "$2015-10-02$" and Relation(O, P, wasPartOf) and Relation(P, E, inContextOf) **Sort-by** O.timestamp
$wQ-OA$	**RangeOf** P is Process, O is Operation, A is Actor, E is Environment **Retrieve** O.description, P.description, E.description, O.domain, A.description **Where** P.workflowID = "20" and Relation(P, A, WasExecutedBy) and Relation(O, P, wasPartOf) and Relation(P, E, inContextOf) and A.description = "*Peter*" **Sort-by** O.timestamp
$wQ-DT$	**RangeOf** P is Process, D is Data **Retrieve** D.description, P.description **Where** P.workflowID = "20" and P.startTimestamp $\geq$ "$2015-09-02$" and P.endTimestamp $\leq$ "$2015-10-02$" and Relation(D, P, wasGeneratedBy) **Sort-by** P.startTimestamp **Union RangeOf** P is Process, D is Data **Retrieve** D.description, P.description **Where** P.workflowID = "20" and P.startTimestamp $\geq$ "$2015-09-02$" and P.endTimestamp $\leq$ "$2015-10-02$" and Relation(P, D, used) **Sort-by** P.startTimestamp
$wQ-DA$	**RangeOf** P is Process, D is Data, A is Actor **Retrieve** D.description, P.description **Where** P.workflowID = "20" and Relation(P, A, WasExecutedBy) and Relation(D, P, wasGeneratedBy) and A.description = "*Peter*" **Sort-by** O.timestamp **Union RangeOf** P is Process, D is Data, A is Actor **Retrieve** D.description, P.description **Where** P.workflowID = "20" and Relation(P, A, WasExecutedBy) and Relation(P, D, used) and A.description = "*Peter*" **Sort-by** O.timestamp
$wQ-IA$	**RangeOf** P is Process, O is Operation, A is Actor **Retrieve** O.description, P.description **Where** P.workflowID = "20" and Relation(P, A, WasExecutedBy) and Relation(O, P, wasPartOf) and A.description = "*Alice*" **IntersectRangeOf** P is Process, O is Operation, A is Actor **Retrieve** O.description, P.description **Where** P.workflowID = "20" and Relation(P, A, WasExecutedBy) and Relation(O, P, wasPartOf) and A.description = "*Peter*"

Table 2.9.: Mapping SimP Model to a Transaction

SimP Entity/Edge	Transaction Attribute
Process	Action
Operation	Action
used	Object
wasExecutedBy	User

Table 2.10.: Transaction-based Query Examples

$tQ - A$	**RangeOf** A is Actor, P is Process, D is Data **Retrieve** P.all **Where** Relation(P, A, wasExecutedBy) and Relation(P, D, used) and A.description = "*Alice*" **union RangeOf** O is Operation, A is Actor, P is Process, D is Data **Retrieve** O.all **Where** Relation(O, P, wasPartOf) and Relation(P, A, wasExecutedBy) and Relation(P, D, used) and A.description = "*Alice*"
$tQ - D$	**RangeOf** A is Actor, P is Process, D_IN is Data, D_OUT is Data **Retrieve** O.all **Where** Relation(P, A, wasExecutedBy) and Relation(D_OUT, P, wasGeneratedBy) and Relation(P, D_IN, used) and D_IN.description = "*Anatomy image*" **Union RangeOf** O is Operation, A is Actor, P is Process, D_IN is Data, D_OUT is Data **Retrieve** O.all **Where** Relation(O, P, wasPartOf) and Relation(P, A, wasExecutedBy) and Relation(D_OUT, O, wasGeneratedBy) and Relation(O, D_IN, used) and D_In.description = "*Anatomy image*"

Table 2.11.: Statistical-based Query Examples

$sQ{-}OT$	**RangeOf** O is Operation, P is Process **Retrieve** P.description, count(O.description) **Where** Relation(O, P, wasPartOf) and P.startTimestamp $\geq$ "$2015-09-02$" and P.endTimestamp $\leq$ "$2015-10-02$" **Group-by** P.description
$sQ{-}AT$	**RangeOf** A is Actor, P is Process **Retrieve** P.description, count(A.description) **Where** Relation(P, A, wasExecutedBy) and P.startTimestamp $\geq$ "$2015-09-02$" and P.endTimestamp $\leq$ "$2015-10-02$" **Group-by** P.Description
$sQ{-}AR$	**RangeOf** A is Actor, P is Process **Retrieve** P.description, count(A.description) **Where** Relation(P, A, wasExecutedBy) and A.Role = "*Accountant*" **Group-by** P.Description
$sQ{-}OR$	**RangeOf** O is Operation, P is Process **Retrieve** P.description, count(O.description) **Where** Relation(O, P, wasPartOf) and A.Role = "*Accountant*" **Group-by** P.description

Statistical Queries: These queries allow one to obtain statistical information about provenance data. These queries provide a tool to analyze the system usage and also support detecting abnormal behaviors by analyzing the temporal usage of system resources by different actors. Examples of these queries include: find how many operations were executed in each executed process within a certain period (referenced as $sQ-OT$ in Table 2.11); find how many actors executed each process within a certain period (referenced as $sQ-AT$ in Table 2.11); find how many operations in each process were executed by actors having a certain role (referenced as $sQ-OR$ in Table 2.11); and find how many different actors executed each process where the actors having a certain role (referenced as $sQ-AR$ in Table 2.11).

All of the above types of queries (i.e., attribute-based, invocation-based, lineage-based, communication-based, access-based, workflow-based, transaction-based, and statistical) can help in detecting anomalies by comparing the expected output in the recorded data provenance with the current execution. Once an anomaly behavior is detected, we can identify actors who have such an anomalous behavior.

Because provenance storage grows fast, there is a need to reduce the size of the provenance graph before executing a query. Our model thus includes some attributes which enable creating provenance views or abstractions which are enough to execute different queries. For example, the domain attribute helps in creating a provenance graph in a certain domain view for executing a query efficiently. Also, the timestamp attributes create a time-sensitive provenance graph. For example, instead of querying how processes and actors modified a document two years ago, we can only consider a subset of the provenance graph in the time frame of interest.

We implemented all these types of provenance queries using our query language and integrated them as a parallel component in the SimP framework. The query component supports viewing the results in tabular and graph format. To view the query result in a visualized graph, we used Graphviz [52] API which is an open source software created by AT&T Research. The API uses a graph description language called DOT.

2.4 Security

Due to the sensitivity of data provenance information collected and stored in provenance storage, security is an essential factor and requirement. Hence, we incorporated an additional entity to specify the privileges granted to actors for accessing the provenance storage and executing provenance queries. Such an entity, referred to as Provenance Query Authorization, records information about which actors have which authorizations. The authorization is expressed as subject, condition, and effect fields. The subject refers to the type of provenance storage entity (e.g., Processes

entity) while the condition field identifies the provenance entity (e.g., Process Description value). The effect field indicates the authorization status (e.g., granted or revoked).

In conclusion, the security requirement in our framework is addressed by the following features: a) access control policy which is a main entity in the SimP model, and b) provenance query authorization to secure the access to the sensitive information in provenance storage.

2.5 Granularity

Another main factor in our provenance framework is the ability to specify and modify the granularity level needed to capture provenance for specific records or entities. For this purpose, in our provenance database, we include an additional entity, referred to as *Granularity Policy*, which is not part of the provenance model, but it is part of our provenance framework. A granularity policy enables users to specify the desired level of provenance details to be captured and stored (e.g., capturing provenance at the activity level in scientific provenance workflow, or the OS level to capture system configurations). A Granularity Policy record includes the following fields: granularity policy ID, actor ID, subject, condition, granularity type. The subject may refer to the targeted process, operations, or communications, whereas the condition refers to the identifier of the subject. The granularity type is the detail level of the required provenance.

In conclusion, the requirement of multi-granularity in our framework is addressed by the following features: a) a provenance model able to represent provenance metadata for data objects at various granularity levels; b) support for the integration with systems utilizing different provenance capturing mechanisms (i.e., provenance for workflow data or file system); and c) support for granularity preferences based on granularity policies.

2.6 Integration with the CRIS System

We integrated our provenance model in the *Computational Research Infrastructure for Science* (CRIS) [31]. CRIS is a scientific data management workflow cyberinfrastructure for scientists lacking extensive computational expertise. The application is currently used by a community of users in Agronomy, Biochemistry, Bioinformatics, and Health Care Engineering at Purdue University. Previously, CRIS had its own provenance model mainly based on versioning mechanism. Such a mechanism is not able to capture provenance at different granularities since it is data-centric. Our model is data and operational centric in that we maintain metadata about the derivation of every data object (i.e., what is the original source of a data object and what is the deriving operation?).

Within the CRIS system, we have a provenance logging component based on aspect oriented programming to instrument the application code. The logging component collects a set of appropriate provenance logs while the CRIS is running. The logs use an XML-based representation of provenance records to facilitate parsing them into SimP model. Periodically, the CRIS provenance logs are fetched and converted into another XML-format file. The new XML file contains a data dependency provenance graph following the SimP representation ontology.

2.7 Experimental Results

The main purpose of the experiment part is to evaluate the provenance model in its two representations, relational and graph-based, using the query component.

2.7.1 Experimental Methodology

To evaluate the performance of provenance queries when executed on the SimP relational and graph databases, we conducted several experiments using three provenance datasets; one is a real dataset (referred to as REAL) extracted from the CRIS

Table 2.12.: Provenance Dataset size

	Actor	Process	Operation	Data	Lineage	Communication
REAL	10	110	997	12.3K	59.8K	936
SYN_1	60	4.8K	21.4K	125.6K	289.7K	12.5K
SYN_2	150	7.2K	81.6K	357.2K	826.6K	50.3K

testing environment, and the other two are synthetic datasets (referred to as SYN_1 and SYN $_2$) generated by a random dataset generator to scale the size of the provenance experimental dataset. Table 2.12 shows the size of each dataset in terms of the main entities.

For each query type, we ran multiple parameterized queries and averaged their execution time. All times are in milliseconds. To avoid the caching effect on timing, each query was executed 5 times and the longest and shortest times were ignored. In each database (MySQL or Neo4J), we built a set of indexes to enhance query performance; for Neo4J we created schema indexes while for MySQL we created b-tree indexes on proper fields for speeding up queries. To avoid any latency posed by database driver or middleware existing between our query component and database engine, all queries were executed natively on the database engine. All experiments were performed on a 3.6 GHz Intel Core $i7$ machine with 12 GB memory running on 64 bit Windows 7. In our provenance framework, we used MySQL 5.7 and Neo4J 2.3.2.

2.7.2 Provenance Query Evaluation Results

Fig. 2.4 shows the execution time for six types of queries (**attribute-based**, **invocation-based**, **communication-based**, **workflow-based**, **transaction-based**, and **statistical**) using the REAL, SYN_1 and SYN_2 datasets. In general,

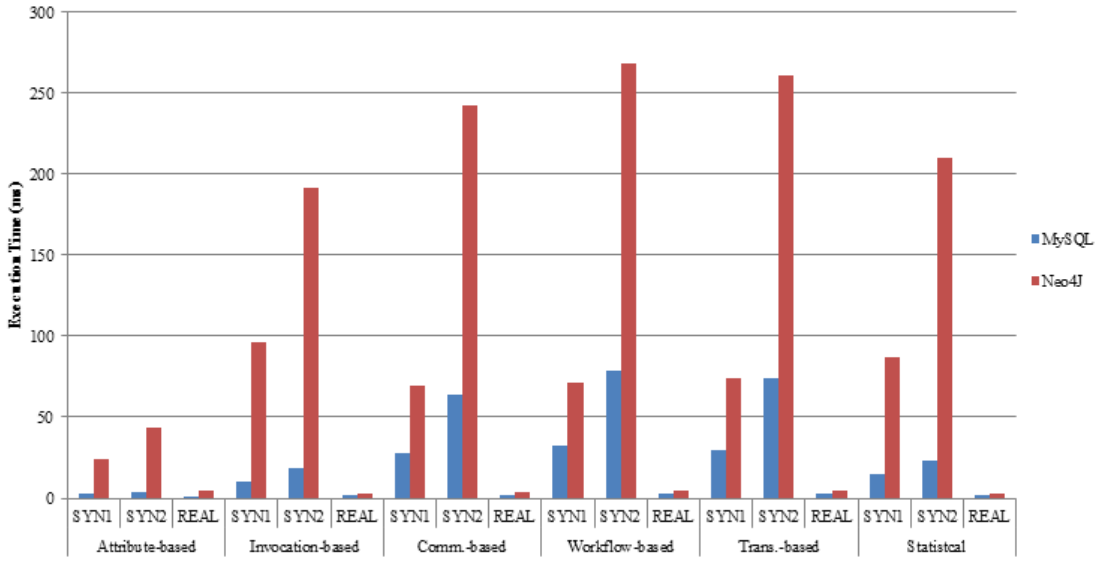


Fig. 2.4.: Average Execution Time for Attribute-based, Invocation-based, Communication-based, Workflow-based, Transaction-based, and Statistical Queries with the SYN_1, SYN_2, and REAL datasets

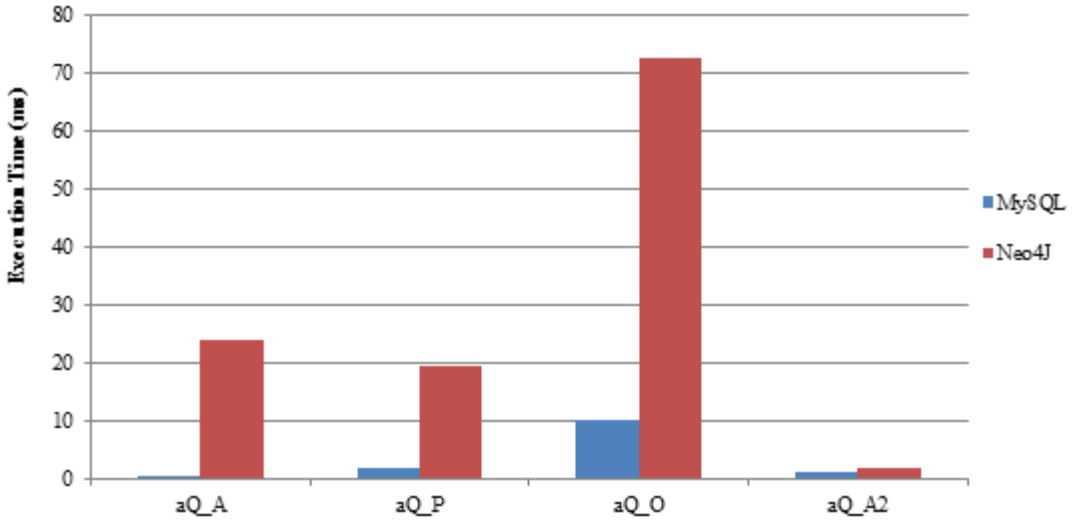


Fig. 2.5.: Execution Time for A Sample of Attribute-based Queries in the SYN_2 Dataset

MySQL outperformed Neo4J, especially with large datasets. The queries used in this experiment are structural queries which require scanning data for a specific attribute value, joining data, or aggregating data. In the SYN_2 dataset, the speedup factors of **attribute-based**, **invocation-based**, **communication-based**, **workflow-based**, **transaction-based**, and **statistical** queries were 9.3, 10.2, 3.8, 3.4, 3.5, and 8.9 respectively. The speedup of **communication-based**, **workflow-based** and **transaction-based** queries was low relatively to the other queries because they require grouping and joining large amounts of data. Fig. 2.5 shows the execution of a sample set of **attribute-based** queries on the SYN_2 dataset ($aQ-A$, $aQ-P$, $aQ-O$, and $aQ-A2$). MySQL achieved the highest speedup with a factor of 53.8 in $aQ-A$ while the lowest speedup was in $aQ-A2$ with a factor of 1.6. $Q-A2$ is based on the Union construct where the relational and graph-based databases have almost the same execution plan. Thus there is no valuable speedup. Furthermore, Fig. 2.6 shows the execution of a sample set of **workflow-based** queries on the SYN_2 dataset ($wQ-OT$, $wQ-OA$, $wQ-DT$, $wQ-DA$, and $wQ-IA$). MySQL achieved the highest speedup with a factor of 8.1 in $wQ-OT$ while the lowest speedup was in $wQ-DT$ with a factor of 3.1. **Lineage-based** queries show the strength of Neo4J. Fig. 2.7 shows the execution time in base-10 logarithmic scale. This type of queries requires provenance graph traversals along certain paths to fetch data. Since Neo4J stores provenance data as a graph, it is able to perform better than MySQL. With the REAL dataset, the speedup factor of Neo4J was 12.4 while it was 10.2 and 13.1 with the SYN_1 and SYN_2 datasets, respectively. Fig. 2.8 shows the execution of a sample set of **lineage-based** queries in base-10 logarithmic scale ($lQ-S-A$, $lQ-S-S$, $lQ-S-N$, $lQ-S-O$, and $lQ-S-D$) in the SYN_2 dataset. Neo4J achieved the highest speedup with a factor of80 in $lQ-S-D$ while the lowest speedup was in $lQ-S-S$ with a factor of 13.7. Based on Fig. 2.8, the cost of $lQ-S-N$ is the most because it requires bi-directional traversal (i.e., forward and backward).

Access-based queries are another example of structural queries. Also for such queries, the performance of MySQL exceeds Neo4J with a very high speedup factor,

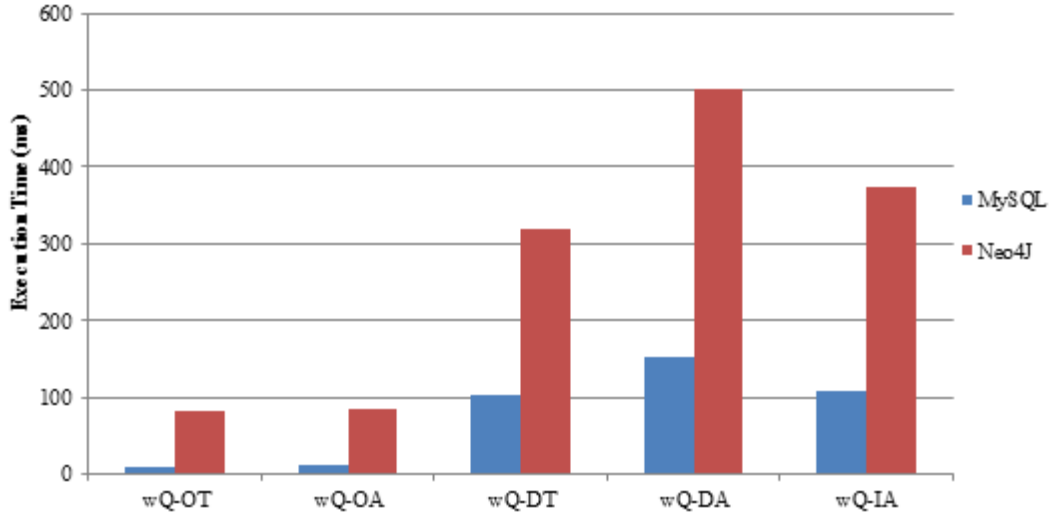


Fig. 2.6.: Execution Time for A Sample of Workflow-based Queries in the SYN_2 Dataset

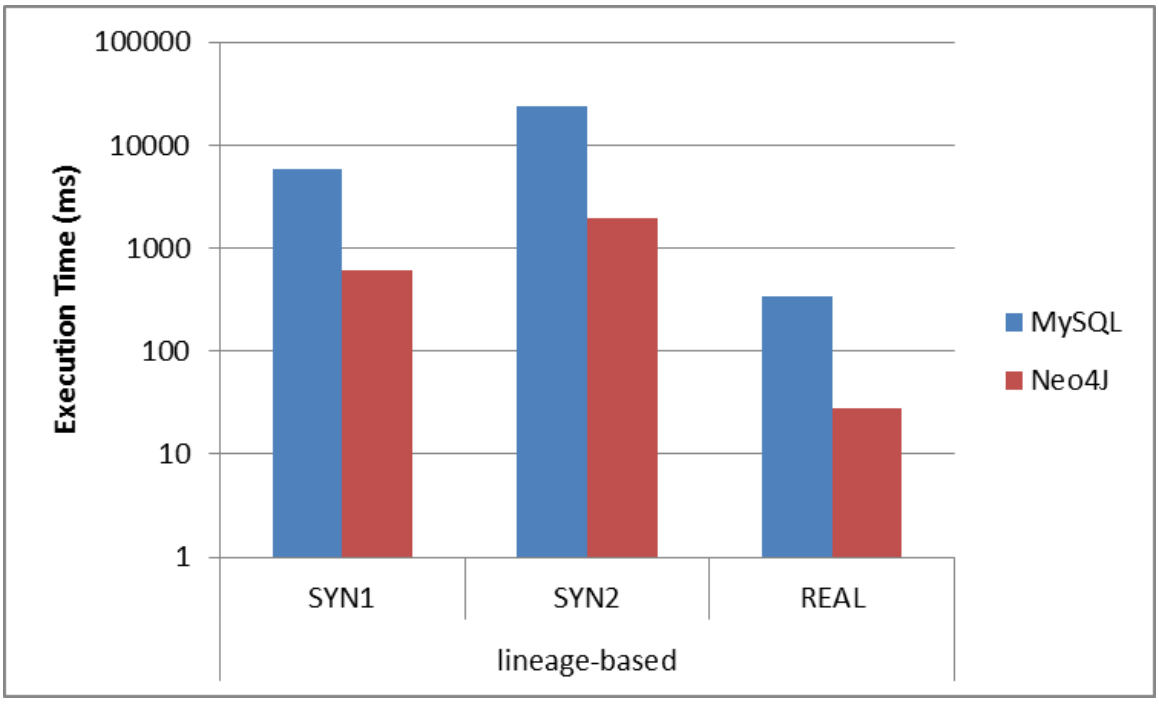


Fig. 2.7.: Average Execution Time of Lineage-based Queries

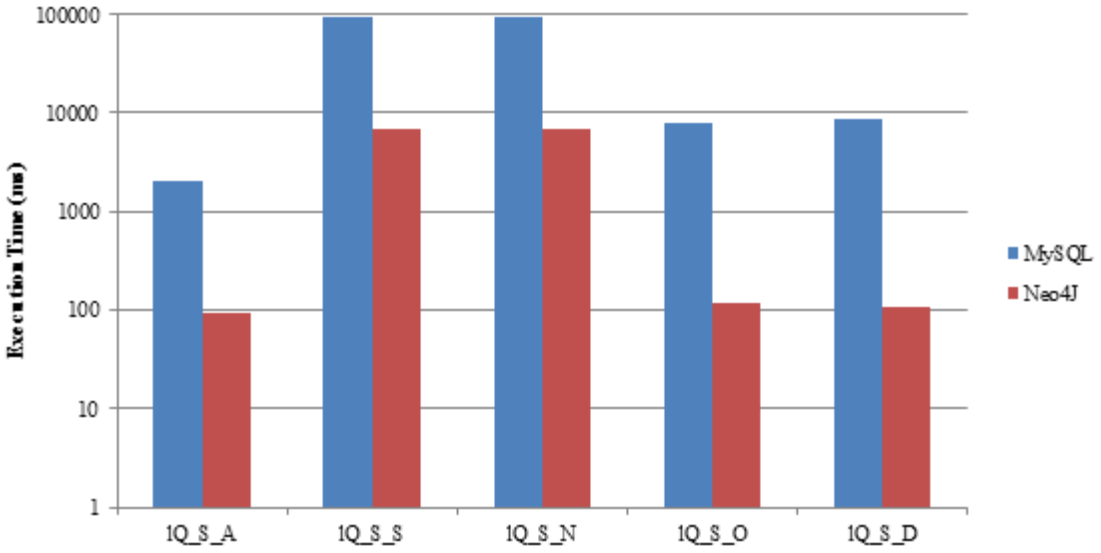


Fig. 2.8.: Execution Time for A sample of Lineage-based Queries with the SYN_2 Dataset

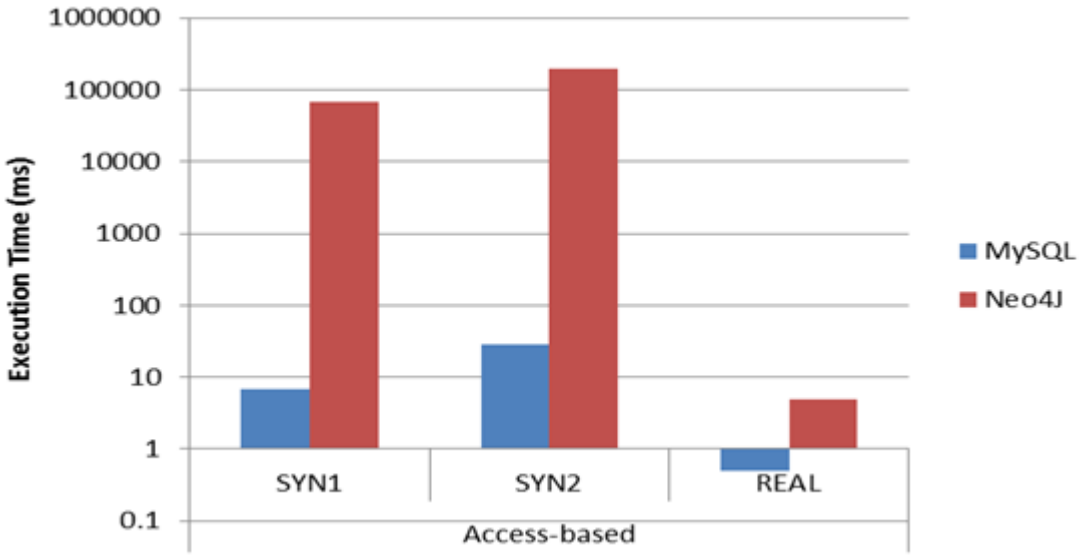


Fig. 2.9.: Average Execution Time for Access-based Queries

especially when dealing with a large dataset. Fig. 2.9 shows the execution time in 10-base logarithmic scale where the speedup factor of MySQL was 9.9 in the REAL dataset.

Based on our evaluation, if the query component is heavily used for lineage queries Neo4J is the best choice. On the other hand, MySQL is much faster if the user queries are of a structural type such as attribute-based, invocation-based and access-based queries.

2.8 Related Work to the SimP Framework

Data provenance has been extensively investigated in e-science systems (surveyed in [53]). Examples of these systems which are data-centric workflow systems include myGrid [9], Chimera [8], and Karma [10]. The provenance model in myGrid focuses on service executions including their duration, data input, and the generated data. Whereas Chimera has its own language (referred to as Virtual Data Language) that considers the relations between activities and resources as the main provenance meta-data. In Karma, the provenance model captures the activities that are executed at the workflow, service and application levels. Furthermore, there are some provenance systems that are devoted to process-centric workflows instead of data-centric as (e.g., PreServ [54]). In PreServ, the provenance system focuses on the relations among processes and data where the relations are categorized as service-service (i.e., source and sink service), data-data (i.e., data derivation), and service-data (a service consumes data, or a service produces data). Other than workflow-based provenance systems, some other systems focus on operating system level (e.g., PASS [11] and ES3 [55]). PASS captures the provenance of the processes that utilize the shared memory and records the data which were used or generated by these processes. ES3 also captures the consumed and produced data at file system level.

On the other hand, OPM [12] and PROV [13] are standard models supporting interoperability and hence are not restricted to a specific system as in the previ-

ously mentioned provenance systems, but can be applied to different systems. PROV provides a toolbox [56] to create PROV graphs in different representations and transforms the graph among these representations; whereas OPM provides a toolbox [57] to create OPM graphs. There are some systems which are based on frameworks compatible with OPM (e.g., SPADE [58]). OPM and PROV are not able to capture access control policies. This limitation addressed by Ni et al. [14] and then by Cadenhead et al. [59]. The provenance models proposed in [14] and [59] support security; however they focus only on operation-based provenance and hence lack multi-granularity support. In [15], Sultana and Bertino proposed an initial comprehensive provenance infrastructure then extended by Abu Jabal and Bertino [16].

Holland et al. [60] discussed the shortcomings of the query languages inherited with three types of data models (i.e., relational, XML, and RDF). They proposed a query language, PQL, to support path queries. The limitation of PQL is an extended form of the Lorel language [61] which is an object-oriented language for semi-structured data. Our provenance query language is composed of general constructs which can be implemented in different data models. Anand et al. [62] proposed a query language which is closed to lineage dependencies. Their query language considers a provenance graph composed of only data nodes and lineage edges. VQuel, by Chavan et al. [50], is a unified query language for versioning systems. We extended it to support our provenance model.

3. PROFACT: A PROVENANCE-BASED ANALYTICS FRAMEWORK FOR ACCESS CONTROL POLICIES

3.1 Preliminaries

In what follows, we introduce background concepts and information needed for the subsequent developments in the discussion.

3.1.1 Role-based Access Control

The role based access control (RBAC model) consists of four basic components [18]: *users*, *roles*, *permissions*, and *sessions*. A role represents an organizational function within a given domain (e.g., a coalition or an enterprise). Roles are granted permissions required for the execution of their functions. A permission consists of the specification of a protected object and an action, defined on the object, and indicates that the action can be executed on the object. Users (e.g., humans, devices) represent the active entities that execute actions on the protected objects. Users are assigned to roles and thus inherit the permissions of their assigned roles. A session is a sequence of accesses executed by a user under one or more roles. When a user becomes active in the system, it establishes a session and, during this session, it uses one or more roles among the ones it has been assigned to.

The RBAC model definition includes several functions. The user assignment (UA) function specifies which user is assigned which roles, whereas the permission assignment (PA) function specifies the set of permissions assigned to each role. The user function maps each session to a single user, while the role function assigns a session to a set of roles (i.e., the roles that are activated by the corresponding user in that

session). The following definition (adapted from Sandhu et al. [18]) formally defines the RBAC model.

Definition 3.1.1 (Role-based Access Control Model [63]) *The model consists of the following components.*

- *U, R, P, S refer to the set of users, roles, permissions, and sessions, respectively.*
- *A permission $p_i \in P$ is a tuple of three components consisting of an object $o_j \in O$, an action $a \in A$, and a sign $g \in \{+, -\}$.*
- *PA is the permission assignment function that assigns permissions to roles (i.e., $PA \subseteq R \times P$ and $PA(r_i) \subseteq P$, $\forall r_i \in R$).*
- *UA is the user assignment function that assigns users to roles (i.e., $UA \subseteq U \times R$ and $UA(u_i) \subseteq R$, $\forall u_i \in U$).*
- *The user function assigns a session to a single user (i.e., $user : S \longrightarrow U \mid user(s_i) \in U$).*
- *The roles function assigns a session to the roles associated with the user activated the corresponding session (i.e., $roles \subseteq S \times 2^R \mid roles(s_i) = \{r \mid (user(s_i), r) \subseteq UA\}$).*
- *RH is the role hierarchy function (i.e., $RH \subseteq R \times R$), which refers to the partially ordered role hierarchy (written $\geq$).*

Notice that in Definition 3.1.1, we associate a "sign" with each permission to support positive and negative authorizations. A positive authorization, denoted by the '+' sign in the permission, is an authorization that allows the role, specified in the authorization, to execute the action on the object specified in the authorization permission. By contrast, a negative authorization, denoted by the '-' sign in the permission, specifies that the role, specified in the authorization, cannot execute the action on the

object specified in the authorization permission. Negative authorizations are particularly useful when dealing with large sets of protected objects organized according to hierarchies. In such contexts, negative authorizations combined with authorization propagation along objects hierarchies support the specification of exceptions, by which one can, for example, allow a role to read an entire directory with the exception of a given file in the directory. Authorization propagation and positive and negative authorizations have been widely investigated [64], and also introduced in access control systems of commercial products. An example is the access control model of SQL Server in which authorizations propagate along securable hierarchies and in which negative authorizations can be specified by means of the DENY authorization command [65]. In our contexts, negative authorizations are also critical in order to provide boundaries to actions that cognitive autonomous devices can execute.

3.1.2 Policy Life Cycle

We assume an iterative policy lifecycle composed of three stages (specification, enforcement, analysis) which form the basis for the deployment of the policies in the system of interest. In the policy specification stage, the administrator coordinates with the representative system users to determine the policies to be enforced. The policy enforcement stage is the one in which policies are applied to control the actions executed by the system users on the protected objects. As the environments we deal with are characterized by dynamic contexts and situations, it is often the case that policies may have to evolve in order to adapt to changes. Therefore, during the policy enforcement stage, additional information is collected that is used by the next stage in the policy lifecycle, that is, the analysis stage. Such a stage evaluates the quality of the current policy set based on the information collected through the enforcement stage and suggests possible changes to the current set of policies. The evolved policies are then deployed and enforced.

3.1.3 Transactions

Policies are mainly important to control transactions which are performed in the system. Each process which is logged using provenance refers to a task in a workflow and each task represents a transaction executed at run-time. We formally define a transaction as follows:

Definition 3.1.2 (Transaction) *A transaction $t \in T$ is an action $a \in A$ executed by a user $u \in U$ on an object $o \in O$. Thus, a transaction t is represented by a tuple (u, o, a).*

3.2 Policy Analysis Metrics And Structures

In this section, we introduce the quality requirements that are used as metrics to evaluate policies through the analysis process. We also describe two data structures that are utilized for the policy analysis.

3.2.1 Policy Quality Requirements

We illustrate the policy quality requirements by the following running example (adapted from [66]).

Example: *We envision an automated delivery management system for army forces operating in a set of bunkers which are supported by a remote supply depot. Autonomous vehicles (mules) are used to transfer supplies (e.g., meals ready to eat (MREs)) from the depot to the bunkers. Supplies in all sites, including the bunkers and the depot, are managed by robotic devices. Smart refrigerators at bunkers manage the MRE inventory and notify the robot at the depot when additional supplies are needed. At the depot, there are several robots and mules. The mules transfer supplies from the depot to bunkers. There are two types of robot. One is the depot manager which receives notifications from smart refrigerators at bunkers and manages the transfer of the required supplies to bunkers; the other type is the worker that is responsible for*

loading the mules with supply cartoons. In addition, there is a computer system that maintains a central database for sharing necessary information (e.g., mule location and status, depot supply, robot status).

In such a delivery management system, we focus on designing an access control system for the robots working at the depot. Because of the two types of robot, we have two corresponding roles: manager and worker. The worker executes the following types of transaction: (i) receive loading requests; (ii) load a mule with the supplies; and (iii) report its status to the computer system. The manager is authorized to perform the same types of transaction as the worker and in addition is authorized to perform the following types of transaction: (i) receive notification from a bunker; (ii) inquire whether a bunker needs supplies; (iii) check availability of supplies; (iv) retrieve the list of available mules and workers; (v) and assign a worker and mule for a delivery request. The access control policies related to these transactions are listed in Tables 3.1 and 3.2.

The problem of assuring the quality of a set of access control policies can be restated as the problem of making sure that policies do not have inconsistency, are not redundant, irrelevant, and incomplete with respect to the actions executed by the users. In addition, it is critical to minimize the number of explicit exceptions that must be allowed with respect to the policies. Minimizing the exceptions is critical to reduce the manual administrative activities to be executed in the system. In what follows we introduce several definitions underlying our policy quality notion.

Definition 3.2.1 (Inconsistency) *Access control policies* acp_i, $acp_j \in ACP$ *are inconsistent if and only if*

- $acp_i.r = acp_j.r \wedge acp_i.p.o = acp_j.p.o \wedge acp_i.p.a = acp_j.p.a$
- $acp_i.p.sign \neq acp_j.p.sign.$

Inconsistency refers to the situation in which for the same access by the same role, one policy allows the access and the other denies it. Policy inconsistency leads to conflicts

Table 3.1.: Example of Access Control Policies for the Depot Manager Role for the Robots Working in a Delivery Management System

Policy	Permission		
	Action	Object	Sign
acp_1	Receive	Notification from bunkers	+
acp_2	Receive	Notification from bunker 10	−
acp_3	Receive	Notification from bunker 5	+
acp_4	Inquire a Bunker	Bunker Status	+
acp_5	Inquire central DB	Supply Status	+
acp_6	Inquire central DB	List of available mules	+
acp_7	Inquire central DB	List of available workers	+
acp_8	Assign loading task	Worker, Mule, Supply	+
acp_9	Receive	Loading task	+
acp_{10}	Load	Supply, mule	+
acp_{11}	Report to central DB	Robot status	+

Table 3.2.: Example of Access Control Policies for the Depot Worker Role for the Robots Working in a Delivery Management System

Policy	Permission		
	Action	Object	Sign
acp_{12}	Receive	Loading task	+
acp_{13}	Load	Supply, mule	+
acp_{14}	Report to central DB	Robot status	+
acp_{15}	Report to Manager	Robot status	−

at policy enforcement stage that then requires conflict resolution strategies [67] be applied. Minimizing the inconsistencies is thus critical to reduce the need for conflict resolutions activities.

Example: As shown in Table 3.1, acp_1 specifies that a robot with the manager role has a positive permission to receive notifications from all bunkers. However, based on acp_2 the role manager is forbidden from receiving notifications from the bunker with number 10. Hence, acp_1 is inconsistent with acp_2.

Definition 3.2.2 (Policies Exceptions) *A transaction* $t_i \in T(u \in U,\ o \in O,\ a \in A)$ *is an exception with respect to an access control policy* $acp_j \in ACP$ *if and only if*

- $t_i.o = acp_j.p.o \wedge t_i.a = acp_j.p.a \wedge acp_j.r \in UA(t_i.u_i) \wedge acp_j.p.sign = \text{`}-\text{'}$
- $\exists PR_k \in SimP.Processes \mid PR_k.Operation = t_i.a \wedge PR_k.Data = t_i.o \wedge PR_k.User = t_i.u.$

A policy exception arises when a transaction is executed that violates a negative authorization. In general, whereas exceptions may arise due for example to unforeseen circumstances; it is important to minimize their occurrences. Exceptions may require explicit ad-hoc and temporary authorizations from human administrators, which can be expensive and not always possible. It is thus therefore critical to analyze exceptions to determine whether policies should be modified so to be able to cover the frequently occurring exceptions.

Example: Consider policy acp_{15} from Table 3.2. According to such a policy, a worker is not authorized to communicate with the manager about emergency situations. Suppose that one of the worker robots has a malfunction while performing a loading task. The worker robot should notify the manager about the situation to enable the manager to reassign the task to another worker. To solve this situation, the administrator allows the worker robot to notify the manager about the task by adding a temporary access control policy and disabling acp_{15}. However, it is clear

that a modification of the policy allowing a robot to notify the manager in the case of malfunctioning would be a more efficient solution.

Definition 3.2.3 (Incompleteness) *A set of access control policies is incomplete if and only if*

- $\exists t_i \in T \mid t_i = (u_i \in U,\ o_i \in O,\ a_i \in A$
- $\exists PR_k \in SimP.Processes \mid PR_k.Operation = t_i.a \wedge PR_k.Data = t_i.o \wedge PR_k.User = t_i.u$
- $\nexists acp \in ACP \mid t_i.o = acp.p.o \wedge t_i.a = acp.p.a \wedge acp.r \in UA(t_i.u_i).$

Incompleteness refers to the situation in which an access request is issued that is not covered by the current policies.

Example: Consider the case in which a worker asks information about the capacity of a mule. In this case, there is no policy either allowing or denying the access to this information.

In cases like the one in the previous example, we say, as in the well-known XACML standard [19], that there is no applicable policy. Making sure that all actions are covered by some policies is critical to enhance the predictability of device behaviors.

Definition 3.2.4 (Redundancy) *An access control policy $acp_i \in ACP$ is redundant if and only if*

- $\exists acp_j \in ACP$
- $acp_i.r = acp_j.r \wedge (acp_i.p.o \subseteq acp_j.p.o \vee acp_j.p.o \subseteq acp_i.p.o) \wedge acp_i.p.a = acp_j.p.a \wedge acp_i.p.sign = acp_j.p.sign.$

Redundancy arises when there is a set of similar policies that control the same situation of interest. Detecting redundancy helps in reducing the size of the policy set. In addition, it enhances security.

Example: Consider the policies in Table 3.1. Based on acp_1 and acp_3 a robot manager is authorized to receive notifications from Bunker 5. Hence, these two policies will be enforced. However it is clear that policy acp_3 is redundant with respect to acp_1 and as the latter is more general, the former can be removed.

Definition 3.2.5 (Irrelevancy) *An access control policy acp_i ACP is irrelevant if and only if*

- $\nexists PR_k \in SimP.Processes \mid PR_k.Operation = acp_i.p, a \wedge PR_k.Data = acp_i.p.o \wedge acp_i.r \in UA(PR_k.User)$
- $\nexists PL_k \in SimP.Policies \mid PL_k.Operation = acp_i.p, a \wedge PL_k.Data = acp_i.p.o \wedge acp_i.r \in UA(PL_k.User)$.

Irrelevancy refers to the situation in which no access requests are issued to which a given policy is applicable. Removing irrelevant policies enhances security and enhances usability in cases in which human users have to inspect the policies, for example when solving policy conflicts.

Example: Consider the policies in Table 3.1. According to acp_4, the robot manager is able to inquire about a bunker status. Nonetheless, such a policy is irrelevant as bunkers automatically send requests.

Removing irrelevant policies is critical when policies are not used, as irrelevant policies may undermine security. For example, an attacker may try to compromise a user in order to exploit the privileges of this user. Thus, making sure that a user does not have permissions for actions that the user is not expected to execute is critical to minimize such exploitations.

3.2.2 Structures for Policy Analysis

Policy analysis requires scanning the policy set to detect the policies which violate the policy quality requirements. Furthermore, assessing policy quality requires

searching two types of data sources: the set of policies and the set of executed transactions in the system. Therefore, to support such searches, we introduce the following structures.

Policy Tree

The set of access control policies ACP is represented by a balanced multi-way tree structure (referred to as *policy tree*) (see Fig. 3.1 for an example[1]). A policy tree contains four types of nodes: role, action, object, and sign. The root node consists of role nodes which represent all distinct roles in ACP. Each role node points to a set of action nodes which represent the authorized actions to the role. Furthermore, each action points to a set of objects on which the corresponding role is allowed to perform the corresponding actions. Each object node points to a leaf node which represents the authorization permission sign. The leaf node is augmented with two additional values: *Policy ID* which refers to the identifier of the policy represented by the corresponding path, and *Counter* which represents how many times the policy was enforced.

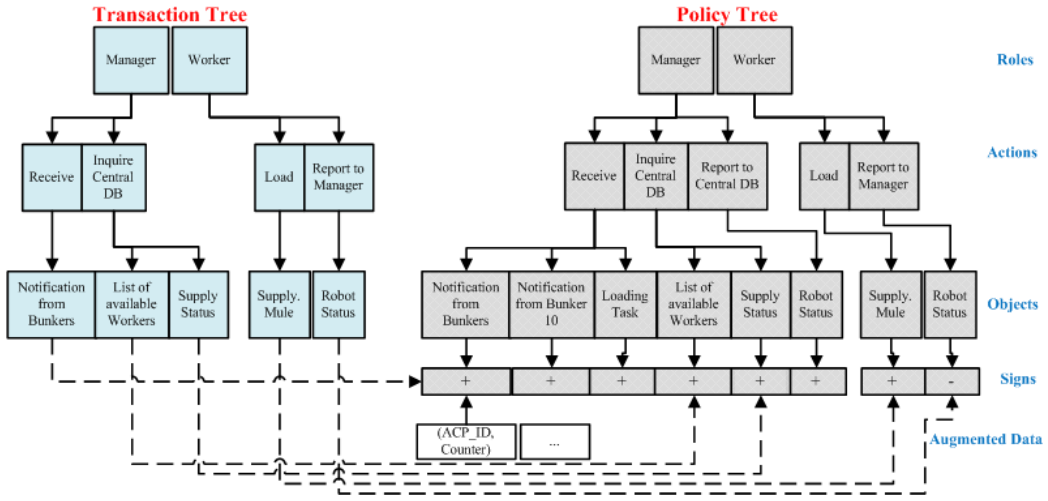


Fig. 3.1.: Policy Analysis Structures: Policy Tree (right) and Transaction Tree (left)

[1]The figure is based on the policies with odd identifiers in Tables 3.1 and 3.2 to simplify the figure.

Each node in the tree has un-limited fan-out (i.e., the maximum number of children) so the constructed structure is a wide and shallow tree. Heuristically, for constructing the policy tree, we choose role nodes to be stored in the first level to minimize the number of tree branches; hence searching the tree becomes more efficient. Furthermore, each node in the tree contains a hash table where the main elements in the node are the keys while the key values are the corresponding pointers to the successor nodes. This enhances the efficiency of the searches on the policy tree for a certain access control policy as it avoids the sequential inspection of all child nodes in each node.

In distributed environments, each party might have its own local policy set; hence a set of policy trees are constructed.

Transaction Tree

The set of transactions that are executed in the system is represented by another balanced multi-way tree (referred to as *transaction tree*) (see Fig. 3.1 for an example). The structure of this tree is similar to that of the policy tree, but the leaf nodes are a set of objects where each leaf node is augmented with a counter (indicates how many times a transaction recurred) and pointer(s) to leaf node(s) in the policy tree(s). A leaf node in the transaction tree is mapped optimally to one leaf node in the policy tree. However, it might be mapped to more than one leaf node (e.g., in the case of the existence of redundant policies).

The transaction tree is populated with the transactions which are captured by the provenance framework. Hence, the provenance repository is periodically queried about recent transactions in order to maintain the transaction tree. The maintenance of the transaction tree is illustrated in Algorithm 3.1. The retrieved provenance records include a set of processes where each process record contains a set of operations and each operation executed by a user and manipulated a set of data. An operation which is executed by a user and manipulates data is mapped to a transaction t (i.e.,

user, object, and action). Since SimP captures the role assigned to the user at the operation execution time, SimP identifies the role which was assigned to the user who executed a transaction. A transaction tuple along with its corresponding role is inserted into the transaction tree. Next, we search the policy tree to identify the policy which potentially controls the corresponding transaction. If a policy is found, we connect the leaf node of a transaction path in the transaction tree with the leaf node of the matching policy path in the policy tree and update their counters accordingly.

Table 3.3.: Notations for The Asymptotic Time Analysis

Notation	Description
n	The number of access control policies.
m	The number of transactions.
f_R	The number of unique roles which represents the maximum fan-out of the root node of the tree structures.
f_A	The number of unique actions which represents the maximum fan-out of an action node in the tree structure.
f_O	The number of unique objects which represents the maximum fan-out of an object node in the tree structure.

Policy-and-Transaction Tree (PT-Tree)

Instead of building two individual structures (i.e., policy tree and transaction tree), a hybrid structure (referred to as a *policy-and-transaction tree*) can be built to store both sets of transactions and access control policies in a tightly-combined organization. In order to build such an integrated structure, the structure of policy tree can be adapted to store access control policies as well as their corresponding transactions. When an executed transaction does not have a corresponding policy

that controls it, the transaction is added as a policy but the value for the sign node is special[2] and the value for *Policy ID* is empty. Otherwise, the path corresponding to the enforced policy while executing the transaction is updated by incrementing the *Counter* value.

Time Complexity of Structure Construction

Algorithm 3.1 Maintain Transaction Tree

Require: $TransactionTree$, $PolicyTree$, $SimP$

1: **for** $\exists PR_k \in$ **query**$(SimP.Processes)$ **do**
2: **Define** t *as* $Transaction$
3: $t.a = PR_k.Operation,\ t.o = PR_k.Data,\ t.u = PR_k.User$
4: $r = PR_k.User.Role$
5: $leaf_1 =$ **Insert**$(r, t.a, t.o)$ *into* $TransactionTree$
6: $leaf_2 =$ **Search**$(r, t.a, t.o)$ *into* $PolicyTree$
7: $leaf_1.counter = leaf_1.counter + 1$
8: **if** $leaf_2 \neq \phi$ **then**
9: $link(leaf_1, leaf_2)$
10: $leaf_2.counter = leaf_2.counter + 1$
11: **end if**
12: **end for**

For discussing the time asymptotic analysis for constructing the policy, transaction, and PT trees, we used the notations listed in Table 3.3.

At the running time of a system, every transaction executed by a user is mapped to a record matching an access control policy. Thus, the unique set of transactions m is bounded by the set of access control policies (i.e., $m = \mathcal{O}(n)$). Regarding the structure of the policy tree and transaction tree, the fan-out of a node depends on the

[2]It implies that there is no corresponding permission neither accepting nor rejecting the access.

node type (i.e., the fan-out of a root node (f_R), action node f_A, and object node f_O vary based on the specifications of an access control system). However, the fan-out of a node in these trees f can be generalized by the maximum of the various node fan-outs (i.e., $f = max(f_R,\ f_R,\ f_R)$). Subsequently, the time cost of searching the policy or transaction tree is $\mathcal{O}(\log_f n)$. Inserting a transaction or a policy involves searching the tree structure; hence the time cost of insertion is also $\mathcal{O}(\log_f n)$.

Based on the aforementioned discussion, given n policies, the time cost of constructing the policy tree is $\mathcal{O}(n \log_f n)$. As shown in Algorithm 3.1, while constructing the transaction tree, the operations of inserting a transaction into the transaction tree and searching for the corresponding policy in the policy tree are performed in sequential; hence these two operations are bounded by $\mathcal{O}(\log_f n)$. Given m transactions, the time cost of constructing the transaction tree is also $\mathcal{O}(m \log_f n)$. Thus, the construction time of a transaction tree is asymptotically larger than the one of a policy tree.

Intuitively, the asymptotic analysis of the PT tree is bounded by the maximum of that for the policy tree and transaction tree (i.e., $\mathcal{O}(max($policy tree, transaction tree$))$. Subsequently, the insertion and construction times of the hybrid structure are $\mathcal{O}(\log_f n)$, $\mathcal{O}(m \log_f n)$, respectively.

3.3 Policy Analysis Services

Our framework supports two types of analysis: structure-based and classification-based. The structure-based analysis requires maintaining two data structures for analyzing all policies and find any "low quality" policy. However, this approach is expensive. Alternatively, the classification-based analysis approach learns the characteristics of policies of each type of "low quality" policies. The classification-based approach is able to quickly predict the quality of a policy, but inaccurate prediction might happen.

3.3.1 Structure-based Analysis

The search space of the policy-based analytic service is bounded by the access control policy set itself. Hence the policy-based analytic service is not able to assess all quality requirements (e.g., cannot assess incompleteness). Thus, the transaction-based analytic service expands the search space to cover both the executed transactions and their corresponding policies. From another perspective, the policy-based analysis follows the paradigm "*analyze first and enforce later*" while the transaction-based analysis follows the opposite paradigm (i.e.,"*enforce first and analyze later*"). For situations that can be detected by both types of analysis, the policy-based one is preferred because it provides early feedback about the quality of policies.

In the distributed environment that has separate policy sets defined for each party, the transaction-based analysis makes it possible to determine which policies belong to different parties while the policy-based analysis is local to each party in the system. In what follows, we describe our approaches.

Policy-based Analysis

In this service, we aim to evaluate the policies in order to detect: inconsistency, redundancy, and irrelevancy. The service utilizes the policy tree structure.

The policy-based analysis is described in Algorithm 3.2. Lines 2-11 traverse every path in the policy tree from the root (role node) to an object node to validate a set of conditions as follows.

- If a path branches to two different signs, this flags for inconsistency (lines 4-5).
- If the leaf node of a path is augmented with multiple policy IDs, this shows a case of redundancy (lines 7-9).
- If the counter value of a leaf node is zero, this flags for irrelevancy (lines 10-11).

Furthermore, lines 12-19 descends the tree from the root to the action nodes to check if there are object nodes which are composite of each other to assess for inconsistency and redundancy.

Time Complexity: The policy-based analysis requires inspecting all access control policies n. Furthermore, for every policy, the approach searches for the other policies that involve similar objects. This leads to the time cost of $\mathcal{O}(n \log_f n)$.

Transaction-based Analysis

In this service, we aim to evaluate the policy set to detect: inconsistency, redundancy, incompleteness, and exceptions. Unlike the policy-based analysis, this service utilizes the transaction tree primarily and explores the policy tree.

Table 3.4.: The Objectives of Policy Analysis Services

	Policy-based Analysis	Transaction-based Analysis
Inconsistency	✓	✓
Exception	✗	✓
Incompleteness	✗	✓
Redundancy	✓	✓
Irrelevancy	✓	✗

The transaction-based analysis is described in Algorithm 3.3. The algorithm traverses every path in the transaction tree and explores its corresponding policies in the policy tree. While traversing the tree, it validates the following conditions:

- If a transaction path does not point to any policy in the tree, this flags for incompleteness (lines 4-5).
- If a transaction points to two policies which have different signs, this shows a case of inconsistency (line 6-7).

Algorithm 3.2 Policy-Based Analysis

Require: $PolicyTree$

$\mathcal{L}_{INCON} \longleftarrow \{\}, \mathcal{L}_{IRR} \longleftarrow \{\}, \mathcal{L}_{RED} \longleftarrow \{\}$
for *path* $(r, a, o) \in PolicyTree$ **do**
  $\mathbb{N} = \{\forall g \mid g\ is\ leaf\ node\ for\ (r, a, o)\}$
  if $g_i \in \mathbb{N} \wedge g_j \in \mathbb{N} \wedge i \neq j \wedge g_i \neq g_j$ **then**
    $\mathcal{L}_{INCON} \longleftarrow \mathcal{L}_{INCON} \cup \{g_i, g_j\}$
  end if
  for $g_i \in \mathbb{N}$ **do**
    $ID = \#ofPolicy\ IDs\ augmented\ with\ g_i$
    if $ID > 1$ **then**
      $\mathcal{L}_{RED} \longleftarrow \mathcal{L}_{RED} \cup \{g_i\}$
    end if
    if $g_i.counter = 0$ **then**
      $\mathcal{L}_{IRR} \longleftarrow \mathcal{L}_{IRR} \cup \{g_j\}$
    end if
  end for
  Traverse *each path* (r, a) **in** $PolicyTree$
  $\mathbb{N} = \{\forall o \mid o\ is\ child\ node\ for\ (r, a)\}$
  if $o_i \in \mathbb{N} \wedge o_j \in \mathbb{N} \wedge i \neq j \wedge o_i \subseteq o_j$ **then**
    $\mathbb{N}_1 = \{\forall g \mid g\ is\ leaf\ node\ for\ (r, a, o_i)\}$
    $\mathbb{N}_2 = \{\forall g \mid g\ is\ leaf\ node\ for\ (r, a, o_j)\}$
    if $g_i \in \mathbb{N}_1 \wedge g_j \in \mathbb{N}_2 \wedge g_i \neq g_j$ **then**
      $\mathcal{L}_{INCON} \longleftarrow \mathcal{L}_{INCON} \cup \{g_i, g_j\}$
    end if
    if $g_i \in \mathbb{N}_1 \wedge g_j \in \mathbb{N}_2 \wedge g_i = g_j$ **then**
      $\mathcal{L}_{RED} \longleftarrow \mathcal{L}_{RED} \cup \{g_i, g_j\}$
    end if
  end if
end for
return $\mathcal{L}_{INCON}, \mathcal{L}_{IRR}, \mathcal{L}_{RED}$

Algorithm 3.3 Transaction-Based Analysis

Require: TT: $TransactionTree$, PT: $PolicyTree$

1: $\mathcal{L}_{INCON} \leftarrow \{\}$, $\mathcal{L}_{INCOMP} \leftarrow \{\}$, $\mathcal{L}_{RED} \leftarrow \{\}$, $\mathcal{L}_{EXP} \leftarrow \{\}$
2: **Traverse** *each path* (r, a, o) **in** TT
3: $\mathbb{N} = \{\forall g \mid g \; is \; pointed \; by \; o \; , g \; is \; a \; leaf \; node \; \in \; PT\}$
4: **if** $\mathbb{N} \equiv \phi$ **then**
5: $\quad \mathcal{L}_{INCOMP} \leftarrow \mathcal{L}_{INCOMP} \cup \{(u, a, o)\}$
6: **end if**
7: **if** $g_i \in \mathbb{N} \wedge g_j \in \mathbb{N} \wedge i \neq j \wedge g_i \neq g_j$ **then**
8: $\quad \mathcal{L}_{INCON} \leftarrow \mathcal{L}_{INCON} \cup \{g_i, g_j\}$
9: **end if**
10: **for** $g_i \in \mathbb{N}$ **do**
11: $\quad ID = \#of Policy \; IDs \; augmented \; with \; g_i$
12: $\quad$ **if** $ID > 1$ **then**
13: $\quad\quad \mathcal{L}_{RED} \leftarrow \mathcal{L}_{RED} \cup \{g_i\}$
14: $\quad$ **end if**
15: $\quad$ **if** $g_i \equiv -$ **then**
16: $\quad\quad \mathcal{L}_{EXP} \leftarrow \mathcal{L}_{EXP} \cup \{(u, a, o)\}$
17: $\quad$ **end if**
18: **end for**
19: **return** $\mathcal{L}_{INCON}, \mathcal{L}_{INCOMP}, \mathcal{L}_{RED}, \mathcal{L}_{EXP}$

- If a transaction points to one policy path, but the path is augmented with multiple policy IDs, this shows a case of redundancy (lines 9-11).
- If a transaction points to a policy path where the sign of the policy is '-', this flags for exceptions (lines 12-13).

Time Complexity: The transaction-based analysis requires inspecting all unique set of transactions (i.e., $\mathcal{O}(n)$). For every transaction, the approach checks its corresponding policy through the associated pointer linking both trees; hence requiring a constant time. Consequently, the time cost of transaction-based analysis approach is $\mathcal{O}(n)$.

Policy-and-Transaction-based Analysis (PT-based Analysis)

Neither of the policy-based analysis nor the transaction-based analysis are able to achieve all of the objectives of the policy analysis as shown in Table 3.4. Nonetheless, performing an analysis utilizing the PT tree can achieve all of the analysis objectives effectively.

Time Complexity: The PT-based analysis requires inspecting all access control policies and their corresponding transactions stored in the hybrid structure and verifying the similarity of each policy with other policies. Thus, the time complexity of the PT-based analysis is bounded by the maximum of that for the Policy-based analysis and Transaction-based analysis. Subsequently, the time cost of the PT-based analysis is $\mathcal{O}(n \log_f n)$.

3.3.2 Classification-based Analysis

The structure-based analysis requires maintaining some data structures. To avoid having to inspect the tree structures periodically, the classification-based analysis (see Fig. 3.2) aims at learning the patterns of low-quality policies based on the historical results of the structure-based analysis and then generating a classifier. Thereafter,

the classifier is used to assess policies and predict whether they will not meet the quality requirements.

Background on Classifiers

Here, we review a set of well-known classifiers. First, the k-Nearest Neighbors (kNN) classifier is the simplest one which relies on the kNN search on the training dataset. The class of an object is identified based on the majority voting of its kNN. Second, Naïve Bayes is a probabilistic classifier based on Bayes' theorem [68] which enables calculating a posterior probability for each class at prediction. Third, Support Vector Machine (SVM) is designed for binary classification. For multi-label classification, SVM is generalized in two schemes: one-versus-all and one-versus-one. Our work considers the one-versus-one scheme in which each pairwise class group is chosen as a two-class SVM each time.

Some classifiers are tree-based such as in the Decision Tree (DT) classifier. A DT organizes a series of test conditions in a tree structure [69] where the internal nodes represent criteria for the attributes of each class, and the leaf nodes are associated with class labels. Random Forest, an extension of DT, includes a set of DTs, and the output of classification is defined by the leaf node that receives the majority of votes [70].

The historical results generated from the structure-based policy analysis can be used to generate the patterns of policies that are of "low quality" and thus create categories (i.e., classes) of policies. Each policy quality requirement characterized by a sample set of policies is considered as a class for the corresponding policies. A subset of policies for each category is used to train a classifier for creating a model that summarizes the patterns of policies belonging to each category. In particular, each policy and transaction is represented as a feature vector consisting of role, ob-

ject, action, sign, and policy quality class[3] (e.g., irrelevant, inconsistent). Then, the classifier utilizes the model for predicting the class of new policies.

There are, however, two main challenges that are associated with the classification-based analysis approach: the imbalanced categories of historical policies and the inevitable classification inaccuracy. Learning the patterns of imbalanced categories potentially leads to a biased pattern learning for the dense categories (i.e., the categories which have many policies); hence, increasing the classification inaccuracy to the sparse categories (i.e., the categories which have few numbers of policies). Thus, for obtaining balanced categories of policies, we sample more policies policies using the SMOTE (Synthetic Minority Oversampling Technique) algorithm [71] to oversample [72] the class with minority samples. Regarding the classification inaccuracy, there is no optimal classifier that definitely guarantees accurate prediction results. Thus, our framework addresses this challenge by adopting the cross-validation mechanism while training a classifier and proposing a classification scheme which combines the classification results obtained from different classifiers to enhance the classification accuracy.

Classification Approaches

Here, we present two classification schemes which we use in our framework.

Classification of Access Control Policies

One Classifier (OC)

The OC approach adopts one of the state-of-the-art classifiers (i.e., k-Nearest Neighbors (kNN), Naïve Bayes, Support Vector Machine (SVM), Decision Tree, or Random Forest) at a time. In the experiment section, we discuss the impact of the choice of the classifier on the policy classification.

[3]The classification of policy quality is a multi-class classification (i.e., irrelevant, inconsistent, redundant, exception, incomplete, and good-quality).

Combined Classifiers (CC)

Since classifiers inevitably suffer from inherent classification inaccuracy, classifying a policy with different classifiers may potentially result in various categories for a given policy. Consequently, combining the results of various classifiers potentially increases the certainty of the classification result. We investigate two methods for

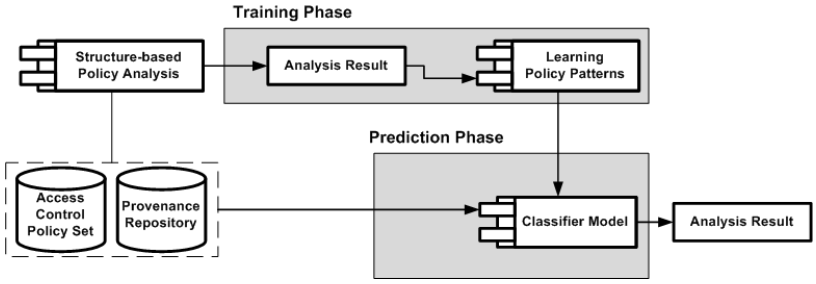


Fig. 3.2.: The Pipeline of Classification-based Policy Analysis

combining the classifiers: majority voting [73] (referenced as *Majority-based Combined Classifiers* (MCC)) and maximum probability [74] (referenced as *Probability-based Combined Classifiers* (PCC)). The majority voting method builds a consensus of opinion among classifiers. In particular, the method selects the predicted class that is supported by the majority of the classifiers. If there is no majority voting (i.e., a strong disagreement among classifiers concerning the predicted class), the method randomly selects one of the classes predicted by one of the classifiers. Meanwhile, PCC utilizes the probabilities associated with the predicted class using every classifier and chooses the class from the classifier whose probability is the highest. The two methods of the combined classification scheme are formalized in Algorithm 3.4.

3.4 Policy Evolution Services

A dynamic system often requires modifying its access control policies. Especially, when the policy analysis process identifies some low-quality policies. The analysis results can be then be used to evolve the system. However, modifying the policies

Algorithm 3.4 Classification-based Analysis - Combined Classifiers Approach

Let D denote the training set of Access Control Policies, k denote the number of base classifiers, T be the test set of policies, m denote the mode of combined classifier approach, and n denote the number of classes.

for $i \in \{1, \ldots, k\}$ **do**
    Build a base classifier C_i from D
    for each class $j \in \{1, \ldots, n\}$ **do**
        $\alpha_{ij} = \text{Accuracy}(C_i, \forall \text{ policy } x \in j)$
    end for
end for
for each policy $x \in T$ **do**
    if m: Majority voting based **then**
        $C^*(x) = \text{Vote}(C_1, C_2, \ldots, C_K)$
    else if m: Probability-based **then**
        $C^*(x) = C_i(x) \mid \alpha_i = \max_{i=1}^{k} \alpha_i$
    end if
end for

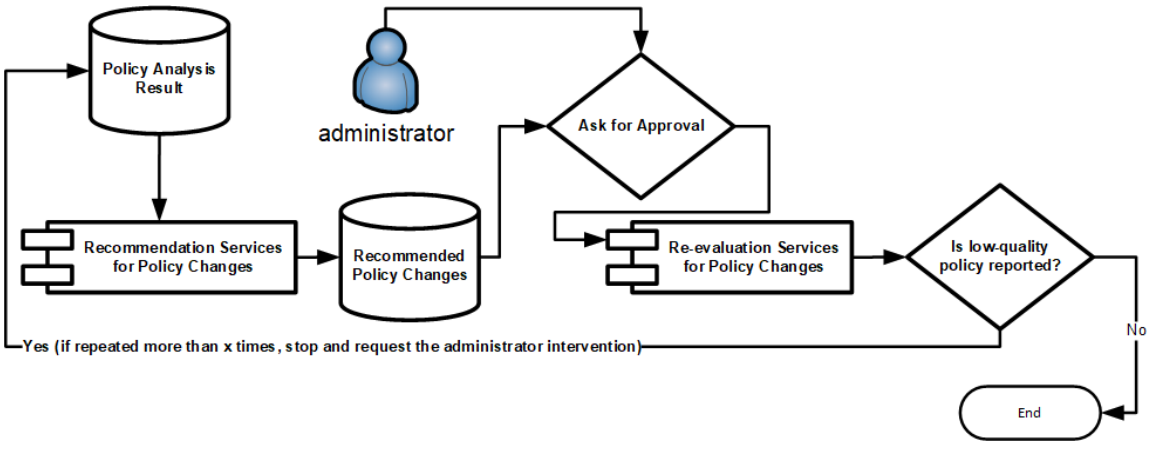


Fig. 3.3.: Policy Evolution Services

manually is not reliable and is expensive in terms of human administrative time and efforts. Hence, we propose a set of policy evolution algorithms that aim at automatically evolving the policies in order to enhance them. The policy evolution module (see Fig. 3.3) comprises two services, i.e., recommendation and re-evaluation services, that we describe in what follows.

Table 3.5.: Primitive Changes on Policies

Primitive	**Description**
$DEL\_POLICY(ACP_i)$	Delete the policy ACP_i
$CHG\_POLICY_r(ACP_i, x)$	Change policy role: $ACP_i.r = x$
$CHG\_POLICY_a(ACP_i, x)$	Change policy action: $ACP_i.p.a = x$
$CHG\_POLICY_o(ACP_i, x)$	Change policy object: $ACP_i.p.o = x$
$CHG\_POLICY_s(ACP_i, x)$	Change policy sign: $ACP_i.p.sign = x$
$ADD\_POLICY(ACP_i)$	Add the policy ACP_i $\langle$role: $ACP_i.r$, permission: $(ACP_i.p.a, ACP_i.p.o, ACP_i.p.sign)\rangle$
$ADD\_ROLE(r_i, r_j)$	Add the role r_i to the parent role r_j
$DEL\_ROLE(r_i)$	Delete the role r_i

3.4.1 Recommendation Services for Policy Changes

The recommendation services receive as input the results of the policy analysis module and return a set of recommended primitive policy changes. In particular, there is a recommendation service for each type of low-quality policies (see Algorithms 3.5-3.9). Each service inspects the set of access control policies and transactions thoroughly to evaluate the causes for low-quality policies and suggests accordingly one or multiple primitive changes (see Table 3.5 for the list of primitive changes).

The recommendation service for redundant policies is described in Algorithm 3.5. Given two redundant policies, the service basically checks whether the two policies

are identical (lines 2-3) or similar (lines 4-33). Checking the similarity of two policies implies verifying the similarity of their corresponding three components (i.e., role, object, and action) [4] as follows:

- If the corresponding roles are related by the parent-child relationship, the two policies are similar (lines 5-22). Hence, the service recommends either deleting the policy having the child role or changing the role of the parent policy through analyzing the executed transactions associated with them.

- If the object of one policy is a child of the object of the other policy, the two policies are similar (lines 23-27). Hence, the service recommends deleting the policy corresponding to the child object.

- If the action of one policy is partially implied by the action of the other policy, the two policies are considered similar (lines 28-32). Hence, the service recommends deleting the policy corresponding to the partial action.

Regarding inconsistent policies, their recommendation service is described in Algorithm 3.6. Given two inconsistent policies, the service checks the similarity of the corresponding three components. In particular, if the corresponding roles are related by the parent-child relationship, the inconsistent policies are considered similar. Subsequently, the service recommends creating a new role for the users belonging to the parent role (except the users belonging to the child role of interest), and then assigning the policy associated with the parent to the new role (lines 4-10). Also, the service recommends similar changes when the inconsistent policies are considered similar with respect to the objects (lines 11-16) and actions (lines 17-23).

The recommendation service for exceptional transactions is described in Algorithm 3.9. Given a policy and its corresponding exceptional transaction, the service checks whether the policy has been enforced. If not, the service recommends negating the sign of the policy. Otherwise, the service checks the following conditions when the

[4]The sign is not checked because the two policies are reported as redundant and thus they have the same sign.

exceptional transaction has been repeated a number of times higher than a certain threshold (i.e., σ):

- If the corresponding roles of the exceptional transaction and the policy are related by a parent-child relationship, the service recommends creating a new role for the users belonging to the parent role (except the users belonging to the child role of interest), and then assigning the policy associated with the parent to the new role (lines 8-11). Also, the service recommends adding a new policy to permit the exceptional transaction for the child role (lines 12-14).

- If the corresponding objects (lines 18-26) or actions (lines 27-35) of the exceptional transaction and the policy are related by a parent-child relationship, the service recommends similar changes to the ones recommended when the roles of the exceptional transaction and its associated policy are related by a parent-child relationship.

Due to space limitations, we omit the algorithms for the recommendation services in two cases: irrelevancy and incompleteness. Those algorithms are quite simple. In the case of irrelevant policies, the corresponding service recommends deleting it, whereas in the case of incomplete policies the service recommends adding new policies to cover the missing scenarios.

Validating the parent-child relations, among the corresponding roles, objects, and actions of the two policies, requires inspecting some auxiliary structures. In particular, the roles are validated using the role hierarchies captured by the RBAC model while the objects are inspected using metadata defining object hierarchies in the system.

3.4.2 Re-evaluation Services for Policy Changes

The second set of services is to perform the changes recommended by the recommendation services, and then re-evaluate the policies which are affected by these recommended changes to assure that the quality of the policy set has been improved

as intended. Evaluating the policy set after its evolution is a critical step. However, re-evaluating the quality of the entire policy set can be expensive. Thus, the re-evaluation services particularly consider only the policies that are directly affected by the evolution.

Towards such goal, the re-evaluation services investigate each of the primitives changes to track their effects on the set of policies and narrows the scope of the re-analysis to be performed only for them. However, some of these primitives, such as ADD_ROLE and DEL_ROLE, have no direct impact on the policy set. Nonetheless, such primitives are associated with other primitives (i.e., CHG_POLICY_r and DEL_POLICY, respectively) which directly affect the policy set.

When adding a new policy or deleting an existing policy acp_i (i.e., the primitive ADD_POLICY and DEL_POLICY, respectively), the policy tree (or PT tree) is first searched using a depth-first-search to find a path whose nodes match the components composing acp_i. In particular, the set of policies that are affected by this change primitive includes any policy acp_j that is specified by the components: a role ($acp_j.r$) matched to $acp_i.r$ (or $acp_j.r$ and $acp_j.r$ are related by a parent-child relation), an action $acp_j.p.a$ matched to (or part of) $acp_i.p.a$, and an object $acp_j.p.o$ matched to $acp_i.p.o$ (or $acp_j.o$ and $acp_j.o$ are related by a parent-child relation).

Meanwhile, when changing a role of a policy (i.e., the primitive CHG_POLICY_r), the set of policies that are affected by this change primitive includes any policy acp_j that is specified by the components: a role $acp_j.r$ matched to (or share a parent-child relation with) either of the old or new values of $acp_i.r$, an action $acp_j.p.a$ matched to (or part of) either of the old or new values of $acp_i.p.a$, and an object $acp_j.p.o$ matched to (or share a parent-child relation with) either of the old or new values of $acp_i.p.o$. Similarly, finding the policies affected by the execution of the primitives CHG_POLICY_a and CHG_POLICY_o follows the same logic. The procedure of finding the set of affected polices for all primitive changes is outlined in Algorithm 3.10. Thereafter, the identified affected policy set is re-evaluated either by partially

traversing the policy tree (or PT tree) or performing the classification-based analysis on the batch of affected policies.

Algorithm 3.5 Recommendation Algorithm for Redundant Policies

1: Given two policies ACP_i and ACP_j are redundant.
2: **if** $ACP_i = ACP_j$ **then**
3: Recommend $DEL\_POLICY(ACP_i)$
4: **else**
5: **if** $ACP_i.r \subseteq ACP_j.r$ **then**
6: t_i = Query_Transactions(ACP_i)
7: t_j = Query_Transactions(ACP_j)
8: **if** $\mid t_j - t_i \mid = 0$ **then**
9: Recommend $DEL\_POLICY(ACP_i)$
10: **else if** $\mid t_j - t_i \mid > \mid t_i \mid$ **then**
11: Recommend $DEL\_POLICY(ACP_i)$
12: **if** |Query_Policies($acp_i.r$) = 1 **then**
13: Recommend $DEL\_ROLE(acp_i.r)$
14: **end if**
15: **else**
16: Recommend $ADD\_ROLE(r_k, acp_j.r)$
17: Assign users(t_j - t_i) to r_k
18: Recommend $CHG\_POLICY_r(ACP_j, r_k)$
19: **end if**
20: **else**
21: Similar Logic to Lines 6-18 but swapping i and j.
22: **end if**
23: **if** $ACP_i.p.o \subseteq ACP_j.p.o$ **then**
24: Recommend $DEL\_POLICY(ACP_i)$
25: **else**
26: Recommend $DEL\_POLICY(ACP_j)$
27: **end if**
28: **if** $ACP_i.p.a \subseteq ACP_j.p.a$ **then**
29: Recommend $DEL\_POLICY(ACP_i)$
30: **else**
31: Recommend $DEL\_POLICY(ACP_j)$
32: **end if**
33: **end if**

3.5 Query Services

ProFact includes a query services component supporting the following query types:

Algorithm 3.6 Recommendation Algorithm for Inconsistent Policies

1: Given two policies ACP_i and ACP_j are inconsistent
2: t_i = Query_Transactions(ACP_i)
3: t_j = Query_Transactions(ACP_j)
4: **if** $ACP_i.r \subseteq ACP_j.r$ **then**
5: Recommend $ADD\_ROLE(r_k, acp_j.r)$
6: Assign users(t_j - t_i) to r_k
7: Recommend $CHG\_POLICY_r(ACP_j, r_k)$
8: **else**
9: Similar Logic to Lines 5-7 but swapping i and j.
10: **end if**
11: **if** $ACP_i.p.o \subseteq ACP_j.p.o$ **then**
12: $o_k = (\forall\ o_w \in ACP_j.p.o)$ - $ACP_i.p.o$
13: Recommend $CHG\_POLICY_o(ACP_j, o_k)$
14: **else**
15: Similar Logic to Lines 12-13 but swapping i and j.
16: **end if**
17: **if** $ACP_i.p.a \subseteq ACP_j.p.a$ **then**
18: Recommend $ADD\_ROLE(r_k, acp_j.r)$
19: $a_k = (\forall\ a_w \in ACP_j.p.a)$ - $ACP_i.p.a$
20: Recommend $CHG\_POLICY_a(ACP_j, a_k)$
21: **else**
22: Similar Logic to Lines 18-20 but swapping i and j.
23: **end if**

Algorithm 3.7 Recommendation Algorithm for Irrelevant Policies

1: Given that a policy ACP_i is irrelevant
2: Recommend $DEL\_POLICY(ACP_i)$
3: **if** | Query_Policies($ACP_i.r$)| = 1 **then**
4: Recommend $DEL\_ROLE(acp_i.r)$
5: **end if**
6: r_p = parent($ACP_i.r$)
7: **if** $r_p \neq NULL$ **then**
8: **if** | Query_Policies(r_p)| = 1 **then**
9: Recommend $DEL\_ROLE(r_p)$
10: **end if**
11: **end if**

Algorithm 3.8 Recommendation Algorithm for Incomplete Policies

1: Given a transaction t_i $\langle r, o, a\rangle$ that does not have a corresponding policy
2: Set ACP_i = $\langle$role: $t_i.r$, permission: $(t_i.a, t_i.o, +)\rangle$
3: Recommend $ADD\_POLICY(ACP_i)$

Algorithm 3.9 Recommendation Algorithm for Exceptional Transactions

1: Given a transaction t_i $\langle r, o, a\rangle$ that violates an existing policy ACP_j
2: n = Count_Transactions(t_i)
3: t_j = Query_Transactions(ACP_j)
4: **if** | t_j | = 0 **then**
5: $s = \neg\ ACP_j.p.sign$
6: $CHG\_POLICY_s(ACP_j, s)$
7: **else if** $n > \sigma$ **then**
8: **if** $t_i.r \subseteq ACP_j.r$ **then**
9: Recommend $ADD\_ROLE(r_k, acp_j.r)$
10: Assign users(t_j - t_i) to r_k
11: Recommend $CHG\_POLICY_r(ACP_j, r_k)$
12: $s = \neg\ ACP_j.p.sign$
13: Set $ACP_i = \langle$role: $t_i.r$, permission: $(t_i.a, t_i.o, s\rangle$
14: Recommend $ADD\_POLICY(ACP_i)$
15: **else**
16: Similar Logic to Lines 9-14.
17: **end if**
18: **if** $t_i.o \subseteq ACP_j.p.o$ **then**
19: $o_k = (\forall\ o_w \in ACP_j.p.o)$ - $t_i.o$
20: Recommend $CHG\_POLICY_o(ACP_j, o_k)$
21: $s = \neg\ ACP_j.p.sign$
22: Set $ACP_i = \langle$role: $t_i.r$, permission: $(t_i.a, t_i.o, s\rangle$
23: Recommend $ADD\_POLICY(ACP_i)$
24: **else**
25: Similar Logic to Lines 19-23.
26: **end if**
27: **if** $t_i.p.a \subseteq ACP_j.p.a$ **then**
28: $a_k = (\forall\ a_w \in ACP_j.p.a)$ - $t_i.a$
29: Recommend $CHG\_POLICY_a(ACP_j, a_k)$
30: $s = \neg\ ACP_j.p.sign$
31: Set $ACP_i = \langle$role: $t_i.r$, permission: $(t_i.a, t_i.o, s\rangle$
32: Recommend $ADD\_POLICY(ACP_i)$
33: **else**
34: Similar Logic to Lines 28-32.
35: **end if**
36: **end if**

Algorithm 3.10 Find Affected Policies

1: Let $acp_i :=< r, p.a, p.o, p.sign >$ denote a policy to be changed, M denote the policy evolution type $\{Add, Update, Delete\}$, and T denote the policy tree constructed on D.
2: **if** $M \equiv Add \vee M \equiv Delete$ **then**
3: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
4: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
5: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
6: $P = \{\forall acp_j \mid (acp_j.r \in R) \wedge (acp_j.p.a \in A) \wedge (acp_j.p.o \in O) \wedge (i \neq j)\}$
7: **else if** $M \equiv Update$ **then**
8: **if** updating $acp_i.r$ **then**
9: $R = \{\forall r \mid (r \subseteq acp_i.r_{old}) \vee (acp_i.r_{old} \subseteq r) \vee (r \subseteq acp_i.r_{new}) \vee (acp_i.r_{new} \subseteq r)\}$
10: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
11: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
12: **else if** updating $acp_i.p.a$ **then**
13: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
14: $A = \{\forall a \mid (a \subseteq acp_i.p.a_{old}) \vee (acp_i.p.a_{old} \subseteq a) \vee (a \subseteq acp_i.p.a_{new}) \vee (acp_i.p.a_{new} \subseteq a)\}$
15: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
16: **else if** updating $acp_i.p.o$ **then**
17: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
18: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
19: $O = \{\forall o \mid (o \subseteq acp_i.p.o_{old}) \vee (acp_i.p.o_{old} \subseteq o) \vee (o \subseteq acp_i.p.o_{new}) \vee (acp_i.p.o_{new} \subseteq o)\}$
20: **else if** updating $acp_i.p.sign$ **then**
21: $R = \{\forall r \mid (r \subseteq acp_i.r) \vee (acp_i.r \subseteq r)\}$
22: $A = \{\forall a \mid (a \subseteq acp_i.p.a) \vee (acp_i.p.a \subseteq a)\}$
23: $O = \{\forall o \mid (o \subseteq acp_i..p.o) \vee (acp_i..p.o \subseteq o)\}$
24: **end if**
25: $P = \{\forall acp_j \mid (acp_j.r \in R) \wedge (acp_j.p.a \in A) \wedge (acp_j.p.o \in O) \wedge (i \neq j)\}$
26: **end if**
27: **Return** P

- **Queries on the Quality of Policies:** Such queries retrieve the policies which do not satisfy our quality requirements. Querying on quality might be general (e.g., find all policies which are inconsistent, find all irrelevant policies) or specific to policy attributes (e.g., find all inconsistencies related to a specific object or find roles with respect to which policies are incomplete).

- **Queries on Policies:** These queries allow one to retrieve basic information on policies (e.g., policies for a role, how many times a policy was enforced) and advanced information on policies (e.g., the history of a policy, how the policy was evolved).

- **Queries on Transactions:** These queries allow one to retrieve information about the executed transactions. Examples include: find the transactions executed by a certain user (through different roles), and find the transactions that accessed specific objects.

- **Queries on Policy Analysis Statistics:** These queries utilize the policy analysis repository to retrieve aggregated analysis results. Examples include: retrieve the most common exceptions and the frequency of an exception.

- **Queries on Policy Evolution Statistics:** These queries allow one to estimate the percentage of policies affected by a specific type of policy changes or find the policy components mostly affected by a specific type of policy changes; these queries are based on the historical runs of the evolution service. Examples include: find the object which was mostly affected by policy deletion, and what is the primitive change that affected the largest number of policies.

3.6 Experiments

The goal of the experiments is to evaluate the two types of policy analysis approaches included in ProFact: structure-based and classification-based.

3.6.1 Dataset and Settings

Table 3.6.: Access Control Policy and Transaction Datasets

	# Roles	# Objects	# Policies	# Transactions
Dataset 1 (*DS1*)	50	2,000	46,800	124,800
Dataset 2 (*DS2*)	75	2,500	427,500	641,250
Dataset 3 (*DS3*)	100	3,000	877,200	1,152,000

Due to the lack of a large-scale real dataset compromising both access control policies and executed transactions for a real system[5], we created three synthetic datasets (referenced as *DS1*, *DS2*, and *DS3*) using a random dataset generator (as explained later) to evaluate the policy analysis approaches. Table 3.6 shows the size of each dataset in terms of the number of roles, protected objects, access control policies, and transactions.

Random Dataset Generator: The dataset generator first generates randomly the basic entities of a dataset (i.e., users, roles, objects, and actions[6]. Thereafter, the generator creates hierarchical relations (i.e., parent-child relations) among the created roles and objects. For example, for generating the role hierarchies, a quarter of the roles are initially selected as parent roles and each of the remaining roles is assigned as a child role to one of the parent roles and this child role is appended to the set of the parent roles. The object hierarchies are similarly generated. Using a maximum value for role assignment (i.e., m), each user is assigned randomly at maximum to a set of m roles. Subsequently, every user is assigned to multiple roles and every role includes multiple users (given the fact that the number of roles is much smaller than the number of users). Using all combinations (referenced as $\mathcal{X}$)

[5]A large-scale dataset is required to generate machine learning models for the classification-based analysis approach.

[6]The generator randomly creates at maximum n instances of an entity where n is specified by the user of the generator.

of roles, object, and actions, we randomly select a subset $\mathcal{Y} \subset \mathcal{X}$ to generate a set of transactions $\mathcal{Y}$. Similarly, we randomly select a subset $\mathcal{Z} \subset \mathcal{X}$ to generate a set of access control policies $\mathcal{Z}$ and the sign of each policy is randomly chosen as positive or negative[7]. Finally, the generator randomly assigns a frequency counter to each transaction. As a result, the generated dataset intentionally includes all types of low-quality policies. The subset of transactions $\mathcal{Y} - (\mathcal{Z} \cap \mathcal{Y})$ indicates missing policies (i.e., incompleteness). Meanwhile, the subset of policies $\mathcal{Z} - (\mathcal{Y} \cap \mathcal{Z})$ is irrelevant. The subset of policies which have corresponding transactions $(\mathcal{Y} \cap \mathcal{Z})$ can potentially include inconsistencies, redundancy, and exceptions. Table 3.7 shows the distribution of the low-quality policies among the three generated datasets.

Table 3.7.: The Distribution of Low-Quality Policies among Datasets

	Datasets		
	DS1	*DS2*	*DS3*
Inconsistency	162	368	478
Exceptions	2,836	5,762	12,062
Incompleteness	1,672	3,052	8,755
Redundancy	787	962	1,507
Irrelevancy	1,398	1,940	3,896

Classification approach settings: For the classifiers adopted in our classification-based analysis, we used the Java Weka library [75]. All classifiers are trained on 70% of the dataset using 10-fold cross-validation. Regarding the kNN classifier, we used the instance-based learner algorithm. For the decision tree classifier, we used the C4.5 Decision algorithm. Overall, we used the Weka default values for the parameters of the machine learning techniques (e.g., for kNN classifier, k = 1).

We implemented the prototype infrastructure of ProFact in Java 1.7. All experiments were performed on high-performance computing clusters at Purdue Research

[7]The generator verifies that $\mathcal{Y} \cap \mathcal{Z} \neq \phi$ to guarantee having all low-quality policy types.

Center. The analysis prototype was run on a cluster of one node with 16 cores and 64GB memory.

3.6.2 Pre-processing Time for the Analysis Approaches

We collected the total time for building the underlying structures for the structure-based approach[8]. Fig. 3.4 shows the construction time (in base-10 logarithmic scale) for the three variants (i.e., policy-based, transaction-based, and PT-based) of the structure-based approach using the three datasets. In general, the construction of the structure for the transaction-based analysis takes longer time than that of the policy-based analysis because of two reasons: a) the number of transactions in a system usually is larger than the number of access control policies, and b) adding a transaction to the transaction tree involves finding its corresponding policy in the policy tree to link them together. Moreover, the construction time for PT-tree is almost similar to that of the transaction tree.

Fig. 3.4.: Construction Time for Structure-based Approaches

[8]We discarded the training time for every classifier of the classification-based analysis approach because training a classifier is an off-line step. Meanwhile, the construction of the tree structures can not be considered off-line because they should be maintained up to date.

3.6.3 Analysis Results

Structure-based Analysis

Performance: Fig. 3.5 shows the analysis time for detecting inconsistent and redundant policies[9] using the policy-based, transaction-based, and PT-based approaches across the three datasets. In general, the transaction-based approach has the best performance compared to the policy-based approach. In particular, the transaction-based approach has a speedup factor of 4x, 5x, and 3x with respect to the policy-based approach using *DS1*, *DS2*, and *DS3*, respectively. All approaches are based on the tree traversal but with different analysis mechanisms. The transaction-based approach only visits every path in the tree and utilizes the associated link to fetch the corresponding policy in the policy tree. Meanwhile, the policy-based approach searches for all policies that involve similar components while inspecting each policy. The transaction-based approach invested the time spent on constructing the transaction tree and linking it with the policy tree to achieve better performance at the analysis phase. Since the PT-based approach uses the hybrid tree structure that stores both transactions and policies, inspecting all policies and corresponding transactions thoroughly through such a structure leads to extra time for the analysis compared to both the policy-based and transaction-based approaches.

Fig. 3.5.: Analysis Time for Structure-based Approaches

[9]We discarded the other types of low-quality policies and considered only the types of policies that can be detected by the three variants of the structure-based approach.

Efficiency: Both policy-based and transaction-based approaches were able to detect all inconsistent and redundant policies. Since irrelevant policies do not have corresponding transactions in the transaction tree, the irrelevant policies were reported by only the policy-based approach. Nonetheless, the policy-based approach was not able to detect the exceptions and the incomplete policies as these can only be detected by analyzing the transactions executed in the system as well as their corresponding access control policies. Thus, such cases are detected by the transaction-based approach. However, the PT-based approach was able to detect all of these cases.

Classification-based Analysis

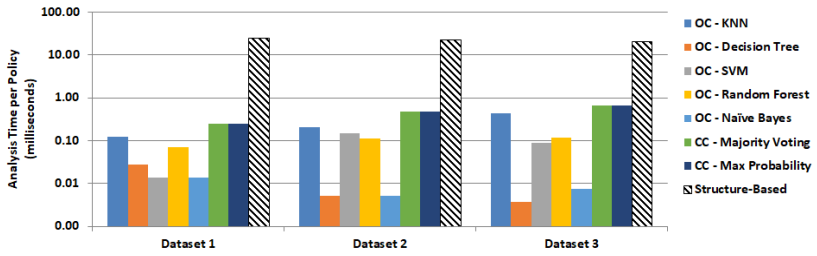


Fig. 3.6.: Analysis Performance for Classification-based Approaches

Performance: Fig. 3.6 shows the average analysis time per policy (in base-10 logarithmic scale) using both analysis approaches (structure-based and classification-based)[10] across the three datasets. In general, all schemes of the classification-based analysis approach (i.e., *OC* and *CC*) outperform the structure-based analysis approach. The analysis time of the classification-based approach using the *OC* scheme varies based on the adopted classifier. In particular, *OC* with the Naïve Bayes and Decision Tree classification algorithms have the best performance. Among all classifiers used with *OC*, the kNN and Random Forest classifiers have the worst performance

[10]The analysis time of the structure-based analysis represents the PT-based one since it is able to detect all types of low-quality policies.

since the kNN classifier requires retrieving all closest objects among the training dataset and Random Forest investigates multiple internal decision trees to conclude the classification result. Intuitively, CC is slower than OC because CC performs the analysis through all classifiers adopted by OC to obtain the final classification results. However, both approaches based on the CC scheme (i.e., Majority Voting and Max Probability) outperform the structure-based approach. In particular, the speedup factor of CC is 31x with respect to the structure-based approach.

Efficiency: To evaluate the efficiency of the classification-based analysis approach, we report the recall, precision, and accuracy values which are formulated in Eqs. 3.1 - 3.3. For calculating recall, precision, and accuracy, we used the output of the structure-based analysis results as the ground truth.

$$Recall = \frac{TP}{TP + FN} \tag{3.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

Figs. 3.7, 3.8, and 3.9 show the recall, precision, and accuracy of both classification schemes (OC and CC) across the three datasets, respectively. In general, using *DS3*, all classification approaches achieve recall and precision values above 70%. Meanwhile, Naïve Bayes was the only one that does not achieve an accuracy value above 70%. Moreover, the efficiency of OC varies based on the trained model and the chosen classifier. For example, Random Forest has the worst recall, and accuracy values using *DS1* and *DS2*, but it has been improved in *DS3*. Consequently, the OC scheme is not reliable for our framework. On the other hand, the CC scheme achieves the best recall, precision, and accuracy compared with OC using all datasets. In particular, the CC scheme achieved at maximum recall, precision, and accuracy of 88%, 87%, and 91%, respectively. This supports our claim that using the combined scheme CC for classification is preferable than using the OC scheme. The CC scheme with

Majority Voting and Max Probability techniques achieved almost similar efficiency. However, Majority Voting was the best with a minor difference.

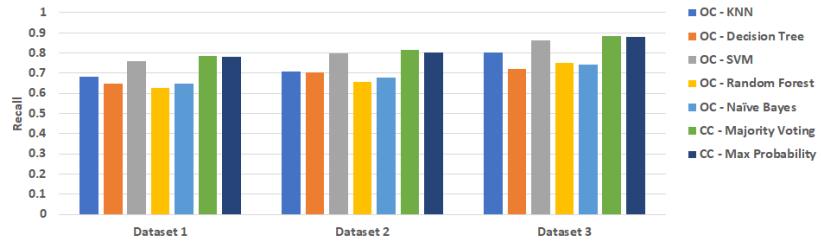


Fig. 3.7.: Analysis Efficiency (Recall) for Classification-based Approaches

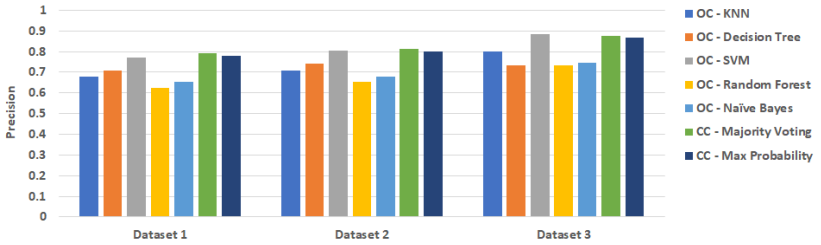


Fig. 3.8.: Analysis Efficiency (Precision) for Classification-based Approaches

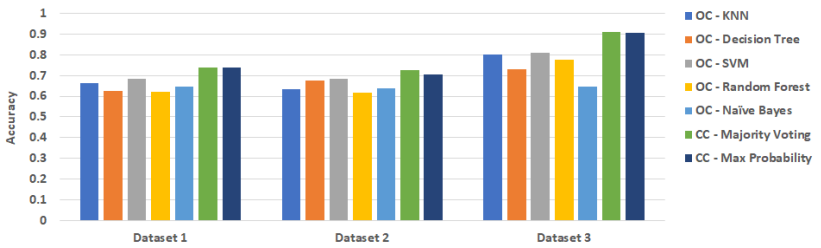


Fig. 3.9.: Analysis Efficiency (Accuracy) for Classification-based Approaches

3.7 Related Work to the ProFact Framework

The area of policy analysis has been widely investigated. Approaches to policy analysis use various methods and are characterized by different goals as we discuss below.

3.7.1 Goals for Policy Analysis

The goals of policy analysis mainly fall into two directions: assessing the fulfillment of a set of quality requirements, and designing and organizing a set of policies.

Policy Quality Assessment: Past work on policy quality requirements have focused on some of the quality requirements that we have introduced. Among these requirements, consistency was the most investigated one. Gupta et al. [20] and Cau et al. [21] focused on detecting policies inconsistencies in the RBAC domain. Meanwhile, Mankai et al. [76] and Turkmen et al. [22] proposed methods to detect inconsistency among XACML policies. Regarding other requirements, redundancy has been evaluated using various approaches such as the ones proposed by Ngo et al. [23], Hadj et al. [77] and Pina et al. [78], while incompleteness is assessed using other research approaches (e.g., [24, 79, 80]). To the best of our knowledge, our approach is the first to assess access control policies with respect to new types of quality requirements (i.e., exceptions and irrelevancy). Because of incorporating provenance metadata, our framework is able to aggregate information about system behavior at execution time, and thus it is able to address all quality requirements.

Policy Design and Organization: For properly re-organizing and evolving policy sets, it is often important to assess the similarity of different policies and the impact of policy modifications. Policy similarity refers to the technique for characterizing the relationships between policies and the actions authorized by them. Several researchers focused on policy similarity including Kolovski et al. [81], Lin et al. [82,83], Craven et al. [84,85], and Mazzoleni et al. [86,87]. Change impact analysis on the policy set evaluates the changes between two versions of a policy by providing

a set of counterexamples that illustrate semantic differences between the two policies. Several research efforts have been devoted to investigating the change impact analysis (e.g., [22, 88, 89]). Our framework includes the policy evolution services which analyzes the impact of changing one of the policies on the quality of the correlated policies of the changed policy. However, for analyzing policies, our framework utilizes the matching metric which is a simple similarity policy for identifying the correlated policies. Our framework can be enhanced by utilizing other similarity metrics.

3.7.2 Methods for Policy Analysis

Various methods have been proposed for policy analysis including formal methods, model checking, data mining, and structure-based. Some policy analysis approaches utilizing formal methods techniques such as reasoning [90] [91], and argumentation [92–94]. Moreover, several approaches and frameworks for policy analysis have been developed using model checking techniques such as SAT solver [83, 95], or SMT solver [22, 96], and binary decision diagrams [21, 78, 83, 97]. Regarding data mining methods, Shaikh et al. [79, 98, 99] and Aqib et al. [100] proposed several approaches for policy analysis using decision tree classifier while Bauer et al. [101] proposed an approach to reorganize the RBAC policies using association rule mining method. Furthermore, structure-based methods were adopted for analyzing access control policies. For example, Xu et al. [102], Staniford et al. [103], Alves et al. [104] used adapted versions of the graph data structure. Meanwhile, the tree structure was used for firewall policy analysis [105, 106]. To the best of our knowledge, our approach is the first to utilize tree-based structures for analyzing access control policies. In addition, our framework proposes another analysis approach based on various classification methods, other than decision trees.

4. PROWS: PROVENANCE-BASED SCIENTIFIC WORKFLOW SEARCH FRAMEWORK

4.1 Queries on Scientific Workflows

In what follows, we formally define the primary concepts and queries used in the subsequent discussion.

Definition 4.1.1 (Scientific Workflow) *A scientific workflow w is defined by two components: a set of metadata, denoted by $w.M$, and a graph, denoted by $w.G$. $w.M$ is a tuple of n attributes $\langle a_0, a_1, \ldots, a_{n-1} \rangle$; the value of each attribute a_i is evaluated by its corresponding function denoted by $\zeta_i(a_i)$. $w.G$ is a connected directed graph consisting of the following components:*

- *N, E, T, P, L refer to the set of nodes, edges, node types, edge types, and node labels, respectively;*
- *α: $N \rightarrow T$ is the node type function;*
- *β: $N \rightarrow L$ is the node label function; and*
- *γ: $E \rightarrow N \times N \times P$ is the node connectivity function which connects two nodes $N \times N$ with an edge type P.*

Definition 4.1.2 (Scientific Workflow Dataset) *A workflow dataset $\mathcal{D}$ is defined as a set of m scientific workflow graphs (i.e., $\mathcal{D} = \{w_0, w_1, \ldots, w_{m-1}\}$) where each workflow w_i in $\mathcal{D}$ is represented by a pair ($w_i.G$, $w_i.M$) defined according to Definition 4.1.1.*

Hereafter, we formally define the notion of the composite query to search a scientific workflow dataset. Such a query is composed of three subqueries: workflow

metadata subquery, workflow node labels subquery, and workflow structure subquery. A composite query should include at least one subquery.

Definition 4.1.3 (Metadata-based Workflow Search Query) *A metadata-based query Q_M is defined as:*

$Q_M(S, \mathcal{D}) \Leftrightarrow \{w_i \in \mathcal{D} \mid \exists a_j \in w_i.M \wedge \Theta(\bowtie, \zeta_j(w_i.M.a_j), v_j) \wedge (a_j, v_j) \in S\}$.

Q_M is represented by a set of pairs $S = (a_0, v_0), (a_1, v_1), \ldots, (a_{n-1}, v_{n-1})$ where a_i is the query metadata attribute and v_i is the attribute value. Θ is an attribute matching function which compares the value of a workflow metadata attribute ζ_j $(w_i.M.a_j)$ with a query attribute value v_j using a comparison operator denoted as $\bowtie$. Without loss of generality, in this chapter $\bowtie$ is an equality (i.e., $=, \neq$) or relational (i.e., $>, <, \leq, \geq$) operator.

Definition 4.1.4 (Label-based Workflow Search Query) *A label-based query Q_L is defined as:*

$Q_L(S, \mathcal{D}) \Leftrightarrow \{w_i \in \mathcal{D} \mid \exists n \in w_i.N \wedge \alpha(n) = t_j \wedge t_j \in w_i.T \wedge \beta(n) \subseteq l_j \wedge l_j \in w_i.L \wedge (t_j, l_j) \in S\}$.

Q_L is represented by a set of pairs $S = (t_0, l_0), (t_1, l_1), \ldots, (t_{n-1}, l_{n-1})$ where t_j is the query node type and l_i is the query node label.

Definition 4.1.5 (Pattern-based Workflow Search Query) *A pattern-based query Q_P is defined as:*

$QP(N', E', T', P', L', \alpha', \beta', \gamma')$ *which is a subgraph similarity search query for retrieving a workflow $w_i \in \mathcal{D}$. Checking if $Q_P \subseteq w_i$ is performed using a mapping function f: $Q_P.N \Rightarrow w_i.N$ such that:*

$\exists n'_i, n'_j \in Q_P.N', \gamma'(n'_i, n'_j) \in Q_P.E' \rightarrow \alpha'(n'_i) = \alpha(f(n'_i)) \wedge \alpha'(n'_j) = \alpha(f(n'_j)) \wedge \beta'(n'_i) \subseteq \beta(f(n'_i)) \wedge \beta'(n'_j) \subseteq \beta(f(n'_j)) \wedge \gamma(f(n'_i), f(n'_j)) \in Q_P.E'$.

An example of a pattern-based query is shown in Fig. 4.1. The pattern query is composed of three data nodes (medical documents, extracted proteins, and extracted diseases per protein) and two operation nodes (extract protein and link proteins to

diseases) and four edges. In this example, we color-coded the nodes based on type (i.e., data nodes in red and operation nodes in blue).

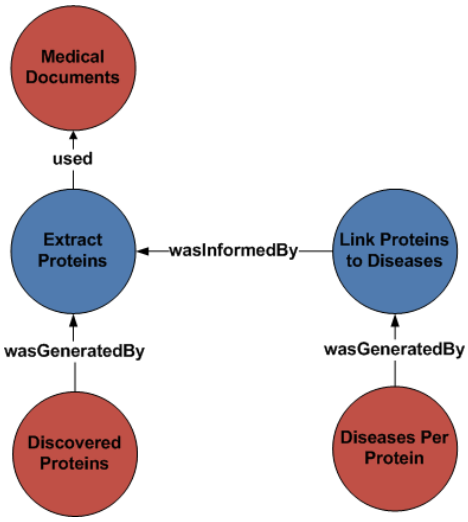


Fig. 4.1.: An Example of Pattern-based Workflow Search Query

To rank candidate workflows based on the predicates of Q_M and Q_L, we use the metrics in Eqs. 4.1 and 4.2, respectively. In both equations, #matches is the number of attributes or labels found in the workflow, and #mismatches is the number of attributes or labels which are not found in the workflow. To rank candidates workflows based on the pattern predicate provided by Q_P, we use the Most Common Subgraph Algorithm (MCS) [107] in the metric defined by Eq. 4.3. MCS calculates the maximum number of matched edges. Eqs. 4.1, 4.2, 4.3 are derived from the Jaccard similarity coefficient metric; thus the outcome of the similarity metric is a normalized value (i.e., between 0 and 1).

$$sim_M = \frac{\#matched\ attributes}{\#matched\ attributes + \#mismatched\ attributes} \tag{4.1}$$

$$sim_L = \frac{\#matched\ labels}{\#matched\ labels + \#mismatched\ labels} \tag{4.2}$$

$$sim_P = \frac{mcs(w, Q_p)}{|w.E| + |Q_p.E| - mcs(w, Q_p)} \tag{4.3}$$

If a composite query contains more than one subquery (i.e., Q_M, Q_L, or Q_P), evaluating such a composite query leads to the separate evaluation of each subquery. Such an evaluation produces individual results; hence the outcome of the composite query is the intersection of the results of the individual subqueries. In this section, we discuss the provenance-workflow transformation, indexing, and querying stages.

4.1.1 Provenance to Workflow Transformation

As discussed earlier, a key phase of our approach is the transformation phase which constructs the workflow repository based on the provenance repository. The transformation phase first groups all provenance graphs based on the *WorkflowID* attribute of Processes. To avoid any loss of information during transformation, each set of provenance graphs having the same *WorkflowID* is transformed into a workflow graph $w_i.G$ based on the modeling mapping shown in Table 4.1. In the current version of our approach, each workflow has the metadata attributes ($w_i.M$) shown in Table 4.2. Extracting $w_i.M$ requires a thorough analysis of provenance graphs. For example, through provenance, we can calculate the number of provenance graphs corresponding to a workflow (i.e., the "*Usage Frequency*" attribute). Furthermore, the list of the actors, who performed the processes of the provenance graphs corresponding to a workflow, defines the "*Users*" attribute. In addition, a workflow might be derived from another workflow; hence provenance enables us to locate the source workflow (i.e., *DerivedFrom*) and hence *CreatedBy* and *CreationTime* are subsequently evaluated.

4.1.2 Indexing and Querying

Retrieving workflows based on a composite query (Q_M, Q_L, and Q_P) comprises two consecutive steps: searching and ranking. The searching step utilizes two strategies to find similar workflows: linear-scan strategy (referred to as *Naïve*) and index-based

Table 4.1.: Modeling Workflows From Provenance Repository

	Provenance Model	Workflow Model
Nodes	Process	N; α = Process
	Operation	N; α = Operation
	Data	N; α = Data
	Environment	N; α = Environment
Edges	Used	E; γ = (Process $\vee$ Operation) x Data x used
	WasGeneratedBy	E; γ = Data x (Process $\vee$ Operation) x WasGeneratedBy
	WasDerivedFrom	E; γ = Data x Data $\times$ WasDerivedFrom
	WasInformedBy	E; γ = Operation x Operation x WasInformedBy
	wasEncapsulatedBy	E; γ = Operation x Operation x wasEncapsulatedBy
	wasPartOf	E; γ = Operation x Process x wasPartOf
	wasForkedBy	E; γ = Process x Process x wasForkedBy
	wasInContext	E; γ = Process x Environment x wasInContext

Table 4.2.: Attribute List of Workflow Metadata

Attribute	Description
Description	Workflow purpose.
Usage Frequency	The number of times a workflow was executed.
Users	The list of users who carried out the various tasks of a workflow.
CreatedBy	The user who defined a workflow.
CreationTime	Workflow creation time.
DerivedFrom	The workflow which originates a workflow.

strategy. Finally, the ranking step sorts the retrieved workflows with respect to their relevance to the query.

Searching Strategies

A. Naïve Search Strategy

Under this approach, each workflow $w_i \in \mathcal{D}$ is evaluated with respect to Q_M, Q_L, and Q_P individually as shown in Algorithm 4.1. The goal of each subquery is to find similar workflows based on the query criteria. Hence, each subquery produces a similarity score separately. A workflow is a candidate for the composite query if the workflow potentially satisfies the criteria of each provided subquery (i.e., the similarity score of each subquery is above zero). Evaluating each subquery requires different algorithms; for Q_P the workflow graph ($w_i.G$) is traversed using the depth-first search algorithm to find MCS; for Q_L only the list of workflow graph nodes (i.e., $w_i.G.N$) is traversed; and for Q_M no graph traversal is required in that only the workflow attributes must be evaluated against the predicates (i.e., $w_i.M$).

B. Index-Based Search Strategy

To speed-up searches on the workflow dataset $\mathcal{D}$, we adopt an approach based on the use of multiple index structures for each subquery. The main goal of an index structure is to minimize the number of workflows to be traversed (i.e., search space reduction). For efficiently executing Q_M, an individual B-tree index is built for each attribute. For Q_L or Q_P, we adopt three forms of inverted index structures: Label-based Inverted Index (LII), Node-based Inverted Index (NII), and Edge-based Inverted Index (EII). The LII, NII, and EII structures are adapted versions of the Inverted Index structure widely used for document search. In document search, an inverted index (see the example in Fig. 4.2) is a collection of keys where each key is a word and the list of the identifiers of the documents containing this word [108].

The keys are referred to as keywords, and the associated lists are known as inverted lists. In our index structures, we change the form of keys and inverted lists to enhance the efficiency of workflow searches. In what follows we explain the construction and searching mechanism of each index structure in detail.

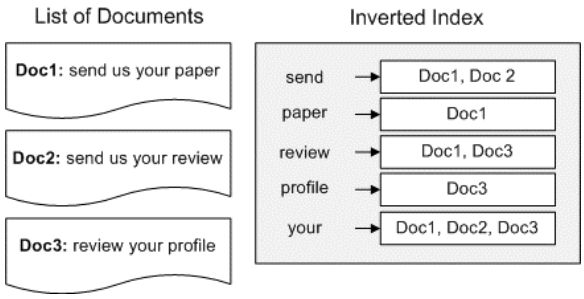


Fig. 4.2.: Example of Inverted Index Structure

Constructing LII, NII, and EII: The distinction among these structures is the type of keys. In LII, the individual tokens which compose the labels of workflow nodes (i.e., $\forall\ k \in w_i.l_j \mid l_j \in w_i.L \wedge w_i \in \mathcal{D}$) are used as keys. In NII, the keys are the sets of the workflow nodes (i.e., $\forall\ n \in w_i.N \mid w_i \in \mathcal{D}$). Finally, in EII the keys are the lists of edges composing the workflows (i.e., $\forall\ e \in w_i.E \mid w_i \in \mathcal{D}$). In all of these structures, the inverted list of each key (i.e., a node label token in LII, a node in NII, or an edge in EII) is the list of workflows which contain the corresponding key.

Constructing B-trees: The B-tree structure is implemented efficiently in any commercial relational database management system. Hence, we store the metadata tuples for all workflows in a relational table where we allocate a B-tree index on each column (i.e., on each attribute of the workflow metadata).

Querying LII: These structures support the evaluation of Q_L and Q_P. Algorithms 4.2 and 4.3 show the usage of LII for processing Q_L, and Q_P, respectively. In both queries, we first query the index structure for each label in $Q_L.S$ or $Q_P.L$' to retrieve an intermediate set of workflows which potentially satisfy the query criteria. Then, we estimate the similarity metric (i.e., sim_L with Q_L or sim_P with Q_P) for each

workflow in the intermediate set. LII might retrieve "false-hit" workflows because the keys of LII do not contain the type of node that contains such keys. The additional "false-hit" workflows require an extra cost for traversing them to validate the relevancy to the query criteria.

Table 4.3.: Types of Retrieved Workflows with Q_L and Q_P using Different Index Structures

Indexes	Q_L Workflows		Q_P Workflows	
	Similar	"false-hit"	Similar	"false-hit"
LII	✓	✓	✓	✓
NII	✓	✗	✓	✓
EII	✓	✗	✓	✗

Querying NII: Like LII, NII supports both Q_L and Q_P. Evaluating these queries with NII is analogous to evaluating the queries with LII, but NII avoids the retrieval of "false-hit" workflows when querying Q_L. Each key of NII is a pair consisting of a node label and the node type; hence it is sufficient to validate all query criteria provided by Q_L. When querying Q_P, NII does not only query for certain node but also queries about the relations among these nodes. Thus, querying Q_P with NII might result with "false-hit" workflows as NII does not maintain all the information required the query evaluation.

Querying EII: Like LII and NII, EII supports both Q_L and Q_P. Evaluating these queries with EII is analogous to evaluating them with LII and NII, but EII avoids the retrieval of "false-hit" workflows when querying both Q_L and Q_P. Each key of EII is composed of an edge connecting two nodes where each node is represented by a pair of label and the node type; hence it is sufficient for retrieving the workflows that satisfy the query criteria of Q_L and Q_P with the minimum relevancy. When querying Q_P, NII does not only query for certain node but also queries about the relations

among these nodes. Thus, querying Q_P with NII might result in "false hit" workflows as it does not maintain all the required information for the query evaluation. Table 4.3 summarizes the types of retrieved workflows with different index structures when querying Q_L and Q_P.

Querying B-trees: This type of index structures is used to enhance the evaluation of Q_M. The similarity of each retrieved workflow by Q_M is calculated by the similarity function sim_M.

Ranking Search Results

Each subquery (i.e., Q_M, Q_L, and Q_P) retrieves an individual set of workflows. The final result of the composite query is the intersection of the results of the subqueries. Each workflow in the final result is associated with three similarity scores (i.e., generated by sim_M, sim_L, and sim_P). Hence, to rank the workflows in the final result set, we use the weighted combined score given by Eq. 4.4. This combined score uses three different weights (ψ_M, ψ_L, and ψ_P) which reflect the impact of each subquery on the workflow relevancy. Estimating these weights requires a thorough user evaluation which is out of the scope of this work. For simplicity, we assume that all the subqueries have the same weight.

$$sim(w_i, Q_M, Q_L, Q_P) = \psi_M * sim_M + \psi_L * sim_L + \psi_P * sim_P \tag{4.4}$$

4.2 Experiments

4.2.1 Experimental Methodology

Dataset: To evaluate the performance of our searching strategies (i.e., *Naïve*, LII, NII, and EII), we conducted several experiments using two scientific workflows datasets: one of them is a real dataset (referred to as REAL) extracted from the CRIS (Computational Research Infrastructure for Science [31]) testing environment, and the other one is a synthetic dataset (referred to as SYN) generated by a random dataset

Algorithm 4.1 Naïve Search Strategy

Require: $\mathcal{D}$, Q_M, Q_L, Q_P

1: $\mathcal{R} = \{\}$
2: **for** $\forall\ w_i \in \mathcal{D}$ **do**
3: $\quad s_M = sim_M(w_i.M,\ Q_M.M)$
4: $\quad s_L = sim_L(w_i.G,\ Q_L.S)$
5: $\quad s_P = sim_P(w_i.G,\ Q_P.P')$
6: $\quad$ **if** $(s_M > 0 \wedge s_L > 0 \wedge s_P > 0)$ **then**
7: $\quad\quad \mathcal{R} = \mathcal{R} \cup \{w_i\}$
8: $\quad$ **end if**
9: **end for**
10: **return** $\mathcal{R}$

Algorithm 4.2 Q_L with Label-based Inverted Index

Require: $\mathcal{D}$, LLI, Q_L

1: $\mathcal{R}_L = \{\}$, $\mathcal{R} = \{\}$
2: **for** $\forall\ l_j \in Q_L.S$ **do**
3: $\quad \mathcal{R}_\mathcal{L} = \mathcal{R}_\mathcal{L} \cup search(LII, l_j)$
4: **end for**
5: **for** $\forall\ w_i \in \mathcal{R}_\mathcal{L}$ **do**
6: $\quad s_L = sim_L(w_i,\ Q_L.S)$
7: $\quad$ **if** $(s_L > 0)$ **then**
8: $\quad\quad \mathcal{R} = \mathcal{R} \cup \{w_i\}$
9: $\quad$ **end if**
10: **end for**
11: **return** $\mathcal{R}$

generator to scale the size of the workflow experimental dataset. Table 4.4 shows the size of each dataset in terms of the number of workflows, nodes, and edges.

Table 4.4.: Scientific Workflow Datasets

	#Workflows	#Nodes	#Edges
REAL	30	600	2.7K
SYN	1000	18K	95K

Queries and Metrics: For each subquery type, we report two metrics; query time (in milliseconds) and the percentage of traversed workflows (see Eq. 4.5). To avoid caching effects on timing, each query was executed five times and the longest and shortest times were ignored and the query time is the average of the remaining times. We performed a set of queries, and we report the average of both query time and the percentage of traversed workflows.

$$Percentage of traversed workflows = \frac{\#traversed workflows}{\#Total workflows} \tag{4.5}$$

Settings: We stored workflow metadata in MySQL 5.7, and allocated a B-tree index on each column. All experiments were performed on a 3.6 GHz Intel Core i7 machine with 12 GB memory running on 64 bit Windows 7.

4.2.2 Search Performance

Label-based Query Evaluation

Query Time: Fig. 4.3 shows the query time (in base-10 logarithmic scale) for label-based queries (Q_L) using the REAL and SYN datasets. In general, the *Naïve* approach has worse performance compared to the index-based approaches, and among the index-based approaches, NII has the best performance. In particular, NII achieved a speedup factor of $34x$ and $23x$ with respect to the *Naïve* approach on

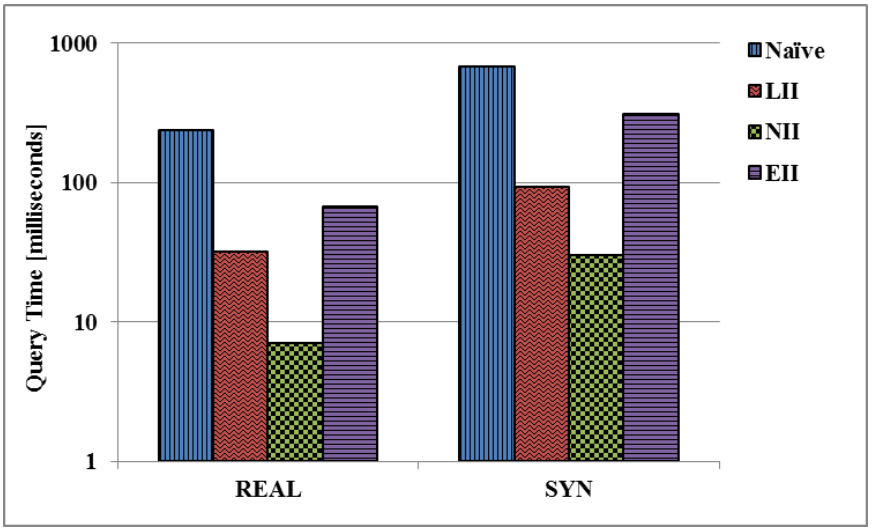


Fig. 4.3.: Average Query Time for Label-based Queries

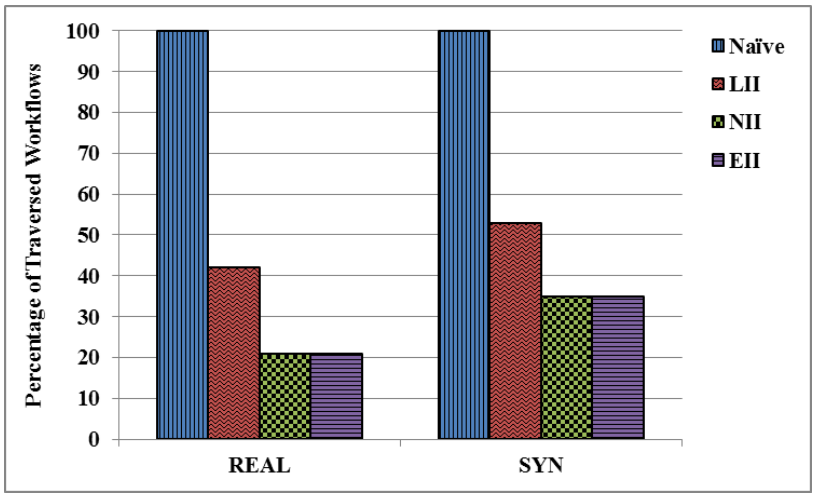


Fig. 4.4.: Average Percentage of Traversed Workflows for Label-based Queries

REAL and SYN, respectively. Furthermore, LII performed worse than NII because unlike NII, the LII structure does not provide information about node types. We recall that the query predicate of Q_L includes both keywords and node types. Moreover, EII incurred the worst performance compared to both LII and NII, because each key in the EII structure is composed of an edge connecting a source and destination nodes; thus, Q_L had scanned all keys in EII sequentially to inspect the source and destination nodes.

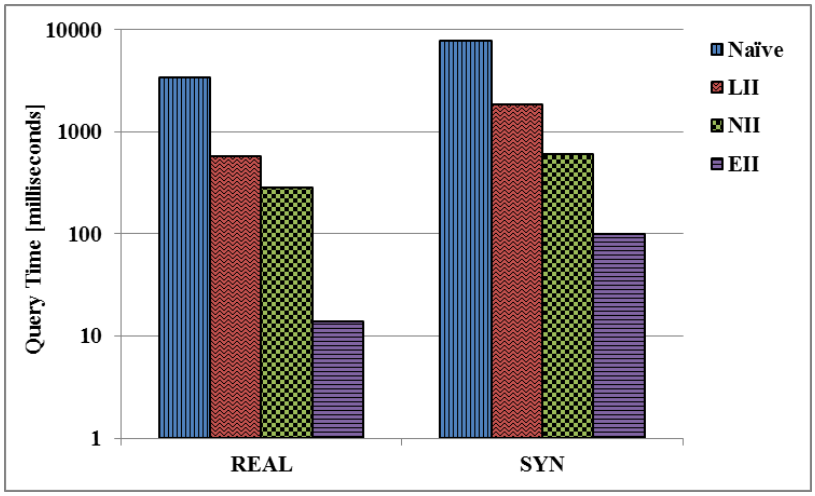


Fig. 4.5.: Average Query Time for Pattern-based Queries

Percentage of Traversed Workflows: Fig. 4.4 shows the percentage of traversed workflows for label-based queries (Q_L) using the REAL and SYN datasets. The *Naïve* approach is not able to reduce the search space; thus, it traversed all workflows (i.e., 100%) on both REAL and SYN. Meanwhile, the index-based approaches successfully reduced the search space by traversing only a subset of scientific workflows. Among the index-based approaches, LII has the highest percentage of traversed workflows compared to NII and EII because LII traverses some workflows which are potentially "false hit" when querying Q_L. Furthermore, NII and EII have the same percentage of traversed workflows because when querying Q_L, both of them reduced the search space and avoided "false hit" workflows. The difference in key structures in NII and EII only affects the time for locating the workflows that are

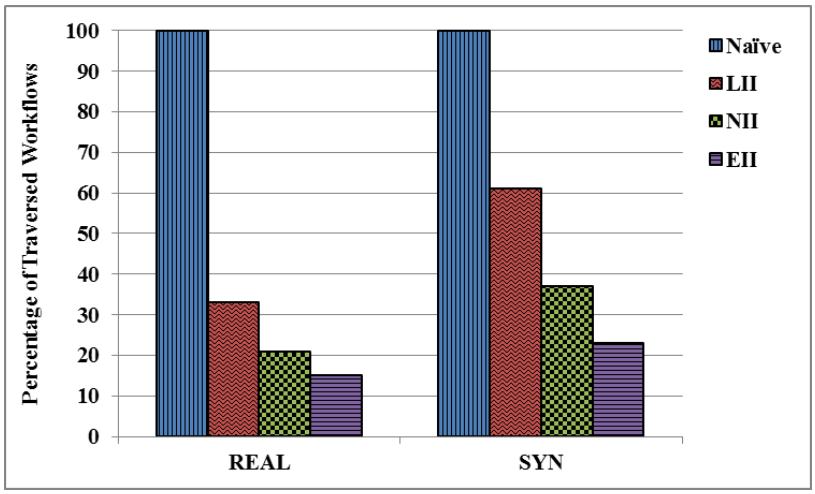


Fig. 4.6.: Average Percentage of Traversed Workflows with Pattern-based Queries

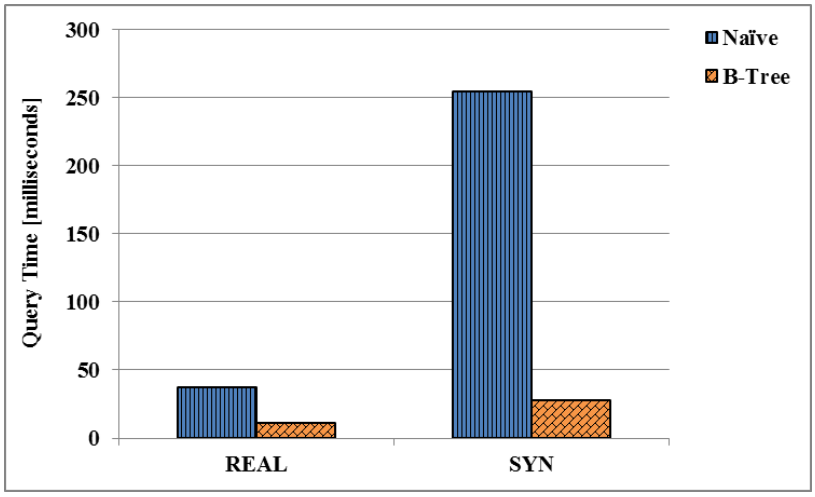


Fig. 4.7.: Average Query Time for Metadata-based Queries

potentially similar to the query predicate. Nonetheless, such a difference does not influence the percentage of traversed workflows.

Pattern-based Query Evaluation

Query Time: Fig. 4.5 shows the query time (in base-10 logarithmic scale) for

Algorithm 4.3 Q_P with Label-based Inverted Index

Require: $\mathcal{D}$, LLI, Q_P

1: $\mathcal{R}_P = \{\}$, $\mathcal{R} = \{\}$
2: **for** $\forall\ l_j \in Q_L.S$ **do**
3: $\quad \mathcal{R}_\mathcal{P} = \mathcal{R}_\mathcal{P} \cup search(LII, l_j)$
4: **end for**
5: **for** $\forall\ w_i \in \mathcal{R}_\mathcal{P}$ **do**
6: $\quad s_P = sim_P(w_i, Q_P.P')$
7: $\quad$ **if** $(s_P > 0)$ **then**
8: $\quad\quad \mathcal{R} = \mathcal{R} \cup \{w_i\}$
9: $\quad$ **end if**
10: **end for**
11: **return** $\mathcal{R}$

pattern-based queries (Q_P) using the REAL and SYN datasets. In general, the index-based approaches outperformed the *Naïve* approach. As opposed to Q_L, EII has the best performance among the index-based approaches when querying Q_P because the keys in EII comprise the segments of structures of workflow graphs. In particular, EII showed a speedup factor of $12x$ and $13x$ with respect to the *Naïve* approach on REAL and SYN, respectively. When querying Q_p, NII has worse performance when compared with EII because the keys in NII comprise only the nodes of the workflow graphs but do not comprise the dependency relations between nodes; thus, NII traversed a higher number of "false hit" workflow. Furthermore, the keys in LII

do not contain nodes or edges; thus, LII had worse performance compared to both NII and EII.

Percentage of Traversed Workflows:

Fig. 4.6 shows the percentage of traversed workflows for pattern-based queries (Q_P) using the REAL and SYN datasets. Like in the case of Q_L, the *Naïve* approach is not able to reduce the search space; thus, it traversed all workflows (i.e., 100%) on both REAL and SYN. Among the index-based approaches, EII requires traversing the smallest percentage of workflows compared to both NII and LII since the keys in EII contain sufficient information to validate the query predicates — thus it had not traversed any "false hit" workflows. On the other hand, LII requires traversing a higher percentage of workflows compared to NII because LII traverses some workflows which are potentially "false hit" when querying Q_P due to the lack of the sufficient information to evaluate the query predicates.

Metadata-based Query Evaluation

Query Time: Fig. 4.7 shows the query time for metadata-based queries (Q_M) using the REAL and SYN datasets. In general, the B-Tree-based approach outperformed the *Naïve* approach. The B-tree-based approach achieved speedup factors of $3x$ and $9x$ with respect to the *Naïve* approach on REAL and SYN, respectively.

We report only the query time for Q_M because the B-Tree index is not our proposed index structure.

In summary, the index-based approaches enhance searching workflows with all subqueries (i.e., Q_M, Q_L, and Q_P). When querying Q_L, NII can outperform the other index structures, while when querying Q_P, EII achieves the best performance compared to the other index structures.

4.3 Related Work to the ProWS Framework

Modeling Workflows: Various representations for workflows have been proposed based on the application domain. For representing business process workflows, multiple languages have been proposed such as Business Process Modeling Notation (BPMN) [109], Event-driven Process Chain (EPC) [110], Business Process Execution Language (BPEL) [111], and Yet Another Workflow Language (YAWL) [112]. For software model workflows, Web Modeling Language (WebML) [113] has been proposed. With respect to scientific workflows, the standard PROV model has recently been extended to represent workflows using the ProvOne model [114]. In our work, we use the SimP provenance model because it is interoperable with other standard provenance models (i.e., PROV and OPM). In each provenance-based scientific workflow representation, each edge is typically associated with a type attribute; hence the query languages for searching workflows should be adapted according to these models to consider edge types.

Querying Workflows: Two categories of query languages have been proposed for searching workflows: keyword-based and graph-based languages. Among the languages in the first category, WISE [115] enables extracting business process workflow models based on keyword matching whereas myExpreiment [36], Galaxy [116], and CrowdLab [117] are examples of keyword-based search engines for scientific workflows. For the languages in the second category, the focus has been on the design of data structures and methods to enhance the efficiency of query processing, such as index-based structures [118], a clustering-based method [119], and more complex techniques combining Grid structures with auxiliary inverted index structures [30]. While several approaches have been proposed for searching scientific workflows by keyword-based queries (e.g., myExperiment) or graph-based queries (e.g., [120], [121], [122], [123], [124], [125], [126]), all those approaches neither utilize provenance to model workflows nor support metadata-based queries. To the best of our knowledge, the search approach closest to ours is PBase [127] which is provenance-based scientific workflow

search using PROV as provenance model and ProvOne as workflow model. However, PBase only supports keyword-based and graph-based queries — thus, it does not support metadata-based queries. Additionally, PBase does not address search efficiency.

Searching Graphs: Different graph indexing algorithms use different graph features as indexes. For example, GraphGrep [128] is an index on paths, gIndex [129] is an index on frequent and discriminative subgraphs, TreePi [130] is an index on frequent subtrees, and FG-Index [131] is an index on subgraphs. Zhao et al. [132] proposed using frequent tree structures and a small number of discriminative subgraphs as indexing features. Furthermore, Zou et al. [133] designed an approach which first builds a summary of sub-structures, and then constructs an index on them, while the approach by Williams et al. [134] first enumerates all induced subgraphs stored in the database and then organize them w.r.t. cross-index hash for efficient lookup. Additionally, the closure-tree indexing technique [135] supports subgraph and similarity queries over generalized graph representations. All of these approaches rely on abstracting graphs with one type of node, while our indexing approaches consider a provenance-based model which has various types of nodes and edges.

5. POLISMA - A FRAMEWORK FOR LEARNING ATTRIBUTE-BASED ACCESS CONTROL POLICIES

5.1 Background and Problem Description

In what follows, we introduce background definitions for ABAC policies and access request examples, and formulate the problem of learning ABAC policies.

5.1.1 ABAC Policies

Attribute-based access control (ABAC) policies are specified as Boolean combinations of conditions on attributes of users and protected resources. The following definition is adapted from Xu *et al.* [136]:

Definition 5.1.1 (cf. ABAC Model [136]) *An ABAC model consists of the following components:*

- $\mathcal{U}$, $\mathcal{R}$, $\mathcal{O}$, $\mathcal{P}$ *refer, respectively, to finite sets of users, resources, operations, and rules.*

- A_U *refers to a finite set of user attributes. The value of an attribute* $a \in A_U$ *for a user* $u \in \mathcal{U}$ *is represented by a function* $d_U(u, a)$*. The range of* d_U *for an attribute* $a \in A_U$ *is denoted by* $V_U(a)$*.*

- A_R *refers to a finite set of resource attributes. The value of an attribute* $a \in A_R$ *for a resource* $r \in \mathcal{R}$ *is represented by a function* $d_R(r, a)$*. The range of* d_R *for an attribute* $a \in A_R$ *is denote*

- *A user attribute expression* e_U *defines a function that maps every attribute* $a \in A_U$*, to a value in its range or* $\perp$*,* $e_U(a) \in V_U(a) \cup \{\perp\}$*. Specifically,* e_U *can be expressed as the set of attribute/value pairs* $e_U = \{\langle a_i, v_i\rangle \mid a_i \in A_U \wedge f(a_i) = v_i \in V_U(a_i)\}$*. A user* u_i *satisfies* e_U *(i.e., it belongs to the set defined by* e_U*)*

iff for every user attribute a not mapped to $\perp$, $\langle a, d_U(u_i, a)\rangle \in e_U$. *Similarly, a resource* s_i *can be defined by a resource attribute expression* e_R.

- *A rule* $\rho \in \mathcal{P}$ *is a tuple* $\langle e_U, e_R, O, d\rangle$ *where* $\rho.e_U$ *is a user attribute expression,* e_R *is a resource attribute expression,* $O \subseteq \mathcal{O}$ *is a set of operations, and* d *is the decision of the rule* ($d \in \{permit, deny\}$)[1].

The original definition of an ABAC rule does not include the notion of "signed rules" (i.e., rules that specify positive or negative authorizations). By default, $d = permit$, and an access request $\langle u, r, o\rangle$ for which there exist at least a rule $\rho = \langle e_U, e_R, O, d\rangle$ in $\mathcal{P}$ such that u satisfies e_U (denoted by $u \models e_U$), r satisfies e_R (denoted by $r \models e_R$), and $o \in O$, is permitted. Otherwise, the access request is assumed to be denied. Even though negative authorizations are the default in access control lists, mixed rules are useful when dealing with large sets of resources organized according to hierarchies, and have been widely investigated [64]. They are used in commercial access control systems (e.g., the access control model of SQL Servers provides negative authorizations by means of the DENY authorization command), and they are part of the XACML standard.

5.1.2 Access Control Decision Examples

An access control decision example (referred to as a *decision example* (l)) is composed of an access request and its corresponding decision. Formally,

Definition 5.1.2 (Access Control Decisions and Examples (l)) *An access control decision is a tuple* $\langle u, r, o, d\rangle$ *where* u *is the user who initiated the access request,* r *is the resource target of the access request,* o *is the operation requested on the resource, and* d *is the decision taken for the access request. A decision example* l *is a tuple* $\langle u, e_U, r, e_R, o, d\rangle$ *where* e_U *and* e_R *are a user attribute expression, and a resource attribute expression such that* $u \models e_U$, *and* $r \models e_R$, *and the other arguments are interpreted as in an access control decision.*

[1]Throughout the paper, we will use the dot notation to refer to a component of an entity (e.g., $\rho.d$ refers to the decision of the rule ρ).

There exists an unknown set $\mathcal{F}$ of all access control decisions that should be allowed in the system, but for which we only have partial knowledge through examples. In an example l, the corresponding e_U and e_R can collect a few attributes (e.g., role, age, country of origin, manager, department, experience) depending on the available log information. Note that an access control decision is an example where e_U and e_R are the constant functions that assign to any attribute $\perp$. We say that an example $\langle u, e_U, r, e_R, o, d\rangle$ belongs to an access control decision set S iff $\langle u, r, o, d\rangle \in S$. Logs of past decisions are used to create an *access control decision example dataset* (see Definition 5.1.3). We say that a set of ABAC policies $\mathcal{P}$ *controls* an access control request $\langle u, r, o\rangle$ iff there is a rule $\rho = \langle e_U, e_R, O, d\rangle \in \mathcal{P}$ that satisfies the access request $\langle u, r, o\rangle$. Similarly, we say that the request $\langle u, r, o\rangle$ *is covered by* $\mathcal{F}$ iff $\langle u, r, o, d\rangle \in \mathcal{F}$, for some decision d. Therefore, $\mathcal{P}$ can be seen as defining the set of all access decisions for access requests controlled by $\mathcal{P}$ and, hence, be compared directly with $\mathcal{F}$ - and with some overloading in the notation, we want decisions in $\mathcal{P}$ ($\langle u, r, o, d\rangle \in \mathcal{P}$) to be decisions in $\mathcal{F}$ ($\langle u, r, o, d\rangle \in \mathcal{F}$).

Definition 5.1.3 (Access Control Decision Example Dataset ($\mathcal{D}$)) *An access control decision example dataset is a finite set of decision examples (i.e., $\mathcal{D} = \{l_1, \ldots, l_n\}$).*

$\mathcal{D}$ is expected to be mostly, but not necessarily, a subset of $\mathcal{F}$. $\mathcal{D}$ is considered to be *noisy* if $\mathcal{D} \not\subseteq \mathcal{F}$.

5.1.3 Problem Definition

Using a set of decision examples, extracted from a system history of access requests and their authorized/denied decisions, together with some context information (e.g., metadata obtained from LDAP directories), we want to learn ABAC policies (see Definition 5.1.4). The context information provides user and resource identifications, as well as user and resource attributes and their functions needed for an ABAC model. Additionally, it might also provide complementary semantic information about the system in the form of a collection of typed binary relations, $\mathcal{T} = \{t_1, \ldots, t_n\}$, such

that each relation t_i relates pairs of attribute values, i.e., $t_i \subseteq V_X(a_1) \times V_Y(a_2)$, with $X, Y \in \{\mathcal{U}, \mathcal{R}\}$, and $a_1 \in A_X$, and $a_2 \in A_Y$. For example, one of these relations can represent the organizational chart (department names are related in an *is_member* or *is_part* hierarchical relation) of the enterprise or institution where the ABAC access control system is deployed.

Definition 5.1.4 (Learning ABAC Policies by Examples and Context ($LAPEC$)) *Given an access control decision example dataset $\mathcal{D}$, and context information comprising the sets $\mathcal{U}$, $\mathcal{R}$, $\mathcal{O}$, the sets A_U and A_R, one of which could be empty, the functions assigning values to the attributes of the users and resources, and a possibly empty set of typed binary relations, $\mathcal{T} = \{t_1, \ldots, t_n\}$, $LAPEC$ aims at generating a set of ABAC rules (i.e., $\mathcal{P} = \{\rho_1 \ldots \rho_m\}$) that are able to control all access requests in $\mathcal{F}$.*

5.1.4 Policy Generation Assessment

ABAC policies generated by $LAPEC$ are assessed by evaluation of two quality requirements: ***correctness***, which refers to the capability of the policies to assign a correct decision to any access request (see Definition 5.1.5), and ***completeness***, which refers to ensuring that all actions, executed in the domain controlled by an access control system, are covered by the policies (see Definition 5.1.6). This assessment can be done against example datasets as an approximation of $\mathcal{F}$. Since datasets can be noisy, it is possible that two different decision examples for the same request are in the set. Validation will be done only against consistent datasets as we assume that the available set of access control examples is noise-free.

Definition 5.1.5 (Correctness) *A set of ABAC policies $\mathcal{P}$ is correct with respect to a consistent set of access control decisions $\mathcal{D}$ if and only if for every request $\langle u, r, o\rangle$ covered by $\mathcal{D}$, $\langle u, r, o, d\rangle \in \mathcal{P} \rightarrow \langle u, r, o, d\rangle \in \mathcal{D}$.*

Definition 5.1.6 (Completeness) *A set of ABAC policies $\mathcal{P}$ is complete with respect to a consistent set of access control decisions $\mathcal{D}$ if and only if, for every request $\langle u, r, o\rangle$, $\langle u, r, o\rangle$ covered by $\mathcal{D} \rightarrow \langle u, r, o\rangle$ is controlled by $\mathcal{P}$.*

These definitions allow $\mathcal{P}$ to control requests outside $\mathcal{D}$. The aim is twofold. First, when we learn $\mathcal{P}$ from an example dataset $\mathcal{D}$, we want $\mathcal{P}$ to be correct and complete with respect to $\mathcal{D}$. Second, beyond $\mathcal{D}$, we want to minimize incorrect decisions while maximizing completeness with respect to $\mathcal{F}$. Outside $\mathcal{D}$, we evaluate correctness statistically through cross validation with example datasets which are subsets of $\mathcal{F}$, and calculating Precision, Recall, F1 score, and accuracy of the decisions made by $\mathcal{P}$. We quantify completeness by the Percentage of Controlled Requests (PCR) which is the ratio between the number of decisions made by $\mathcal{P}$ among the requests covered by an example dataset and the total number of requests covered by the dataset.

Definition 5.1.7 (Percentage of Controlled Requests (PCR)) *Given a subset of access control decision examples $\mathcal{N} \subseteq \mathcal{F}$, and a policy set $\mathcal{P}$, the percentage of controlled requests is defined as follows:*

$$PCR = \frac{|\{\langle u, r, o\rangle \mid \langle u, r, o\rangle \text{ is covered by } \mathcal{N} \text{ and } \langle u, r, o\rangle \text{ is controlled by } \mathcal{P}\}|}{|\{\langle u, r, o\rangle \mid \langle u, r, o\rangle \text{ is covered by } \mathcal{N}\}|}$$

5.2 The Learning Framework

Our framework comprises four steps, see Fig. 5.1. Throughout this section, we use the following running example.

Example. *Consider a system including users and resources both associated with projects. User attributes, resource attributes, operations, and possible values for two selected attributes are shown in Table 5.1. Assume that a log of access control decision examples is given.*

5.2.1 Rules Mining

$\mathcal{D}$ provides a source of examples of user accesses to the available resources in an organization. In this step, we use association rule mining (ARM) [137] to analyze the association between users and resources.

Table 5.1.: Details about A Project Management System

User Attributes (A_U)	{id, role, department, project, technical area}
Resource Attributes (A_R)	{id, type, department, project, technical area}
Operations List ($\mathcal{O}$)	{request, read, write, setStatus, setSchedule, approve, setCost}
V_U(role)	{planner, contractor, auditor, accountant, department manager, project leader}
V_R(type)	{budget, schedule, and task}

Thus, rules having high association metrics (i.e., support and confidence scores) are kept to generate access control rules. ARM is used for capturing the frequency and data patterns and formalize them in rules. *Polisma* uses Apriori [138] (one of the ARM algorithms) to generate rules that are *correct and complete* with respect to $\mathcal{D}$. This step uses only the examples in $\mathcal{D}$ to mine a first set of rules, referred to as *ground rules*. These ground rules potentially are overfitted (e.g., ρ_1 in Fig. 5.2) or unsafely-generalized (e.g., ρ_2 in Fig. 5.2). Overfitted rules impact completeness over $\mathcal{F}$ while unsafely-generalized rules impact correctness. Therefore, *Polisma* post-processes the ground rules in the next step.

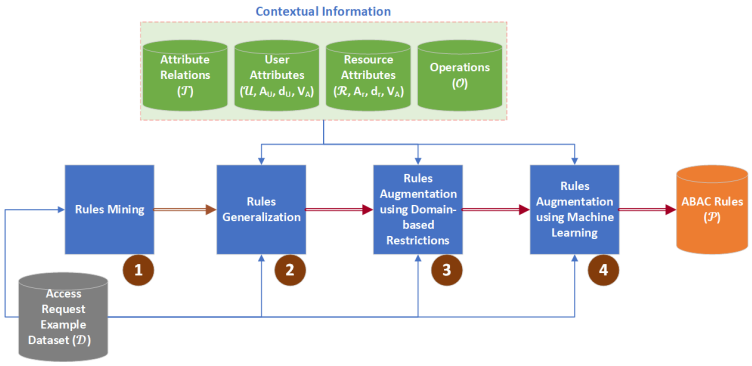


Fig. 5.1.: The Architecture of *Polisma*

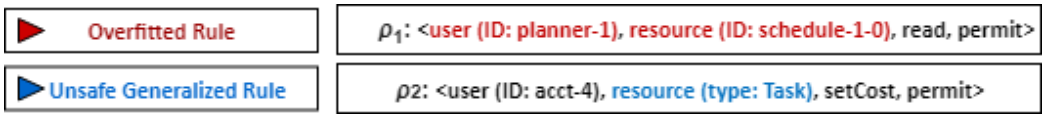


Fig. 5.2.: Examples of ground rules generated from rule mining based on the specifications of the running example

5.2.2 Rules Generalization

To address overfitting and unsafe generalization associated with the ground rules, this step utilizes the set of user and resource attributes A_U, A_R provided by either $\mathcal{D}$, external context information sources, or both. Straightforwardly, e_U (i.e., user expression) or e_R (i.e., resource expression) of a rule can be adapted to expand or reduce the scope of each expression when a rule is overfitted or is unsafely generalized, respectively. In particular, each rule is post-processed based on its corresponding user and resource by analyzing A_U and A_R to statistically "choose" the most appropriate attribute expression that captures the subsets of users and resources having the same permission authorized by the rule according to $\mathcal{D}$. In this way, this step searches for an attribute expression that minimally generalizes the rules.

Polisma provides two strategies for post-processing ground rules. One strategy, referred to as Brute-force Strategy BS, uses only the attributes associated with users and resources appearing in $\mathcal{D}$. The other strategy assumes that attribute relationship metadata ($\mathcal{T}$) is also available. Hence, the second strategy, referred to as Structure-based Strategy SS, exploits both attributes and their relationships. In what follows, we describe each strategy.

Brute-force Strategy (BS)

To post-process a ground rule, this strategy searches heuristically for attribute expressions that cover the user and the resource of interest. Each attribute expression

is weighted statistically to assess its "safety" level for capturing the authorization defined by the ground rule of interest.

The safety level of a user attribute expression e_U for a ground rule ρ is estimated by the proportion of the sizes of two subsets: a) the subset of decision examples in $\mathcal{D}$ such that the access requests issued by users satisfy e_U while the remaining parts of the decision example (i.e., resource, operation, and decision) match the corresponding ones in ρ; and b) the subset of users who satisfy e_U. The safety level of a resource attribute expression is estimated similarly. Thereafter, the attribute expression with the highest safety level is selected to be used for post-processing the ground rule of interest. Formally:

Definition 5.2.1 (User Attribute Expression Weight $W_{uav}(u_i, a_j, d_U(u_i, a_j), \mathcal{D})$) *For a ground rule ρ and its corresponding user $u_i \in U$, the weight of a user attribute expression $\langle a_j, d_U(u_i, a_j)\rangle$ is defined by the formula: $W_{uav} = \frac{|U_D|}{|U_C|}$, where $U_D = \{l \in \mathcal{D} \mid d_U(u_i, a_j) = d_U(l.u, a_j) \wedge l.r \in \rho.e_R \wedge l.o \in \rho.O \wedge \rho.d = l.d\}$ and $U_C = \{u_k \in \mathcal{U} \mid d_U(u_i, a_j) = d_U(u_k, a_j)\}$*

Different strategies for selecting attribute expressions are possible. They can be based on a single attribute or a combination of attributes, and they can consider either user attributes, resource attributes, or both. Strategies using only user attributes are referred to as *BS-U-S* "for a single attribute" and *BS-U-C* "for a combination of attributes". Similarly, *BS-R-S* and *BS-R-C* are strategies for a single or a combination of resources attributes. The strategies using the attributes of both users and resources are referred to as *BS-UR-S* and *BS-UR-C*. Due to space limitation, we show only the algorithm using the selection strategy *BS-U-S* (see Algorithm 5.1).

Example. Assume that BS-UR-C is used to generalize ρ_2 defined in Fig. 5.2. A_U is {id, role, department, project, technical area} and A_R is {id, type, department, project, technical area}. Moreover, ρ_2 is able to control the access for the user whose ID is "acct-4' when accessing a resource whose type is task. The attribute values of the user and resources controlled by ρ_2 are analyzed. To generalize ρ_2 using BS-UR-C, each attribute value is weighted as shown in Fig. 5.3. For weighting each

Algorithm 5.1 Brute-force Strategy (*BS-U-S*) for Generalizing Rules

Require: ρ_i: a ground rule, u_i: the user corresponding to ρ_i, $\mathcal{D}$, A_U.

1: Define w_{max} = -∞, $a_{selected}$= $\perp$.
2: **for** $a_i \in A_U$ **do**
3: $w_i = W_{uav}(u_i, a_i, d_U(u_i, a_i), \mathcal{D})$
4: **if** $w_i > w_{max}$ **then**
5: $a_{selected} = a_i$, $w_{max} = w_{a_i}$
6: **end if**
7: **end for**
8: $e_U \leftarrow \langle\ a_{selected}, d_U(u_i, a_{selected})\ \rangle$
9: Create Rule $\rho_i' = \langle e_U, \rho_i.e_R, \rho_i.O, \rho_i.d \rangle$
10: **return** ρ_i'

Algorithm 5.2 Structure-based Strategy (*SS*) for Generalizing Rules

Require: ρ_i: a ground rule to be Generalized, u_i: the user corresponding to ρ_i, r_i: the resource corresponding to ρ_i, $\mathcal{T}$, A_U, A_R.

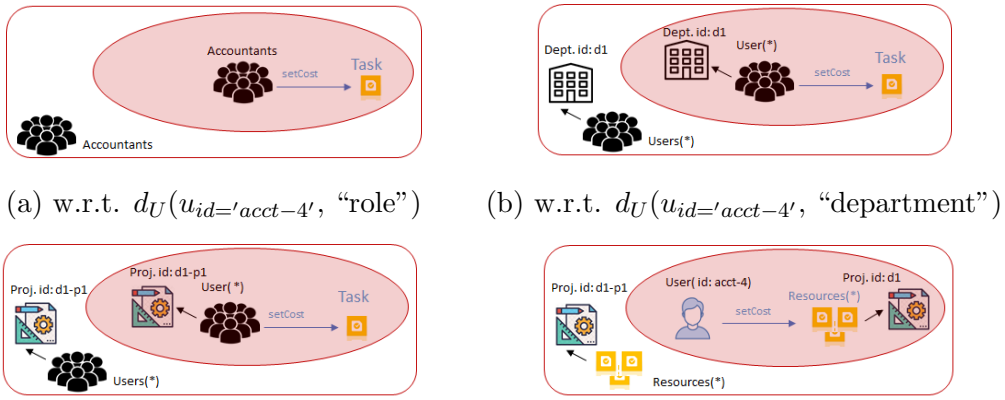
1: $g_i = G(u_i, r_i, A_U, A_R, \mathcal{T})$
2: $\langle x \in A_U, y \in A_R \rangle$ = First-Common-Attribute-Value(g_i)
3: $e_U \leftarrow \langle\ x, d_U(u_i, x)\ \rangle$; $e_R \leftarrow \langle\ y, d_R(r_i, y)\ \rangle$
4: Create Rule $\rho_i' = \langle e_U, e_R, \rho_i.O, \rho_i.d \rangle$
5: **return** ρ_i'

user/resource attribute value, the proportion of the sizes of two user/resource subsets is calculated according to Definition 5.2.1.

In particular, for the value of the "department" attribute corresponding to the user referred by ρ_2 (i.e., "d1") (Fig. 5.3b), two user subsets are first found: a) the subset of the users belonging to department "d1"; and b) the subset of the users belonging to department "d1" and having a permission to perform the "setCost" operation on a resource of type "task" based on $\mathcal{D}$. Then, the ratio of the sizes of these subsets is

considered as the weight for the attribute value "d1". The weights for the other user and resource attributes values are calculated similarly. Thereafter, the user attribute value and resource attribute value with the highest wights are chosen to perform the generalization. Assume that the value of the "department" user attribute is associated with the highest weight and the " project" resource attribute is associated with the highest weight. ρ_2 *is generalized as:*

ρ_2'=⟨*user(department: d1), resource (type: task, project: d1-p1), setCost, permit*⟩



(a) w.r.t. $d_U(u_{id='acct-4'}$, "role") (b) w.r.t. $d_U(u_{id='acct-4'}$, "department")

(c) w.r.t. $d_U(u_{id='acc-4'}$, "project") (d) w.r.t. $d_R(r_{type='task'}$, "project")

Fig. 5.3.: Generalization of ρ_2 defined in Fig. 5.2 using the Brute Force Strategy (*BS-UR-C*)

Structure-based Strategy (*SS*)

The *BS* strategy works without prior knowledge of $\mathcal{T}$. When $\mathcal{T}$ is available, it can be analyzed to gather information about "hidden" correlations between common attributes of a user and a resource. Such attributes can potentially enhance rule generalization. For example, users working in a department t_i can potentially access the resources owned by t_i. The binary relations in $\mathcal{T}$ are combined to form a directed graph, referred to as *Attribute-relationship Graph* G (see Definition 5.2.2). Traversing this graph starting from the lowest level in the hierarchy to the highest one allows

one to find the common attributes values between users and resources. Among the common attribute values, the one with the least hierarchical level, referred to as *first common attribute value*, has heuristically the highest safety level for generalization because the generalization using such attribute value supports the least level of generalization. Subsequently, to post-process a ground rule, this strategy uses $\mathcal{T}$ along with A_U, A_R of both the user and resource of interest to build G. Thereafter, G is used to find the *first common attribute value* between the user and resource of that ground rule to be used for post-processing the ground rule of interest (as described in Algorithm 5.2).

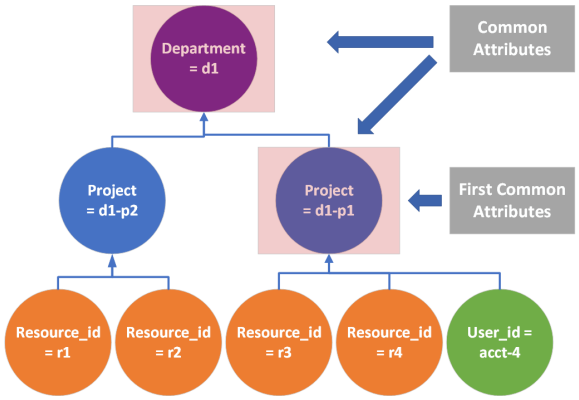


Fig. 5.4.: Generalization of ρ_2 defined in Fig. 5.2 using Structure-based Strategy: An example of Attribute-relationship Graph

Example. Assume that SS is used to generalize ρ_2, defined in Fig. 5.2. Also, suppose that Polisma is given the following information:

- *The subset of resources R' satisfying $\rho_2.e_R$ has two values for the project attribute (i.e., "d1-p1", "d1-p2").*
- *The user "acct-4" belongs to the project "d1-p1".*
- *R' and "acct-4" belong to the department "d1".*
- *$\mathcal{T}$ = {("d1-p1", "d1"), ("d1-p2", "d1")}.*

G is constructed as shown in Fig. 5.4. Using G, the two common attributes for "acct-4" and R′ are "d1-p1" and "d1" and the first common attribute is "d1-p1". Therefore, ρ_2 is generalized as follows:

$\rho_2'' = \langle$*user(project: d1-p1), resource (type: task, project: d1-p1), setCost, permit*$\rangle$

Definition 5.2.2 (Attribute-relationship Graph G) *Given A_U,A_R, ρ and $\mathcal{T}$, the attribute-relationship graph (G) of ρ is a directed graph composed of vertices V and edges E where*

$V = \{v \mid \forall u_i \in \rho.e_U, \forall a_i \in A_U, v = d_U(u_i, a_i)\}$

$\cup \{v \mid \forall r_i \in \rho.e_R, \forall a_i \in A_R, v = d_R(r_i, a_i)\}$, *and*

$E = \{(v_1, v_2) \mid \exists t_i \in \mathcal{T} \wedge (v_1, v_2) \in t_i \wedge v_1, v_2 \in V\}$.

Proposition 5.2.1 *Algorithms 1 and 2 output generalized rules that are correct and complete with respect to $\mathcal{D}$.*

As discussed earlier, the first step generates ground rules that are either overfitted or unsafely-generalized. This second step post-processes unsafely-generalized rules into safely-generalized ones; hence, improving correctness. It may also post-process overfitted rules into safely-generalized ones; hence, improving completeness. However, completeness can be further improved as described in the next subsections.

5.2.3 Rules Augmentation using Domain-based Restrictions

"Safe" generalization of ground rules is one approach towards fulfilling completeness. Another approach is to analyze the authorization domains of users and resources appearing in $\mathcal{D}$ to augment restriction rules, referred to as *domain-based restriction rules*[2]. Such restrictions potentially increase safety by avoiding erroneous accesses. Basically, the goal of these restriction rules is to augment negative authorization.

One straightforward approach for creating domain-based restriction rules is to analyze the authorization domain for each individual user and resource. However, such

[2]This approach also implicitly improves correctness.

an approach leads to the creation of restrictions suffering from overfitting. Alternatively, the analysis can be based on groups of users or resources. Identifying such groups requires pre-processing A_U and A_R. Therefore, this step searches heuristically for preferable attributes to use for partitioning the user and resource sets. The heuristic prediction for preferable attributes is based on selecting an attribute that partitions the corresponding set evenly. Hence, estimating the attribute distribution of users or resources allows one to measure the ability of the attribute of interest for creating even partitions[3]. One method for capturing the attribute distribution is to compute the attribute entropy as defined in Eq. 5.1. The attribute entropy is maximized when the users are distributed evenly among the user partitions using the attribute of interest (i.e., sizes of the subsets in $G_U^{a_i}$ are almost equal). Thereafter, the attribute with the highest entropy is the preferred one.

Example. Assume that we decide to analyze the authorization domain by grouping users based on the "role" user attribute. As shown in the top part of Fig. 5.5, the authorization domains of the user groups having distinct values for the "role" attribute are identified using the access requests examples of $\mathcal{D}$. These authorization domains allow one to recognize the set of operations authorized per user group. Thereafter, a set of negative authorizations are generated to restrict users having a specific role from performing specific operations on resources. Given the preferred attributes, this step constructs user groups as described in Definition[4] 5.2.3. These groups can be analyzed with respect to $\mathcal{O}$ as described in Algorithm 5.3 Consequently, user groups $G_U^{a_i}$ and resource groups $G_R^{a_i}$ along with $\mathcal{O}$ comprise two types of authorization domains: operations performed by each user group, and users' accesses to each resource group. Subsequently, the algorithm augments restrictions based on these access domains as follows.

[3]Even distribution tends to generate smaller groups. Each group potentially has a similar set of permissions. A large group of an uneven partition potentially includes a diverse set of users or resource; hence hindering observing restricted permissions

[4]The definition of constructing resource groups is analogous to that of user groups.

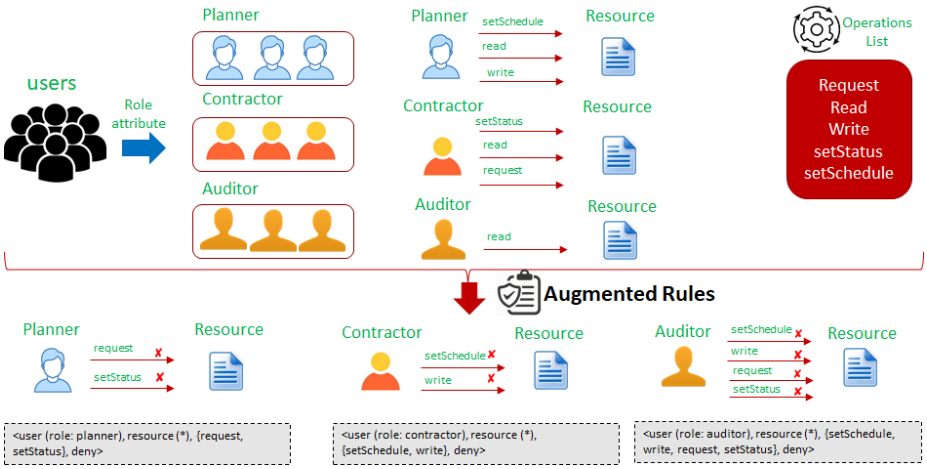


Fig. 5.5.: Rules Augmentation Using Domain-based Restrictions

- It generates deny restrictions based on user groups (similar to the example in Fig. 5.5). These restriction rules deny any access request from a specific user group using a specific operation to access any resource.
- It generates deny restrictions based on both groups of users and resources. These restriction rules deny any access request from a specific user group using a specific operation to access a specific resource group.

On the other hand, another strategy assumes prior knowledge about preferred attributes to use for partitioning the user and resource sets (referred to as Semantic Strategy (SS)) .

Definition 5.2.3 (Attribute-based User Groups $G_U^{a_i}$) *Given $\mathcal{U}$ and $a_i \in A_U$, $\mathcal{U}$ is divided into is a set of user groups $G_U^{a_i}$ $\{g_1^{a_i}, \ldots, g_k^{a_i}\}$ where $(g_i^{a_i} = \{u_1, \ldots, u_m\} \mid \forall u', u'' \in g_i, d_U(u', a_i) = d_U(u'', a_i)$ $, m \leq \mid \mathcal{U} \mid) \wedge (g_i \cap g_j = \phi \mid i \neq j) \wedge (k = \mid V_U(a_i) \mid)$.*

$$Entropy(G_U^{a_i}, a_i) = (\frac{-1}{\ln m} * \sum_{j=1, g_j \in G_U^{a_i}}^{m} p_j * \ln p_j), where\ m = |G_U^{a_i}|,\ p_j = \frac{|g_j|}{\sum_{l=1, g_l \in G_U^{a_i}}^{m} |g_l|} \tag{5.1}$$

Proposition 5.2.2 *Step 3 outputs generalized rules that are correct and complete with respect to* $\mathcal{D}$.

Algorithm 5.3 Rules Augmentation using Domain-based Restrictions

---

Require: $\mathcal{D}$, $\mathcal{U}$, $\mathcal{R}$, x: a preferable attribute of users, y: a preferable attribute of resources, $\mathcal{O}$.

1: $G_U = \mathcal{G}_U^{a_i}(\mathcal{U}, x)$; $G_R = \mathcal{G}_R^{a_i}(\mathcal{R}, y)$

2: $\forall\ g_i \in G_U$, $O^u_{g_i} \rightarrow \{\}$; $\forall\ g_i \in G_R$, $O^r_{g_i} \rightarrow \{\}$; $\forall\ g_i \in G_R$, $U^r_{g_i} \rightarrow \{\}$

3: **for** $l_i \in \mathcal{D}$ **do**

4: $\quad g' \leftarrow g_i \in G_U \mid l_i.u \in g_i \wedge l_i.d = Permit$; $O^u_{g'} \rightarrow O^u_{g'} \cup l_i.o$

5: $\quad O^r_{g''} \rightarrow O^r_{g''} \cup l_i.o$; $U^r_{g''} \rightarrow U^r_{g''} \cup l_i.u$

6: **end for**

7: $\mathcal{P}' \rightarrow \{\}$

8: **for** $g_i \in G_U$ **do**

9: $\quad \rho_i = \langle\langle x, d_U(u_i, x)\rangle, *, o_i, Deny\rangle \mid u_i \in g_i \wedge o_i \in (\mathcal{O} \setminus O^u_{g_i})$; $\mathcal{P}' \rightarrow \mathcal{P}' \cup \rho_i$

10: **end for**

11: **for** $g_i \in G_R$ **do**

12: $\quad \rho_i = \langle\langle x, d_U(u_i, x)\rangle, \langle y, d_R(r_i, y)\rangle, *, Deny\rangle \mid r_i \in g_i \wedge u_i \in (\mathcal{U} \setminus U^r_{g_i})$; $\mathcal{P}' \rightarrow \mathcal{P}' \cup \rho_i$

13: **end for**

14: **return** $\mathcal{P}'$

5.2.4 Rules Augmentation using Machine Learning

The rules generated from the previous steps are generalized using domain knowledge and data statistics extracted from $\mathcal{D}$. However, these steps do not consider generalization based on the similarity of access requests. Thus, these rules cannot control a new request that is similar to an old one (i.e., a new request has a similar

pattern to one of the old requests, but the attribute values of the new request do not match the old ones).

Example. Assume that $\mathcal{D}$ includes a decision example (l_i) for a user (id: "pl-1") accessing a resource (id: "sc-1") where both of them belong to a department "d1". Assuming Polisma generated a rule based on l_i as follows: $\rho_i = \langle$ user(role: planner, department: d1), resource (type: schedule, department: d1), read, permit $\rangle$ Such a rule cannot control a new request by a user (id: "pl-5" for accessing a resource (id: "sc-5") where both of them belong to another department "d5"). Such a is similar to l_i. Therefore, a prediction-based approach is required to enable generating another rule.

A possible prediction approach is to use an ML classifier that builds a model based on the attributes provided by $\mathcal{D}$ and context information. The generated model implicitly creates patterns of accesses and their decisions, and will be able to predict the decision for any access request based on its similarity to the ones controlled by the rules generated by the previous steps. Thus, this step creates new rules based on these predictions. Once these new rules are created, *Polisma* repeats Step 2 to safely generalize the ML-based augmented rules. Notice that ML classification algorithms might introduce some inaccuracy when performing the prediction. Hence, we utilize this step only for access requests that are not covered by the rules generated by the previous three steps. Thus, the inaccuracy effect associated with the ML classifier is minimized but the *correctness and completeness* are preserved.

Note that Polisma is used as a one-time learning. However, if changes arise in terms of regulations, security policies and/or organizational structure of the organization, the learning can be executed again (i.e., on-demand learning).

5.3 Evaluation

In this section, we summarize the experimental methodology and report the evaluation results of *Polisma*.

Table 5.2.: Overview of Datasets

	Datasets	
	Project Management *(PM)*	***Amazon*** *(AZ)*
# of decision examples	550	1000
# of users	150	480
# of resources	292	271
# of operations	7	1
# of examples with a "Permit" decision	505	981
# of examples with a "Deny" decision	50	19
# of User Attributes	5	12
# of Resource Attributes	6	1

5.3.1 Experimental Methodology

Datasets. To evaluate *Polisma*, we conducted several experiments using two datasets: one is a synthetic dataset (referred to as *project management* (PM) [136]), and the other is a real one (referred to as *Amazon*[5]). The PM dataset has been generated by the Reliable Systems Laboratory at Stony Brook University based on a probability distribution in which the ratio is 25 for rules, 25 for resources, 3 for users, and 3 for operations. In addition to decision examples, PM is tagged with context information (e.g., attribute relationships and attribute semantics). We used such a synthetic dataset (i.e., PM) to show the impact of the availability of such semantic information on the learning process and the quality of the generated rules. Regarding the Amazon dataset, it is a historical dataset collected in 2010 and 2011. This dataset is an anonymized sample of access provisioned within the Amazon company. One characteristic of the Amazon dataset is that it is sparse (i.e., less than 10% of the attributes are used for each sample). Furthermore, since the Amazon dataset is large (around 700K decision examples), we selected a subset of the Amazon dataset (referred to as AZ). The subset was selected randomly based on a statistical analysis [139] of the size of the Amazon dataset where the size of the selected samples suffices a confidence score of 99% and a margin error score of 4%. Table 5.2 shows

[5] `http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples`

statistics about the PM and AZ datasets.

Comparative Evaluation. We compared *Polisma* with three approaches. We developed a *Naïve ML approach* which utilizes an ML classifier to generate rules. A classification model is trained using $\mathcal{D}$. Thereafter, the trained model is used to generate rules without using the generalization strategies which are used in *Polisma*. The rules generated from this approach are evaluated using another set of new decision examples (referred to as $\mathcal{N}$)[6]. Moreover, we compared *Polisma* with two recently published state-of-the-art approaches for learning ABAC policies from logs: a) Xu & Stoller Miner [136], and b) Rhapsody by Cotrini *et al.* [140].

Evaluation and Settings. On each dataset, *Polisma* is evaluated by splitting the dataset into training $\mathcal{D}$ (70%) and testing $\mathcal{N}$ (30%) subsets. We performed a 10-fold cross-validation on $\mathcal{D}$ for each dataset. As the ML classifiers used for the fourth step of *Polisma* and the naïve approach, we used a combined classification technique based on majority voting where the underlying classifiers are Random Forest and kNN[7]. All experiments were performed on a 3.6 GHz Intel Core i7 machine with 12 GB memory running on 64-bit Windows 7.

Evaluation Metrics. The correctness of the generated ABAC rules is evaluated using the standard metrics for the classification task in the field of machine learning. Basically, predicted decisions for a set of access requests are compared with their ground-truth decisions. Our problem is analogous to a two-class classification task because the decision in an access control system is either "permit" or "deny". Therefore, for each type of the decisions, a confusion matrix is prepared to enable calculating the values of accuracy, precision, recall, and F1 score as outlined in Eqs 5.2-5.3 [8].

[6]Datasets are assumed to be noise-free, that is, $(\mathcal{N} \subset \mathcal{F}) \wedge (\mathcal{D} \subset \mathcal{F}) \wedge (\mathcal{N} \cap \mathcal{D} = \phi)$. Note that $\mathcal{F}$ is the complete set of control decisions which we will never have in a real system.

[7]Other algorithms can be used. We used Random Forest and kNN classifiers since they showed better results compared to SVM and Adaboost.

[8]F1 Score is the harmonic mean of precision and recall.

$$Precision = \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}, Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

Regarding the completeness of the rules, they are measured using *PCR* (see Definition 5.1.7).

5.3.2 Experimental Results

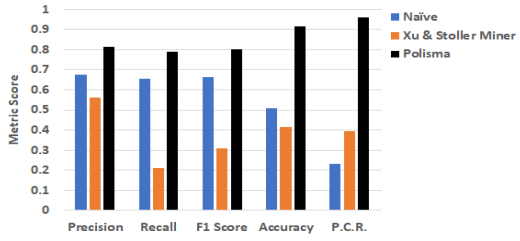


Fig. 5.6.: Comparison of Naïve, Xu & Stoller Miner, and *Polisma* Using the *PM* Dataset

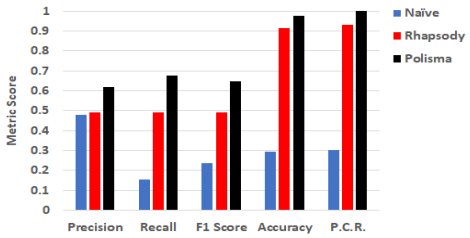
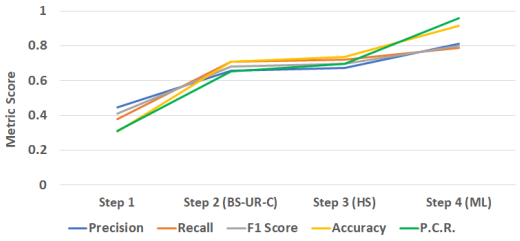
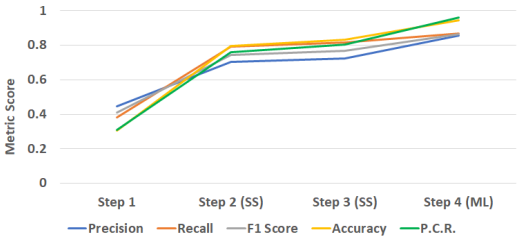


Fig. 5.7.: Comparison of Naïve, Rhapsody, and *Polisma* Using the *AZ* Dataset



(a) *Polisma* (BS-HS-ML)



(b) *Polisma* (SS-SS-ML)

Fig. 5.8.: Evaluation of *Polisma* using the *PM* dataset.

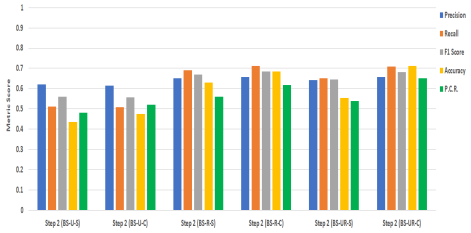


Fig. 5.9.: Comparison between the variants of the brute-force strategy (Step 2) using the *PM* dataset.

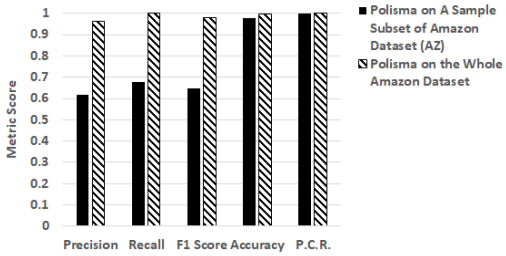


Fig. 5.10.: *Polisma* Evaluation on the Amazon Dataset (a sample subset and the whole set).

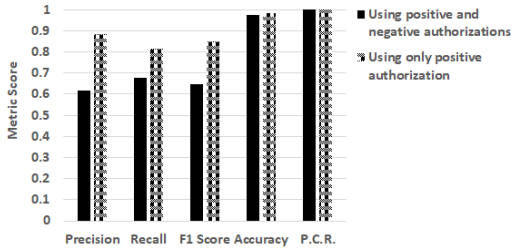


Fig. 5.11.: *Polisma* Evaluation on a sample subset of Amazon Dataset for only positive authorizations.

Polisma vs. Other Learning Approaches

Here, *Polisma* uses the default strategies in each step (i.e., the *BS-UR-C* strategy in the second step and the *HS* strategy in the third step[9]) and the impact of the other strategies in *Polisma* is evaluated next. The results in Fig. 5.6 show that *Polisma* achieves better results compared to the naïve and Xu & Stoller Miner using the *PM* dataset. In this comparison, we did not include Rhapsody because the default parameters of Rhapsody leads to generating no rule. With respect to Xu & Stoller Miner, *Polisma*'s F1 score (*PCR*) improves by a factor of 2.6 (2.4). This shows the importance of integrating multiple techniques (i.e., data mining, statistical,

[9]Since the *AZ* dataset does not contain resource attributes, *BS-R-C* (instead of *BS-UR-C*) is executed in the second step and the execution of the third step is skipped.

and ML techniques) in *Polisma* to enhance the policy learning outcome. Moreover, Xu & Stoller Miner requires prior knowledge about the structure of context information while *Polisma* benefits from such information when available. Xu & Stoller Miner [136] generates rules only for positive authorization (i.e., no negative authorization). This might increase the possibility of vulnerabilities and encountering incorrect denials or authorizations. Moreover, Xu & Stoller in their paper [136] used different metrics from the classical metrics for data mining that are used in our paper. In their work [136], they used a similarity measure between the generated policy and the real policy which given based on some distance function. However, such a metric does not necessarily reflect the correctness of the generated policy (i.e., two policies can be very similar, but their decisions are different). In summary, this experiment shows that the level of correctness (as indicated by the precision, recall and F1 score) and completeness (as indicated by the *PCR*) of the policies that are generated by *Polisma* is higher than for the policies generated by Xu & Stoller Miner and the naïve approach due to the reasons explained earlier (i.e., the lack of negative authorization rules), as well as generating generalized rules without considering the safety of generalization. Furthermore, as shown in Fig. 5.7, *Polisma* also outperforms the naïve approach and Rhapsody using the *AZ* dataset. Rhapsody [140] only considers positive authorization (similar to the problem discussed above about Xu & Stoller Miner [136]); hence increasing the chances of vulnerabilities. In this comparison, we have excluded Xu & Stoller Miner because of the shortage of available resource attributes information in the *AZ* dataset. Concerning the comparison with the naïve approach, on the *PM* dataset, the F1 score (*PCR*) of *Polisma* improves by a factor of 1.2 (4.1) compared to that of the naïve. On the *AZ* dataset, *Polisma*'s F1 score (*PCR*) improves by a factor of 2.7 (3.3) compared to the naïve. Both *Polisma* and the naïve approach use ML classifiers. However, *Polisma* uses an ML classifier for generating only some of the rules. This implies that among *Polisma* steps which improve the results of the generated rules (i.e., Steps 2 and Step 4), the rule generalization step is essential for enhancing the final outcome. In summary, the reported results show that the

correctness level of the rules generated by *Polisma* is better than that of the ones generated by Rhapsody and the naïve approach (as indicated by the difference between the precision, recall, and F1 scores metrics). Meanwhile, the difference between the completeness level (indicated by PCR) of the generated rules using *Polisma* and that of Rhapsody is not large. The difference in terms of completeness and correctness is a result of Rhapsody missing the negative authorization rules.

Steps Evaluation

Fig. 5.8 shows the evaluation results for the four steps of *Polisma* in terms of precision, recall, F1 score, accuracy, and *PCR*. In general, the quality of the rules improves gradually for the two datasets, even when using different strategies in each step. The two plots show that the strategies that exploit additional knowledge about user and resource attributes (e.g., $\mathcal{T}$) produce better results when compared to the ones that require less prior knowledge. Moreover, the quality of the generated rules is significantly enhanced after the second and the fourth steps. This shows the advantage of using safe generalization and similarity-based techniques. On the other hand, the third step shows only a marginal enhancement on the quality of rules. However, the rules generated from this step can be more beneficial when a large portion of the access requests in the logs are not allowed by the access control system.

We also conducted an evaluation on the whole Amazon dataset; the results are shown in Fig. 5.10. On the whole dataset, *Polisma* achieves significantly improved scores compared to those when using the *AZ* dataset because *Polisma* was able to utilize a larger set of decision examples. Nonetheless, given that the size of *AZ* was small compared to that of the whole Amazon dataset, *Polisma* outcomes using *AZ*[10] are promising. In summary, the increase of recall can be interpreted as reducing denials of authorized users (i.e., smaller number of false-negatives). A better precision value is interpreted as being more accurate with the decisions

[10]We also experimented samples of different sizes (i.e., 2k-5k), the learning results using these sample sizes showed slight improvement of scores.

(i.e., smaller number of false-positives). Figs. 5.8a-5.8b show that most of the reduction of incorrect denials is achieved in Steps 1 and 2, whereas the other steps improve precision which implies more correct decisions (i.e., decreasing the number of false-positives and increasing the number of true-positives). As Fig. 5.11 shows, the results improve significantly when considering only positive authorizations. This is due to the fact that the negative examples are few and so they are not sufficient in improving the learning of negative authorizations. As shown in the figure, precision, recall, and F1 score increase indicating that the learned policies are correct. Precision is considered the most effective metric especially for only positive authorizations to indicate the correctness of the generated policies since higher precision implies less incorrect authorizations.

Variants of the Brute-force Strategy

As the results in Fig. 5.9 show, using resource attributes for rules generalization is better than using the user attributes. The reason is that the domains of the resource attributes (i.e., $V_R(a_i) \mid a_i \in A_R$) is larger than that of the user attributes (i.e., $V_U(a_j) \mid a_j \in A_U$) in the PM dataset. Thus, the selection space of attribute expressions is expanded; hence, it potentially increases the possibility of finding better attribute expression for safe generalization. In particular, using resources attributes achieves 23% and 19% improvement F1 score and PCR when compared to that of using users attributes producing a better quality of the generated policies in terms of correctness (as indicated by the values of precision, recall, and F1 score) and completeness (as indicated by the PCR value). Similarly, performing the generalization using both user and resource attributes is better than that of using only user attribute or resource attributes because of the larger domains which can be exploited to find the best attribute expression for safe generalization. In particular, $BS\text{-}UR\text{-}C$ shows a significant improvement compared to $BS\text{-}U\text{-}C$ (22% for F1 score (which reflects a

better quality of policies in terms of correctness), and 25% for *PCR* (which reflects a better quality of policies in terms of completeness)). In conclusion, the best variant of the *BS* strategy is the one that enables exploring the largest set of possible attribute values to choose the attribute expression which has the highest safety level.

5.4 Related Work to the Polisma Framework

Policy mining has been widely investigated. However, the focus has been on RBAC role mining that aims at finding roles from existing permission data [141] [142]. More recent work has focused on extending such mining-based approaches to ABAC [136] [140] [143] [144] [145]. Xu and Stoller [136] proposed the first approach for mining ABAC policies from logs. Their approach iterates over a log of decision examples greedily and constructs candidate rules; it then generalizes these rules by utilizing merging and refinement techniques. Medvet *et al.* [143] proposed an evolutionary approach which incrementally learns a single rule per group of decision examples utilizing a divide-and-conquer algorithm. Cotrini *et al.* [140] proposed another rule mining approach, called Rhapsody, based on APRIORI-SD (a machine-learning algorithm for subgroup discovery) [146]. It is incomplete, only mining rules covering a significant number of decision examples. The mined rules are then filtered to remove any refinements. All of those approaches ([136] [140] [143]) mainly rely on decision logs assuming that they are sufficient for rule generalization. However, logs, typically being very sparse, might not contain sufficient information for mining rules of good quality. Thus, those approaches may not be always applicable. Moreover, neither of those approaches is able to generate negative authorization rules.

Mocanu *et al.* [144] proposed a deep learning model trained on logs to learn a Restricted Boltzmann Machine (RBM). Then, the learned model is used to generate candidate rules. Their proposed system is still under development and further testing is needed. Karimi and Joshi [145] proposed to apply clustering algorithms over the decision examples to predict rules, but they don't support rules general-

ization. To the best of our knowledge, *Polisma* is the first generic framework that incorporates context information, when available, along with decisions logs to increase accuracy. Another major difference of *Polisma* with respect to existing approaches is that *Polisma* can use ML techniques, such as statistical ML techniques, for policy mining while at the same time being able to generate ABAC rules expressed in propositional logics.

6. FLAP - A FEDERATED LEARNING FRAMEWORK FOR ATTRIBUTE-BASED ACCESS CONTROL POLICIES

6.1 Background and Problem Description

In this section, we first introduce background notions about the attribute-based access control (ABAC) model. We then briefly describe our approach to learn ABC policies. Finally we formally introduce the problem addressed in this paper.

6.1.1 ABAC Policies

We assume a formal attribute-based access control (ABAC) model [136] that includes several finite sets: users $\mathcal{U}$, resources $\mathcal{R}$, operations $\mathcal{O}$, and rules $\mathcal{P}$. Each user (i.e., $u \in \mathcal{U}$) and resource (i.e., $r \in \mathcal{R}$) is represented by independent sets of attributes (referred to as user attributes A_U and resource attributes A_R, respectively). The values of these attributes are represented by a function which assigns to each attribute a value from the value range for the attribute as shown in Eqs. 6.1 and 6.2. An ABAC policy consists of a set of rules $\mathcal{P}$ where each rule ρ is defined as $\langle e_U, e_R, O, d\rangle$ where e_U is a user attribute expression, e_R is a resource attribute expression, $O \subseteq \mathcal{O}$ is a set of operations, and d is the decision of the rule ($d \in \{permit, deny\}$). An attribute expression comprises a set of attribute/value pairs. For example, a user u_i satisfies e_U (denoted by $u_i \models e_U$) iff for every user attribute a not mapped to $\perp$, $\langle a,\ d_U(u_i, a)\rangle \in e_U$, and a resource r_i satisfies e_R (i.e., it belongs to the set defined by e_R, denoted by $r_i \models e_R$) iff for every resource attribute a not mapped to $\perp$, $\langle a,\ d_R(r_i, a)\rangle \in e_R$.

$$d_U(u_i, a_j) = v_k \mid u_i \in \mathcal{U} \wedge a_j \in \mathcal{A}_\mathcal{U} \wedge v_k \in \mathcal{V}_\mathcal{U}(a_j) \tag{6.1}$$

$$d_R(r_i, a_j) = v_k \mid r_i \in \mathcal{R} \wedge a_j \in \mathcal{A}_\mathcal{R} \wedge v_k \in \mathcal{V}_\mathcal{R}(a_j) \tag{6.2}$$

Intuitively, policy rules can be compared or combined using algebra operators (referred to policy algebra operators). These operators are a policy algebra could be represented as rule set operations. Compound policies can be obtained by combining policy rule sets through the algebra operators. We formally define the following algebra operators over rule sets:

6.1.2 Policy Learning

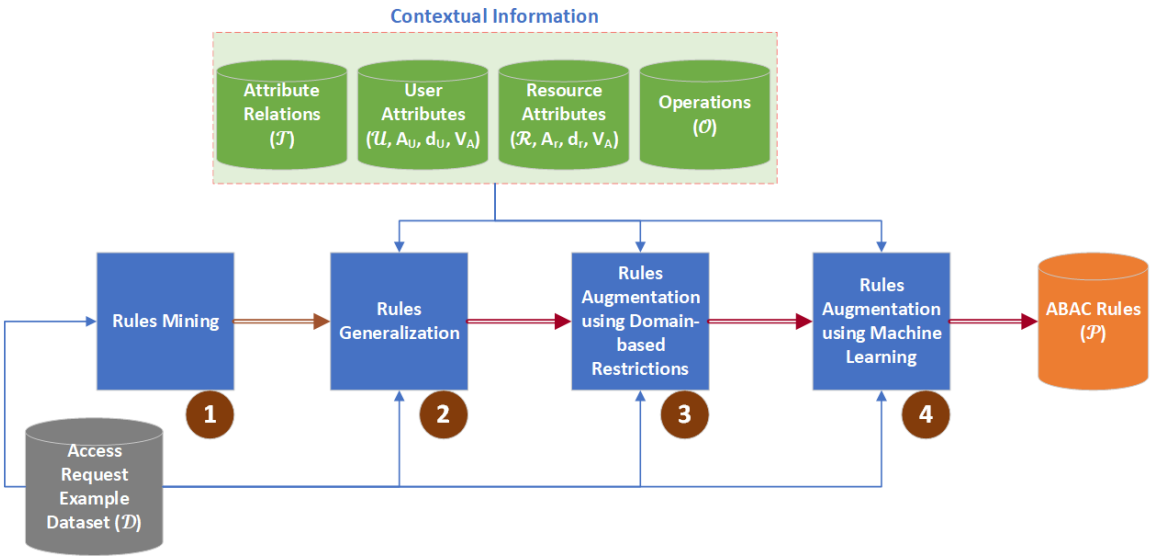


Fig. 6.1.: The Architecture of Polisma [49]

With the availability of a log of access requests and their corresponding control decisions, one can analyze such a log and generate the specifications of access control policies to control future requests. This problem is referred to as policy learning

(a.k.a. policy mining) and several approaches have been proposed to solve this problem [136] [140] [147]. Recently, we introduced a novel framework, named Polisma [49], which utilizes both examples of access requests and corresponding access control decisions as well as contextual information obtained from other data sources (e.g., organizational charts, user directories (e.g., LDAP directories), workflows, security tracking's logs (e.g., SIEM)), if available.

In Polisma (see Fig. 6.1), learning process is performed according to a sequence of steps. A data mining technique is first used to infer associations between parties and resources in the set of decision examples and based on these associations a set of rules is generated. In the second step, each constructed rule is generalized based on statistically significant attributes and context information. In the third step, policy domains are analyzed to augment the rules with restrictions as for some application domains (e.g. security) generalization can have undesired consequences. Policies learned by those stages are safe generalizations with minimal overfitting. To improve the completeness of the learned set, Polisma applies a ML classifier to decision examples not covered by the learned rules; it uses the classification result to generate additional rules in an "ad-hoc" manner.

6.1.3 Problem Definition

Policy learning algorithms typically analyze past access requests associated with their control decisions (as defined in Definition 6.1.1) and aims at generating a set of ABAC rules to control any access request.

Definition 6.1.1 (Access Control Decisions and Examples (l)) *An access control decision is a tuple $\langle u, r, o, d\rangle$ where u is the user who initiated the access request, r is the resource target of the access request, o is the operation requested on the resource, and d is the decision taken for the access request. A decision example l is a tuple $\langle u, e_U, r, e_R, o, d\rangle$ where e_U and e_R are a user attribute expression, and a re-*

source attribute expression such that $u \models e_U$, and $r \models e_R$, and the other arguments are interpreted as in an access control decision.

In a coalition environment, different parties are involved and each party might encounter different scenarios that enrich their experience with respect to control several access requests. However, such expertise might not be similar for all parties. These differences open the opportunity for collaboration among the parties and sharng of their experience and access control decisions. Therefore, in this work we assume there are two parties that are willing to exchange knowledge about access control decisions. In particular, one party has a set of ABAC policies (referred to source ABAC policies) and another party has a set of access control decision examples, extracted from a system history of access requests and their corresponding authorized/denied decisions, together with some context information (e.g., metadata obtained from LDAP directories). The problem is to generate "local" ABAC policies at the second party utilizing both a local log as well as the ABAC policies from the other party. More precisely:

Definition 6.1.2 (Transferring ABAC Policies by Local Examples and Context ($TAPEC$)) *In an environment that consists of different parties, given information of two parties (source and target) including:*

- *a local set of access control decision examples (i.e., $\mathcal{D} = \{l_1, l_2, \ldots, l_n\}$) for the target party,*
- *local context information (the sets $\mathcal{U}$, $\mathcal{R}$, $\mathcal{O}$, the sets A_U and A_R, one of which could be empty, and the functions assigning values to the attributes of the users and resources) for the target party*
- *a set of ABAC rules (i.e., $\mathcal{P}_S = \{\rho_1 \ldots \rho_m\}$) implemented or learned at the source party,*

$TAPEC$ aims at generating a new set of ABAC rules (i.e., $\mathcal{P}_L = \{\rho_1 \ldots \rho_w\}$) that are able to control access requests at the target party.

6.2 Methodology

Transferring ABAC policies from a party to another includes three operations. First, the rules from the source party are compared with access request decision examples and the generated rules at the target party. Second, some rules potentially require adaptation to accommodate the differences between the domains, context, and obligations of the target party. Thus an adaptation process has to be executed. Finally, the original rules (from the source party) along with the new rules generated from the adaptation process, as well as the context information of the target party, are incorporated to enrich the target party with the security requirements from the source party.

6.2.1 Rule Similarity Analysis

The first step for transferring policies from a source party to a target party is to assess the similarity of each source rule with respect to the log of access control decision examples available at the target party. This process includes verifying that an access control decision example satisfies a rule of interest. Specifically, the verification comprises checking three conditions (see Definition 6.2.1): a) the user of the decision example satisfies the user attribute expression of the rule; b) the resource of the decision example satisfies the resource attribute expression of the rule; and c) the operation of the decision example is included in the rule operations set. Moreover, the target party might generate a set of rules (as we will discuss in Subsection 6.2.3). In this case, the policy transfer procedure requires checking the similarity between the source and target rules (see Definition 6.2.3).

Recognizing rule similarity enables next checking their consistency. A rule is consistent with another similar rule if their corresponding decisions are identical (i.e., both of them are *deny* or *permit*); otherwise they are inconsistent. Similarly, an access control decision example is consistent with a similar rule if their decisions are

identical. The consistency of a rule with respect to a decision example or another rule is formally defined in Definitions 6.2.2 and 6.2.4, respectively.

Definition 6.2.1 (Similarity of ABAC Rule and Access Control Decision Example) *Given an ABAC rule* $\rho = \langle e_U, e_R, O, d \rangle$ *and an access control decision example* $l = \langle u, e_U, r, e_R, o, d \rangle$, l *is similar to* ρ *(i.e.,* $l \approx \rho$*) if and only if:*

- $(l.u \models \rho.e_U \vee l.e_U \subseteq \rho.e_U)$,
- $(l.r \models \rho.e_R \vee l.e_R \subseteq \rho.e_R)$, *and*
- $l.o \subseteq \rho.O$.

Definition 6.2.2 (Consistency of ABAC Rule and Access Control Decision Example) *An ABAC rule* ρ *and access control decision example* l *are consistent (i.e.,* $\rho \simeq l$*) if and only if:*

- $\rho \approx l$, *and*
- $\rho.d = l.d$.

Definition 6.2.3 (Similarity of a Pair of ABAC Rules) *Given two ABAC rules* $\rho_i = \langle e_U, e_R, O, d \rangle$ *and* $\rho_j = \langle e_U, e_R, O, d \rangle$, ρ_i *and* ρ_j *are similar to each other (i.e.,* $\rho_i \approx \rho_j$*) if and only if:*

- $(\rho_i.e_U \subseteq \rho_j.e_U \vee \rho_i.u \models \rho_j.e_U) \vee (\rho_j.e_U \subseteq \rho_i.e_U \vee \rho_j.u \models \rho_i.e_U)$,
- $(\rho_i.e_R \subseteq \rho_j.e_R \vee \rho_i.r \models \rho_j.e_R) \vee (\rho_j.e_R \subseteq \rho_i.e_R \vee \rho_j.r \models \rho_i.e_R)$, *and*
- $(\rho_i.o \subseteq \rho_j.O) \vee (\rho_j.o \subseteq \rho_i.O)$.

Definition 6.2.4 (Consistency of a Pair of ABAC Rules) *Two rules* ρ_i *and* ρ_j *are consistent (i.e.,* $\rho_i \simeq \rho_j$*) if and only if:*

- $\rho_i \approx \rho_j$, *and*
- $\rho_i.d = \rho_j.d$.

6.2.2 Rules Adaptation

The second operation for policy transfer is to adapt the rules that are identified as inconsistent ones after conducting the similarity analysis. A straightforward approach to resolve the inconsistency between two rules (or a rule and an access request decision example) is to apply one of the policy-combining-algorithms used in XACML (e.g., deny overrides, permit overrides, and first applicable). However, prioritizing one of the rules on the other and ignoring the one with the least priority may lead to unintended consequences that increase over-privileged or under-privileged accesses. Thus, to avoid such scenarios it is safer to adapt the inconsistent rules in a way that resolves the inconsistency and preserves the intended privilege at the same time.

Adaptation of Two Inconsistent Rules

Resolving the conflict of two inconsistent rules (ρ_i, ρ_j) is performed by first identifying the mutual and non-mutual rule predicates (i.e., which operations, users and resources) between the two rules (see Definition 6.2.6), and then deriving new rules from the original ones based on either the identified mutual or non-mutual conditions (see Algorithm 6.1). Therefore, the first derived rule is generated based on the mutual rule predicates while setting the access control decision as either always "permit" (i.e., permissive paradigm) or always "deny" (restrictive paradigm) (see lines 2-3 in Algorithm 6.1). This strategy is analogous to the XACML combining algorithms applied when encountering a conflict. Using either permissive or restrictive paradigm is heuristically decided based on the number of conflicting decision examples that are resolved by each paradigm (see lines 4-10 in Algorithm 6.1). Next, new rules are generated by adapting the original rules using the non-mutual rule predicates. In this case, the adaptation is performed by employing the following mechanisms on both the inconsistent rules.

- *user-based rule adaptation*: preserve the decision of the access request for only the subset of non-mutual users while the rule resource expression and operations remain the same (see lines 12-13 in Algorithm 6.1);

- *resource-based rule adaptation*: preserve the decision of the access request for only the subset of non-mutual resources while the rule user expression and operations remain the same (see lines 15-16 in Algorithm 6.1); and

- *operation-based rule adaptation*: preserve the decision of the access request for only the subset of non-mutual operations while the rule user and resource expressions remain the same (see lines 18-19 in Algorithm 6.1).

Nonetheless, employing only one of these mechanisms may result in a loss of some authorizations which are specified in the original rules. Thus, all of these mechanisms are applied in sequence. Nonetheless, to perform each of the adaptation mechanisms successfully, the corresponding non-mutual predicate should be refer to an empty predicate (i.e., $U_n \neq \phi \vee R_n \neq \phi \vee O_n \neq \phi$). Finally, the original inconsistent rules are overridden by the rules derived from both of mutual and non-mutual rule predicates.

Definition 6.2.5 (Mutual Predicates of ABAC Rules) *Given two rules* $\rho_i = \langle e_U, e_R, O, d\rangle$ *and* $\rho_j = \langle e_U, e_R, O, d\rangle$*, the mutual predicates comprise:*

- *the mutual user expression* U_m*: the intersection between the user expressions of both rules (i.e.,* $U_m = \{\rho_i.e_U \cap \rho_j.e_U\}$*).*

- *the mutual resource expression* R_m*: the intersection between the resource expressions of both rules (i.e.,* $R_m = \{\rho_i.e_R \cap \rho_j.e_R\}$*).*

- *the mutual operations* O_m*: the subset of operations that are part of the operations of both rules (i.e.,* $O_m = \{\rho_i.O \cap \rho_j.O\}$*).*

Such that $(U_m \neq \phi) \wedge (R_m \neq \phi) \wedge (O_m \neq \phi)$*.*

Definition 6.2.6 (Non-Mutual Predicates of ABAC Rules) *Given two rules* $\rho_i = \langle e_U, e_R, O, d\rangle$ *and* $\rho_j = \langle e_U, e_R, O, d\rangle$*, the non-mutual predicates comprise:*

- *the non-mutual user expression U_n: the user expression that is a subset of the user expression of "only one" of the two rules (i.e., $U_n = \{(\rho_i.e_U \setminus \rho_j.e_U) \vee (\rho_j.e_U \setminus \rho_i.e_U)\}$.*

- *the non-mutual resource expression U_n: the resource expression that is a subset of the resource expression of "only one" of the two rules (i.e., $R_n = \{(\rho_i.e_R \setminus \rho_j.e_R) \vee (\rho_j.e_R \setminus \rho_i.e_R)\}$.*

- *the non-mutual operations O_n: the subset of operations are part of the operations of "only one" of the two rules (i.e., $O_n = \{(\rho_i.O \setminus \rho_j.O) \vee (\rho_j.O \setminus \rho_i.O)\}$.*

Example: Given two inconsistent rules $\rho_1 = \langle$ {dept id: 9, 10, 11}, {resource id: 1, 2, 3}, {read, write}, permit $\rangle$, $\rho_2 = \langle$ {dept id: 9, 12}, {resource id: 1}, {read}, deny $\rangle$, the resolution is performed as follows:
For the non-mutual predicates: By the user-based adaptation, the new rules are:

- $\rho'_1 = \langle$ {dept id: 10,11}, {resource id: 1,2,3}, {read,write}, permit $\rangle$

- $\rho'_2 = \langle$ {dept id: 12}, {resource id: 1}, {read}, deny $\rangle$

By the resource-based adaptation, the new rules are:

- $\rho''_1 = \langle$ {dept id: 9,10,11}, {resource id: 2,3}, {read,write}, permit $\rangle$

By the operation-based adaptation, the new rules are:

- $\rho'''_1 = \langle$ {dept id: 9,10,11}, {resource id: 1,2,3}, {write}, permit $\rangle$.

For the mutual predicates:

- *According to the permissive paradigm, the new rule is : $\rho''_2 = \langle$ {dept id: 9}, {resource id: 1}, {read}, permit $\rangle$*

- *According the restrictive paradigm, the new rule is : $\rho'''_2 = \langle$ {dept id: 9}, {resource id: 1}, {read}, deny $\rangle$*

Eventually, when adapting two inconsistent rules, at maximum six rules can be generated based on the non-mutual predicates, while one rule at maximum can be generated based on the mutual predicates.

Algorithm 6.1 Adapt Two Inconsistent ABAC Rules (*Adapt2Rules*)

Require: $\mathcal{D}$: Access control decision examples and (ρ_i, ρ_j) a pair of inconsistent ABAC rules

$\mathcal{P} = \{\}$ # New adapted rules.

$\rho_k = \langle\ \rho_i.e_U \cap \rho_j.e_U,\ \rho_i.e_R \cap \rho_j.e_R,\ \rho_i.o \cap \rho_j.o,\ permit\ \rangle$

$\rho'_k = \langle\ \rho_i.e_U \cap \rho_j.e_U,\ \rho_i.e_R \cap \rho_j.e_R,\ \rho_i.o \cap \rho_j.o,\ deny\ \rangle$

$\mathcal{D}_{\rho_k} = \{\forall l \in \mathcal{D} \mid (l \simeq \rho_k\ \}$

$\mathcal{D}_{\rho'_k} = \{\forall l \in \mathcal{D} \mid (l \simeq \rho'_k\ \}$

if $\mid \mathcal{D}_{\rho_k} \mid > \mid \mathcal{D}_{\rho'_k} \mid$ **then**

  $\mathcal{P} = \mathcal{P} \cup \rho_k$

else

  $\mathcal{P} = \mathcal{P} \cup \rho_{k\prime}$

end if

User-based Adaptation.

$\rho'_i = \langle\ \rho_i.e_U \setminus \rho_j.e_U,\ \rho_i.e_R,\ \rho_i.O,\ \rho_i.d\ \rangle$

$\rho'_j = \langle\ \rho_j.e_U \setminus \rho_i.e_U,\ \rho_j.e_R,\ \rho_j.O,\ \rho_j.d\ \rangle$

Resource-based Adaptation.

$\rho''_i = \langle\ \rho_i.e_U,\ \rho_i.e_R \setminus \rho_j.e_R,\ \rho_i.O,\ \rho_i.d\ \rangle$

$\rho''_j = \langle\ \rho_j.e_U,\ \rho_j.e_R \setminus \rho_i.e_R,\ \rho_j.O,\ \rho_j.d\ \rangle$

Operation-based Adaptation.

$\rho'''_i = \langle\ \rho_i.e_U,\ \rho_i.e_R,\ \rho_i.O \setminus \rho_j.O,\ \rho_i.d\ \rangle$

$\rho'''_j = \langle\ \rho_j.e_U,\ \rho_j.e_R,\ \rho_j.O \setminus \rho_i.O,\ \rho_j.d\ \rangle$

$\mathcal{P} = \mathcal{P} \cup \rho'_i \cup \rho'_j \cup \rho''_i \cup \rho''_j \cup \rho'''_i \cup \rho'''_j$

return $\mathcal{P}$

Adaptation of Groups of Inconsistent Rules

When an ABAC rule contradicts a set of ABAC rules, one approach is to execute the resolution algorithm separately for each pair of inconsistent rules (i.e., Algo-

rithm 6.1). However, this might not be as simple as it seems. Applying such an approach potentially generates additional conflicts because the newly adapted rules are generated by considering only a pair of rules but not the other ones. Thus, a recursive adaptation is performed until no further inconsistency is encountered within the original group of inconsistent rules and the ones adapted from them (i.e., the newly generated ones).

6.2.3 Rule Transferability Approaches

Naïve Approaches

Transfer Policies using a Local Log ($TPLG$)

The straightforward approach for transferring policies is to use the raw log of historical access control decisions collected at the target system for adapting the source rules to be used in the system of interest. In particular, the rules are tuned using the local log, as illustrated in Algorithm 6.4 and shown in Fig. 6.2. Towards this, first the source rules are enforced using the historical set of access control requests available.

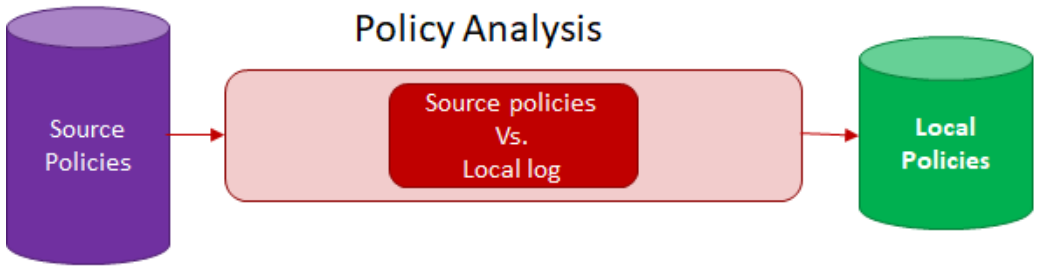


Fig. 6.2.: Transfer Policies using Local Log ($TPLG$)

After the enforcement, the approach inspects the decision of each rule for the corresponding examined access requests compared to the historical access control decisions. The inspection can results in three cases: a) the rule has not been enforced

by any of the historical requests, b) the rule has been enforced with no inconsistency with any of the historical decisions, c) the rule has been enforced with inconsistency with respect to the historical decisions. In the first case, the rule is transferred in order to be used for handling situations not yet encountered. For the second case, the rule is also transferred because of its compliance with the local historical access control decisions. Finally, in the third case, the rule is adapted by restricting its attribute expressions to fit with either only the corresponding historical decision requests that comply with or the contradicting set of historical decision requests.

Transfer Policies using Local Policies ($TPLP$)

$TPLG$ is easy to implement; however, the main limitation of this approach is that it utilizes the raw local log for tuning the source policies directly without analyzing the log. Such an approach assumes that the observation of each historical access control decision represents a rule; however, each rule is typically an abstraction of security specifications for multiple access control decisions. Alternatively, another approach is to utilize the local log for generating ABAC rules referred to as "local policies" using one of the state-of-the-art policy learning approaches [49] [136] [140].

In general, the approach is composed of two main steps (see Algorithm 6.5 and Fig. 6.3). First, the approach uses a policy learner to generate local rules using the local log. Thereafter, a similarity analysis is performed on both local and source rules. The outcome of such analysis has three cases: a) no local rule is similar to a source rule (see lines 11-12); b) all of the local rules which are similar to a source rule are consistent with it (see lines 6-7); and c) some of the local rules which are similar to a source rule are inconsistent with it (see lines 8-10). In the first and the second cases, the source rule is migrated to the target system. In the third case, the conflicts between the local and source rules are resolved by performing the adaptation algorithm such that their corresponding attribute expressions are tailored either by expanding or restricting their covered scope.

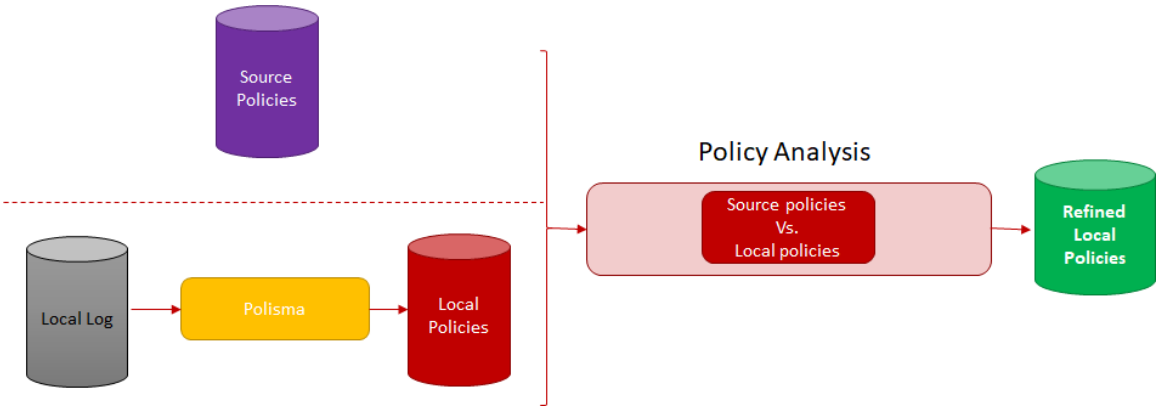


Fig. 6.3.: Transfer Policies using Local Policies ($TPLP$)

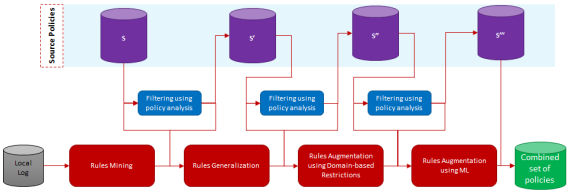


Fig. 6.4.: Transfer Policies using Local Learning

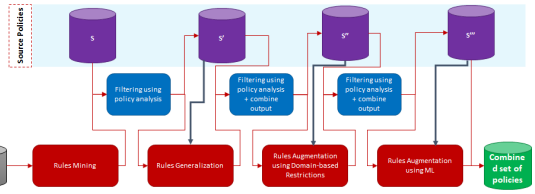


Fig. 6.5.: Transfer Policies using Hybrid Learning

Transfer Policies using Local Learning ($TPLL$)

$TPLP$ post-processes the generated rules from a policy learner using the source rules. However, such a mechanism potentially generates local rules that are inconsistent with the source rules and also increases the conflicts among the rules learned throughout the learning stages. One approach to avoid such a problem is to use the source rules to adapt the intermediate forms of the local rules. In particular, the rule adaptation is performed in tandem with policy learning from the local log.

Since a policy learner is typically composed of multiple learning phases, the intermediate rules generated after each step are compared with the source rules using a similarity analysis (see Algorithm 6.6 and Fig. 6.4). If these intermediate local rules conflicts with the source rules, the local rules are adapted in the early stages; hence enabling the evolution of more accurate local rules among the phases of the policy learner (i.e., reducing error propagation) while filtering the source policies to exclude the ones that have a conflict with any of the local rules. Moreover, this mechanism enables either increasing or decreasing the significance of intermediate rules and controlling their evolution. After the local rules are generated, the remaining source rules are transferred to the target system.

Transfer Policies using Hybrid Learning ($TPHL$)

$TPLL$ does not fully exploit the source rules while learning the local rules from the local log since $TPLL$ only uses the source rules for controlling the evolution of the intermediate rules. Therefore, we propose another approach in which the source rules are in-lined with the intermediate local rules to allow the next learning phases to exploit the source rules as well as the local log in the learning stages. Such an approach allows one to generate correct rules that cover further security aspects and requirements of the local system.

As shown in Fig. 6.5, the source rules are used in two ways in this approach. First, the intermediate local rules are adapted after each learning phase of the policy learner. Second, the filtered source rules (by excluding the ones that have a conflict with any of the local rules) are used as input for the subsequent learning phases alongside the intermediate local rules. The steps of $TPHL$ are illustrated in Algorithm 6.7.

6.3 Evaluation

In this section, we summarize the experimental methodology and report the evaluation results for *FLAP*.

6.3.1 Experimental Methodology

Dataset. We conducted experiments using two datasets: a synthetic dataset (referred to as project management (PM) dataset [136]) and real dataset (referred to as Amazon dataset), obtained from Amazon[1]. This dataset is an anonymized sample of accesses provisioned within the Amazon company and it is composed of around 700K decision examples. We used 70% of the dataset as decision examples of the target party and 30% for the source party.

Evaluation Metrics. To evaluate $FLAP$, we use three metrics: precision, recall, and F1 score. These metrics are able of conveying the correctness of the adapted policies, by checking their responses to the local decision examples, and their ability to cover new access requests derived from the source policies.

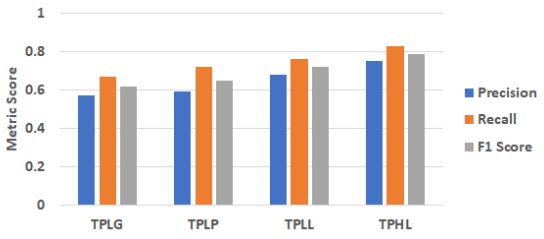


Fig. 6.6.: Transfer Policies using the PM dataset

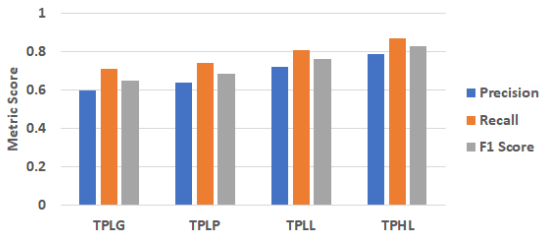


Fig. 6.7.: Transfer Policies using the Amazon dataset

6.3.2 Experimental Results

Figs 6.6 and 6.7 shows the evaluation results for the four approaches introduced in the paper, that is, $TPLG$, $TPLP$, $TPLL$, and $TPHL$, on the PM and Amazon datasets, respectively. In general, both $TPLL$ and $TPHL$ perform better than $TPLG$, $TPLP$ in terms of all evaluation metrics due to the fact that both $TPLL$ and $TPHL$ utilize the source policies along with the local log in the learning pro-

[1] `http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples`

cess to filter and adapt the rules including the intermediate ones; hence reducing the effect of the error propagation. On the other hand, $TPLG$ and $TPLP$ perform the worst because $TPLG$ uses "only" the local log for adapting the source policies without analyzing the local log while $TPLP$ uses the local log to learn local rules for the target system without utilizing the source rules. Moreover, the results on the Amazon dataset are better than that of the PM dataset because the Amazon dataset contains more decision examples in the local log and more source policies compared to the PM dataset, and thus, such an increase in the input data positively affects the transfer process. Figs 6.6 and 6.7 show the evaluation results with respect to three metrics: precision, recall, and F1 score. The recall metric reflects the ability of the rules generated of the transfer process to correctly control more scenarios, while the precision metric reflects the correctness of the decisions produced by the generated rules[2].

6.4 Related Work to Policy Transfer

The problem of ABAC policy transfer has not yet been investigated. Approaches have been proposed for policy adaptation in the context of mobile systems, such as [148]. However the policies considered in such approaches are not attribute-based. Work related to ours also includes work on mining ABAC policies from access control decision logs [140], [49], [147], [144], [136], and work on ABAC policy similarity [83], [82]. However such approaches do not address policy migration and adaptation. More recently approaches have been proposed that apply transfer learning techniques in the context of intrusion detection [149], [150], [151], [152], [153]. However those approaches have been designed for transferring neural networks used for classifying network packets as benign or malicious and thus do not deal with rule transfer and adaptation.

[2]F1 score is the harmonic mean of precision and recall.

Algorithm 6.2 Subtract Rules ($SubAttrFromRule$)

Require: An ABAC rule ρ_0 inconsistent with a group of rules ($\mathcal{P}_\mathcal{C} = \{\rho_1, \ldots, \rho_m\}$) and $\mathcal{P}$ the set of resulted rules

if $\mid \mathcal{P}_\mathcal{C} \mid = 0$ **then**
  return ρ_0
else
  for $a_i \in (\rho_0.a_U \cup \rho_0.a_R \cup \rho_0.O)$ **do**
    # ρ_1 is first rule of $\mathcal{P}_\mathcal{C}$
    $\rho_1 = \mathcal{P}_\mathcal{C} \setminus \{\forall \rho_i \in \mathcal{P}_\mathcal{C} \mid i > 1\}$
    if $a_i \in \rho_0.a_U$ **then**
      $e_{Ua0} = \{(a_i, d_U(u, a_i)) \mid \forall u \models \rho_0.a_U\}$
      $e_{Ua1} = \{(a_i, d_U(u, a_i)) \mid \forall u \models \rho_1.a_U\}$
      $a'_U = \{\forall u \models e_{Ua0}\} \setminus \{\forall u \models e_{Ua1}\}$
      $\rho'_0 = \langle a'_U, \rho_0.a_R, \rho_0.O, \rho_0.d \rangle$
    else if $a_i \in \rho_0.a_R$ **then**
      $e_{Ra0} = \{(a_i, d_R(r, a_i)) \mid \forall r \models \rho_0.a_R\}$
      $e_{Ra1} = \{(a_i, d_R(r, a_i)) \mid \forall r \models \rho_1.a_R\}$
      $a'_R = \{\forall r \models e_{Ra0}\} \setminus \{\forall r \models e_{Ra1}\}$
      $\rho'_0 = \langle \rho_0.a_U, a'_R, \rho_0.O, \rho_0.d \rangle$
    else if $a_i \in \rho_0.O$ **then**
      $a'_O = \rho_0.O \setminus \rho_1.O$
      $\rho'_0 = \langle \rho_0.a_U, \rho_0.a_R, a'_O, \rho_0.d \rangle$
    end if
    if $\rho'_0 \neq \phi$ **then**
      $\rho_0 = \rho'_0$
      # Remove the 1^{st} rule from $\mathcal{P}_\mathcal{C}$
      $\mathcal{P}_\mathcal{C} = \{\forall \rho_i \in \mathcal{P}_\mathcal{C} \mid i > 1\}$
      $\mathcal{P} = \mathcal{P} \cup SubAttrFromRule(\rho_0, \mathcal{P}_\mathcal{C}, \mathcal{P})$
    end if
  end for
end if
return $\mathcal{P}$

Algorithm 6.3 Adapt a Group of Inconsistent ABAC Rules (*AdaptGRules*)

Require: $\mathcal{D}$: Access control decision examples and an ABAC rule ρ_0 inconsistent with a group of rules ($\mathcal{P}_\mathcal{G} = \{\rho_1, \ldots, \rho_m\}$)

1: $\mathcal{P}_\mathcal{C} = \{\}$, $\mathcal{P} = \{\}$
2: **for** $\rho_i \in \mathcal{P}_\mathcal{G}$ **do**
3: $\quad \mathcal{P}_\mathcal{C} = \mathcal{P}_C \cup (\rho_0 \cap \rho_i)$
4: **end for**
5: $\mathcal{P} = SubAttrFromRule(\rho_0, \mathcal{P}_\mathcal{C}, \mathcal{P})$
6: **for** $\rho_i \in \mathcal{P}_\mathcal{G}$ **do**
7: $\quad \mathcal{P} = \mathcal{P} \cup SubAttrFromRule(\rho_i, \{\rho_0 \cap \rho_i\}, \mathcal{P})$
8: **end for**
9: **for** $\rho_i \in \mathcal{P}_\mathcal{C}$ **do**
10: $\quad \rho_k = \langle\ \rho_i.e_U,\ \rho_i.e_R,\ \rho_i.o,\ permit\ \rangle$
11: $\quad \rho'_k = \langle\ \rho_i.e_U,\ \rho_i.e_R,\ \rho_i.o,\ deny\ \rangle$
12: $\quad \mathcal{D}_{\rho_k} = \{\forall l \in \mathcal{D} \mid (l \simeq \rho_k\ \}$
13: $\quad \mathcal{D}_{\rho'_k} = \{\forall l \in \mathcal{D} \mid (l \simeq \rho'_k\ \}$
14: $\quad$ **if** $\mid \mathcal{D}_{\rho_k} \mid > \mid \mathcal{D}_{\rho'_k} \mid$ **then**
15: $\quad\quad \mathcal{P} = \mathcal{P} \cup \rho_k$
16: $\quad$ **else**
17: $\quad\quad \mathcal{P} = \mathcal{P} \cup \rho_{k\prime}$
18: $\quad$ **end if**
19: **end for**
20: **return** $\mathcal{P}$

Algorithm 6.4 Transfer Policies using Local Log ($TPLG$)

Require: $\mathcal{D}$: Access control decision examples, $\mathcal{P}_S$: "Source" access control policies.

$\mathcal{P}_L = \{\}$ # Adapted "Local" access control policies.
for $\rho_i \in \mathcal{P}_S$ **do**
  $\mathcal{D}_{\rho_i} = \{\forall l \in \mathcal{D} \mid l \approx \rho_i\}$
  $\mathcal{D}^m_{\rho_i} = \{\forall l \in \mathcal{D}_{\rho_i} \mid l \simeq \rho_i\}$
  $\mathcal{D}^c_{\rho_i} = \{\forall l \in \mathcal{D}_{\rho_i} \mid l \not\simeq \rho_i\}$
  if $\mathcal{D}^m_{\rho_i} \neq \phi \wedge \mathcal{D}^c_{\rho_i} = \phi$ **then**
    $\mathcal{P}_L = \mathcal{P}_L \cup \rho_i$
  else if $\mathcal{D}^c_{\rho_i} \neq \phi$ **then**
    $\mathcal{P}^{c'}_{\rho_i} = \{\}$ # Adapted conflicting "Local" access control policies.
    $\mathcal{P}^{c'}_{\rho_i} = AdaptGRules(\mathcal{D}, \rho_i, \mathcal{D}^{c'}_{\rho_i})$
    $\mathcal{P}_L = \mathcal{P}_L \cup \mathcal{P}^{c'}_{\rho_i}$
  else if $\mathcal{D}^m_{\rho_i} = \phi \wedge \mathcal{D}^c_{\rho_i} = \phi$ **then**
    $\mathcal{P}_L = \mathcal{P}_L \cup \rho_i$
  end if
end for
return $\mathcal{P}_L$

Algorithm 6.5 Transfer Policies using Local Policies ($TPLP$)

Require: $\mathcal{P}_S$: "Source" access control policies, $\mathcal{P}_L$: "Local" access control policies.

1: $\mathcal{P}'_L = \{\}$ # "Adapted Local" access control policies.
2: **for** $\rho_i \in \mathcal{P}_S$ **do**
3: $\quad \mathcal{P}_{\rho_i} = \{\forall p_j \in \mathcal{P}_L \mid p_j \approx \rho_i \}$
4: $\quad \mathcal{P}^m_{\rho_i} = \{\forall p_k \in \mathcal{P}_{\rho_i} \mid p_k \simeq \rho_i\}$
5: $\quad \mathcal{P}^c_{\rho_i} = \{\forall p_k \in \mathcal{P}_{\rho_i} \mid p_k \not\simeq \rho_i\}$
6: $\quad$ **if** $\mathcal{P}^m_{\rho_i} \neq \phi \wedge \mathcal{P}^c_{\rho_i} = \phi$ **then**
7: $\quad\quad \mathcal{P}'_L = \mathcal{P}'_L \cup \rho_i$
8: $\quad$ **else if** $\mathcal{P}^c_{\rho_i} \neq \phi$ **then**
9: $\quad\quad \mathcal{P}^{c'}_{\rho_i} = \{\}$ # Adapted conflicting "Local" access control policies.
10: $\quad\quad \mathcal{P}^{c'}_{\rho_i} = AdaptGRules(\mathcal{D}, \rho_i, \mathcal{P}^c_{\rho_i})$
11: $\quad\quad \mathcal{P}'_L = \mathcal{P}'_L \cup \mathcal{P}^{c'}_{\rho_i}$
12: $\quad$ **else if** $\mathcal{P}^m_{\rho_i} = \phi \wedge \mathcal{P}^c_{\rho_i} = \phi$ **then**
13: $\quad\quad \mathcal{P}'_L = \mathcal{P}'_L \cup \rho_i$
14: $\quad$ **end if**
15: **end for**
16: **return** $\mathcal{P}'_L$

Algorithm 6.6 Transfer Policies using Local Learning ($TPLL$)

Require: $\mathcal{D}$: Access control decision examples, $\mathcal{P}_S$: "Source" access control policies,

: a policy learner consisted of n steps $\{_1, _2, \ldots, _n\}$.

$\mathcal{P}'_S = \mathcal{P}_S$ # "Adapted Source" access control policies.

$\mathcal{P}'_L = \{\}$ # "Adapted Local" access control policies.

for $_k \in$ **do**

  $\mathcal{P}_L = _k(\mathcal{D}, \mathcal{P}'_L)$

  for $\rho_i \in \mathcal{P}'_S$ **do**

    $\mathcal{P}_{\rho_i} = \{\forall p_j \in \mathcal{P}_L \mid p_j \approx \rho_i \}$

    $\mathcal{P}^m_{\rho_i} = \{\forall p_k \in \mathcal{P}_{\rho_i} \mid p_k \simeq \rho_i\}$

    $\mathcal{P}^c_{\rho_i} = \{\forall p_k \in \mathcal{P}_{\rho_i} \mid p_k \not\simeq \rho_i\}$

    if $\mathcal{P}^{\rho_i}_c \neq \phi$ **then**

      $\mathcal{P}^{c'}_{\rho_i} = \{\}$ # Adapted conflicting "Local" access control policies.

      $\mathcal{P}^{c'}_{\rho_i} = AdaptGRules(\mathcal{D}, \rho_i, \mathcal{P}^c_{\rho_i})$

      $\mathcal{P}'_S = \mathcal{P}'_S - \rho_i$

      $\mathcal{P}_L = \mathcal{P}_L - \mathcal{P}^c_{\rho_i}$

      $\mathcal{P}_L = \mathcal{P}_L \cup \mathcal{P}^{c'}_{\rho_i}$

    end if

  end for

  $\mathcal{P}'_L = \mathcal{P}_L$

end for

$\mathcal{P}'_L = \mathcal{P}'_L \cup \mathcal{P}'_S$

return $\mathcal{P}'_L$

Algorithm 6.7 Transfer Policies using Hybrid Learning ($TPHL$)

Require: $\mathcal{D}$: Access control decision examples, $\mathcal{P}_S$: "Source" access control policies,

: a policy learner consisted of n steps $\{_1, _2, \ldots, _n\}$.

$\mathcal{P}'_S = \mathcal{P}_S$.

$\mathcal{P}'_L = \{\}$.

for $_k \in$ **do**

  $\mathcal{P}_L = _k(\mathcal{D}, \mathcal{P}'_L)$

  for $\rho_i \in \mathcal{P}'_S$ **do**

    $\mathcal{P}_{\rho_i} = \{\forall p_j \in \mathcal{P}_L \mid p_j \approx \rho_i \}$

    $\mathcal{P}^m_{\rho_i} = \{\forall p_k \in \mathcal{P}_{\rho_i} \mid p_k \simeq \rho_i\}$

    $\mathcal{P}^c_{\rho_i} = \{\forall p_k \in \mathcal{P}_{\rho_i} \mid p_k \not\simeq \rho_i\}$

    if $\mathcal{P}^c_{\rho_i} \neq \phi$ **then**

      $\mathcal{P}^{c'}_{\rho_i} = \{\}$ # Adapted conflicting "Local" access control policies.

      $\mathcal{P}^{c'}_{\rho_i} = AdaptGRules(\mathcal{D}, \rho_i, \mathcal{P}^c_{\rho_i})$

      $\mathcal{P}'_S = \mathcal{P}'_S - \rho_i$

      $\mathcal{P}_L = \mathcal{P}_L - \rho_i$

      $\mathcal{P}_L = \mathcal{P}_L - \mathcal{P}^{\rho_i}_c$

      $\mathcal{P}_L = \mathcal{P}_L \cup \mathcal{P}^{c'}_{\rho_i}$

    end if

  end for

  $\mathcal{P}_L = \mathcal{P}_L \cup \mathcal{P}'_S$

  $\mathcal{P}'_L = \mathcal{P}_L$

end for

return $\mathcal{P}'_L$

7. CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this thesis, we present a provenance framework in addition to a set of applications utilizing provenance metadata. In particular, these provenance applications include auditing, quality assessment, and reproducibility.

Data Provenance Framework. In Chapter 2, we have introduced SimP [16], a comprehensive provenance framework that is built on a comprehensive provenance model provided with relational and graph specifications. Our provenance model is interoperable with the OPM and PROV provenance models. In addition, SimP includes a unified provenance query language (QL-SimP) [41, 154] which is independent of the provenance representation. Based on our benchmark, the performance for provenance queries which require path traversal is better when using a graph database while the performance for structural provenance queries is better when using a relational database.

Provenance for Assessing the Quality of Access Control Policies. In Chapter 3, we have proposed a set of requirements to evaluate the quality of access control policies. We have also shown the use of provenance for capturing fine-grained metadata essential for evaluating the quality of policies. Our framework (ProFact) [26, 27] supports various types of query services which convey detailed information about the system environment in the context of transactions and access control policies. ProFact supports two approaches for policy analysis: structure-based and classification-based. Regarding the structure-based approach, our experiments show that transaction-based analysis is faster than policy-based analysis with a

maximum speedup factor of 5X. Regarding the classification-based analysis approach, kNN and SVM achieved the best efficiency obtaining a maximum recall of 80% and 86%, respectively. By adopting the combined classifiers, the efficiency improved reaching a recall of 88%. Moreover, classification-based approach outperformed the structure-based approach with a maximum speedup factor of 31x. If the system is not in a real-time environment, the system can adopt the structure-based analysis approach to obtain a 100% recall.

Provenance for Reproducing Workflows of Repetitive Tasks. In Chapter 4, we have presented an architecture to facilitate creating scientific workflow repositories and enable searching them efficiently based on the provenance representation of scientific workflows. Our architecture (ProWS) [6] supports composite queries comprising three subqueries, namely: metadata-based, label-based, and pattern-based queries. We have underpinned our formalisms with respect to logical constructs and have proposed efficient techniques to match search parameters with provenance details of the workflows. To enhance the search, we proposed a set of index structures. Based on our experiments, our index-based approaches are able to enhance the retrieval of similar workflows efficiently by minimizing the query time and avoiding the traversal of "false hit" workflows.

Provenance for Learning Attribute-based Access Control Policies. In Chapter 5, we have proposed *Polisma* [49], a framework for learning ABAC policies from examples and context information. *Polisma* comprises four steps employing various techniques, namely data mining, statistical, and machine learning. Our evaluations, carried out on a real-world decision log (referred to as AZ) and on a synthetic one (referred to as PM), show that *Polisma* is able to learn policies that are both complete and correct. The rules generated by *Polisma* using the PM dataset achieve an F1 score of 0.81 and PCR of 0.95; also, when using the AZ dataset, the generated rules achieve an F1 score of 0.86 and PCR of 0.98. Moreover,

By using the semantic information available with the *PM* dataset, *Polisma* improves the F1 score to reach 0.85.

Provenance for Transferring Attribute-based Access Control Policies. In Chapter 6, we have proposed *FLAP* [155], a framework for policy transfer in a federated environments. It allows one to transfer attribute-based policies from a source party to a target party. We have proposed four approaches that vary in their interaction levels between the source and target resources, as well as the timing of this interaction. Our preliminary evaluation, carried out on a real-world access control decision dataset and a synthetic one, show the ability of *FLAP* to transfer the source policies to a target party to generate correct policies which can be used in future for unseen scenarios.

7.2 Future Work

In future, I intend to extend this work in the following directions.

7.2.1 Provenance Similarity Measure

As discussed earlier, provenance metadata is crucial for several purposes. However, provenance metadata often is quite large in size which leads to restrictions on its usages. Addressing such an issue requires analyzing provenance metadata to extract useful metadata to the application of interest and guarantee its effective and efficient utilization. Such analysis includes multiple methods such as querying, filtering, removing redundancy, and abstraction. All such methods primarily require an effective provenance similarity measure. Evaluating the provenance similarity varies based on the purpose of the application utilizing the provenance metadata. One of the main applications of data provenance is auditing and anomaly detection which require measuring the similarity of application resources (e.g., data objects) effectively. Another emerging application is to utilize provenance metadata for data quality purpose.

Provenance metadata provides rich information about how data has been generated by processes. Hence, we need to devise a provenance similarity measure based on the activities executed in the application of interest (e.g., ProFact [26, 27]). Moreover, provenance can be utilized for data reproducibility. Provenance provides metadata about the sequence of tasks executed by different users for achieving similar goals. Predicting such patterns needs a similarity measure to compare all actions operated by such users. (e.g., ProWS [6]). Therefore, we need to define multi-purpose provenance similarity measure.

Similarity analysis has been investigated in other domain other than data provenance. Lin *et al.* [82, 156] proposed a policy similarity approach mainly based on XACML [19] policies. Their proposed approach is based on information retrieval and is utilized for policy analysis purposes. Another domain is text similarity which includes sentence similarity [157], ontology similarity [158] and document similarity [159]. Such approaches can not be applied in the domain of data provenance since the underlying representation is complex and different.

Provenance analysis has been the focus of some research efforts which include investigating mechanisms for provenance similarity analysis. For example, Biton *et al.* [160] presented a system to retrieve relevant provenance metadata in the context of workflows. The main target of this system was retrieving the most relevant provenance metadata and reducing the amount of returned provenance of users queries. Hence, they focused on abstraction techniques mainly by constructing provenance users views based on workflow specifications. Garijo *et al.* [161] have also focused on provenance abstraction for workflows context. Garijo's approach is to abstract provenance based on finding common tasks according to workflow templates among the provenance logs using graph algorithms. Similarly, Missier *et al.* [162] designed a tool for provenance abstraction using graph transformations. All these works have primarily focused on provenance analysis for the workflows context and mostly using graph-based approaches. Hence, there is a need for a similarity approach which is capable of considering different application domains including workflows.

To address the provenance similarity challenges (i.e., the rich representation of data provenance and the applicability for various contexts), we thus plan to design a provenance similarity measure using two approaches: model checking (e.g., using a SAT solver [163], and a multi-terminal binary decision diagram (MTBDD) [164]), and information retrieval. The first approach is computationally expensive, especially when dealing with a large-scale repository of data provenance. The problem is equivalent to solving Boolean satisfiability, which is NP-complete. The second approach is based on principles from the information retrieval field. This approach uses the notion of provenance similarity measure, based on which a similarity score can be quickly computed for corresponding provenance entities and relations. In our plan, we will investigate the similarity approaches using the SimP model [16] which is interoperable with OPM [12] and PROV [13].

7.2.2 A Provenance-based Trustworthiness Model for Evaluating Human Activities on Social Networks as Valuable Sensors

Recently, online social networks (e.g., Flickr[1], Facebook[2], Instagram[3], Snapchat[4], and Twitter[5]) have been massively used especially with the advances of sensor-rich smartphones equipped with the Internet connectivity. Due to the growth of using these online social networks, humans are uncontrollably reporting and publishing a massive amount of information about various topics. For example, people daily post 500 million tweets on Twitter [165] and 734 million comments on Facebook [166]. Such huge streams of data potentially provide a valuable source of knowledge about the physical environment, social phenomena, and people daily lives. Subsequently, many research works have utilized social media data to investigate public characteristics (e.g., demographic and urban characteristics [167,168], customer opinions about prod-

[1]http://www.flickr.com
[2]http://www.facebook.com
[3]http://www.instagram.com
[4]https://www.snapchat.com
[5]http://www.twitter.com

ucts [169, 170], political views [171], and public health information [172]). Furthermore, social media has been used as active sensors during emergency events [173,174] such as disasters (e.g., flood, earthquake, and hurricane). In particular, the Federal Emergency Management Agency (FEMA) identifies social media as an essential component of future disaster management [175]. Hence, it is crucial to measure the trustworthiness of human posts on social networks.

We thus plan to investigate designing a trustworthiness model based on provenance which can consider the reliability level of an individual posted data with respect to its source among the active users participating in the social network as well as the trust score of each user based on the aggregated value of his posted data. As a result, our trustworthiness model aims at filtering only reliable data for using them in other applications which depend on the available streams of data on online social networks.

The two future steps can be considered as a middle layer between provenance data and its applications. Therefore, such a middle layer can be an additional component for the SimP framework to be ultimately and effectively used in the different applications and contexts.

REFERENCES

REFERENCES

[1] D. Fadolalkarim, A. Sallam, and E. Bertino, "PANDDE: Provenance-based anomaly detection of data exfiltration," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. ACM, 2016, pp. 267–276.

[2] A. Awad, S. Kadry, G. Maddodi, S. Gill, and B. Lee, "Data leakage detection using system call provenance," in *Proceedings of the 2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS)*. IEEE, 2016, pp. 486–491.

[3] C. Dai, D. Lin, E. Bertino, and M. Kantarcioglu, "An approach to evaluate data trustworthiness based on data provenance," in *Proceedings of the 5th VLDB Workshop on Secure Data Management SDM*. Springer, 2008, pp. 82–98.

[4] X. Wang, K. Govindan, and P. Mohapatra, "Provenance-based information trustworthiness evaluation in multi-hop networks," in *Proceedings of the Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[5] N. Prat and S. Madnick, "Measuring data believability: A provenance approach," in *Proceedings of the 41st Hawaii International International Conference on Systems Science (HICSS)*. IEEE, 2008, pp. 393–393.

[6] A. Abu Jabal, E. Bertino, and G. De Mel, "Provenance-based scientific workflow search," in *Proceedings of the 2017 IEEE 13th International Conference on e-Science (e-Science)*. IEEE, 2017, pp. 119–127.

[7] J. Wang, N. Car, B. Evans, L. Wyborn, and E. King, "Supporting data reproducibility at nci using the provenance capture system," *D-Lib Magazine*, vol. 23, no. 1/2, 2017.

[8] I. Foster, J. Vockler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *Proceedings. 14th International Conference on Scientific and Statistical Database Management (SSDBM)*. IEEE, 2002, pp. 37–46.

[9] J. Zhao, C. Goble, R. Stevens, and S. Bechhofer, "Semantically linking and browsing provenance logs for e-science," in *Proceedings of the International Conference on Semantics for the Networked World*. Springer, 2004, pp. 158–176.

[10] Y. L. Simmhan, B. Plale, and D. Gannon, "Query capabilities of the karma provenance framework," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 441–451, 2008.

[11] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems," in *USENIX Annual Technical Conference, General Track*, 2006, pp. 43–56.

[12] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson, "The open provenance model," 2007. [Online]. Available: http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf

[13] "Prov overview." [Online]. Available: http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/

[14] Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han, "An access control language for a general provenance model," in *Workshop on Secure Data Management*. Springer, 2009, pp. 68–88.

[15] S. Sultana and E. Bertino, "A distributed system for the management of fine-grained provenance," *Journal of Database Management (JDM)*, vol. 26, no. 2, pp. 32–47, 2015.

[16] A. Abu Jabal and E. Bertino, "Simp: Secure interoperable multi-granular provenance framework," in *Proceedings of the 2016 IEEE 12th International Conference on e-Science (e-Science)*. IEEE, 2016, pp. 270–275.

[17] E. Bertino, G. Ghinita, and A. Kamra, "Access control for databases: Concepts and systems," *Foundations and Trends® in Databases*, vol. 3, no. 1–2, pp. 1–148, 2011.

[18] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[19] "Extensible access control markup language (xacml) version 3.0." [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

[20] P. Gupta, S. D. Stoller, and Z. Xu, "Abductive analysis of administrative policies in rule-based access control," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 5, pp. 412–424, 2014.

[21] A. Cau, H. Janicke, and B. Moszkowski, "Verification and enforcement of access control policies," *Formal Methods in System Design*, vol. 43, no. 3, pp. 450–492, 2013.

[22] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone, "Analysis of xacml policies with smt," in *Proceedings of the 2015 4th International Conference on Principles of Security and Trust (POST), London, UK, April 11-18, 2015*, 2015, pp. 115–134.

[23] C. Ngo, Y. Demchenko, and C. de Laat, "Decision diagrams for xacml policy evaluation and management," *Computers & Security*, vol. 49, pp. 1–16, 2015.

[24] X. Ma, R. Li, Z. Lu, and W. Wang, "Mining constraints in role-based access control," *Mathematical and Computer Modelling*, vol. 55, no. 1, pp. 87–96, 2012.

[25] A. Abu Jabal, M. Davari, E. Bertino, C. Makaya, S. Calo, D. Verma, A. Russo, and C. Williams, "Methods and tools for policy analysis," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 121:1–121:35, 2019.

[26] E. Bertino, A. Abu Jabal, S. Calo, C. Makaya, M. Touma, D. Verma, and C. Williams, "Provenance-based analytics services for access control policies," in *Proceedings of 2017 IEEE World Congress on Services (SERVICES)*. IEEE, 2017, pp. 94–101.

[27] A. Abu Jabal, M. Davari, E. Bertino, C. Makaya, S. Calo, D. Verma, and C. Williams, "ProFact: A provenance-based analytics framework for access control policies," *IEEE Transactions on Services Computing (TSC)*, 2019.

[28] A. Jain, S. P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier *et al.*, "Fireworks: A dynamic workflow system designed for high-throughput applications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5037–5059, 2015.

[29] M. Kunze, M. Weidlich, and M. Weske, "Querying process models by behavior inclusion," *Software & Systems Modeling*, vol. 14, no. 3, pp. 1105–1125, 2015.

[30] P. Fraternali *et al.*, "Graph search of software models using multidimensional scaling," in *CEUR WORKSHOP PROCEEDINGS*, 2015, pp. 163–170.

[31] E. C. Dragut, P. Baker, J. Xu, M. I. Sarfraz, E. Bertino, A. Madhkour, R. Agarwal, A. Mahmood, and S. Han, "Cris—computational research infrastructure for science," in *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*. IEEE, 2013, pp. 301–308.

[32] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[33] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[34] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Managing the evolution of dataflows with vistrails," in *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*. IEEE, 2006, pp. 71–71.

[35] "mygrid project." [Online]. Available: http://www.mygrid.org.uk/

[36] D. De, R. Carole, and G. R. Stevens, "The design and realisation of the myExperiment virtual research environment for social sharing of workflows," *Future Generation Comp. Syst.*, vol. 25, no. 5, pp. 561–567, 2009.

[37] V. Hu, D. Ferraiolo, K. Rick, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (abac) definition and considerations, 2017," [Online]. Available from: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-162.pdf.

[38] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao, "Understanding and capturing people's privacy policies in a mobile social networking application," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 401–412, 2009.

[39] R. A. Maxion and R. W. Reeder, "Improving user-interface dependability through mitigation of human error," *International Journal of Human-Computer Studies*, vol. 63, no. 1-2, pp. 25–50, 2005.

[40] Q. Ni, J. Lobo, S. Calo, P. Rohatgi, and E. Bertino, "Automating role-based provisioning by learning from examples," in *SACMAT*. ACM, 2009, pp. 75–84.

[41] A. Abu Jabal and E. Bertino, "Ql-simp: Query language for secure interoperable multi-granular provenance framework," in *Proceedings of the 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2016, pp. 131–138.

[42] S. B. Calo, D. C. Verma, and E. Bertino, "Distributed intelligence: Trends in the management of complex systems," in *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*. ACM, 2017, pp. 1–7.

[43] R. G. Michael and S. J. David, "Computers and intractability: a guide to the theory of np-completeness," *WH Free. Co., San Fr*, pp. 90–91, 1979.

[44] "OASIS eXtensible Access Control Markup Language (XACML) TC," [Online]. Available from: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[45] "AuthZForce," [Online]. Available from: https://authzforce.ow2.org/.

[46] "Balana," [Online]. Available from: https://github.com/wso2/balana.

[47] R. Kohavi and D. Sommerfield, "Feature subset selection using the wrapper method: Overfitting and dynamic search space topology," in *KDD*, 1995, pp. 192–197.

[48] "The U.S. army in multi-domain operations 2028," [Online]. Available from: https://www.tradoc.army.mil/Portals/14/Documents/MDO/TP525-3-1_30Nov2018.pdf.

[49] A. Abu Jabal, E. Bertino, J. Lobo, M. Law, A. Russo, S. Calo, and D. A. Verma, "Polisma - a framework for learning attribute-based access control policies," in *Proceedings of ESORICS 2020*, 2020.

[50] A. Chavan, S. Huang, A. Deshpande, A. Elmore, S. Madden, and A. Parameswaran, "Towards a unified query language for provenance and versioning," *arXiv preprint arXiv:1506.04815*, 2015.

[51] R. Jin, Y. Xiang, N. Ruan, and H. Wang, "Efficiently answering reachability queries on very large directed graphs," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 595–608.

[52] "Graphviz." [Online]. Available: http://www.graphviz.org/

[53] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM Sigmod Record*, vol. 34, no. 3, pp. 31–36, 2005.

[54] R. Hoekstra, P. Groth *et al.*, "Linkitup: link discovery for research data," in *AAAI Fall Symposium Series Technical Reports (FS-13-01)*, 2013, pp. 28–35.

[55] S. Bowers, T. Mcphillips, S. Riddle, M. K. Anand, and B. Ludäscher, "Kepler/ppod: Scientific workflow and provenance support for assembling the tree of life," in *International Provenance and Annotation Workshop*. Springer, 2008, pp. 70–77.

[56] "Prov toolbox." [Online]. Available: https://github.com/lucmoreau/ProvToolbox

[57] "Opm toolbox." [Online]. Available: https://github.com/lucmoreau/OpenProvenanceModel

[58] A. Gehani and D. Tariq, "Spade: support for provenance auditing in distributed environments," in *Proceedings of the 13th International Middleware Conference*. Springer-Verlag New York, Inc., 2012, pp. 101–120.

[59] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham, "A language for provenance access control," in *Proceedings of the first ACM conference on Data and application security and privacy*. ACM, 2011, pp. 133–144.

[60] D. A. Holland, U. J. Braun, D. Maclean, K.-K. Muniswamy-Reddy, and M. I. Seltzer, "Choosing a data model and query language for provenance," 2008.

[61] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener, "The lorel query language for semistructured data," *International journal on digital libraries*, vol. 1, no. 1, pp. 68–88, 1997.

[62] M. K. Anand, S. Bowers, and B. Ludäscher, "Techniques for efficiently querying scientific workflow provenance graphs." in *EDBT*, vol. 10, 2010, pp. 287–298.

[63] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 191–233, 2001.

[64] F. Rabitti, E. Bertino, W. Kim, and D. Woelk, "A model of authorization for next-generation database systems," *ACM Transactions on Database Systems (TODS)*, vol. 16, no. 1, pp. 88–131, 1991.

[65] "Authorization and permissions in sql server." [Online]. Available: https://msdn.microsoft.com/en-us/library/bb669084(v=vs.110).aspx

[66] M. Touma, E. Bertino, B. Rivera, D. Verma, and S. Calo, "Framework for behavioral analytics in anomaly identification," in *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VIII*, vol. 10190. International Society for Optics and Photonics, 2017, p. 101900H.

[67] E. Bertino, S. Calo, M. Touma, D. Verma, C. Williams, and B. Rivera, "A cognitive policy framework for next-generation distributed federated systems: concepts and research directions," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1876–1886.

[68] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI (EMP AI Workshop)*, vol. 3, no. 22. IBM, 2001, pp. 41–46.

[69] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[70] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *PAMI*, vol. 28, no. 10, pp. 1619–1630, 2006.

[71] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[72] N. Japkowicz, "The class imbalance problem: Significance and strategies," in *Proc. of the Int'l Conf. on Artificial Intelligence*, 2000.

[73] D. Ruta and B. Gabrys, "Classifier selection for majority voting," *Information fusion*, vol. 6, no. 1, pp. 63–81, 2005.

[74] R. P. Duin and D. M. Tax, "Experiments with classifier combining rules," in *MCS*. Springer, 2000, pp. 16–29.

[75] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[76] M. Mankai and L. Logrippo, "Access control policies: Modeling and validation," in *5th NOTERE Conference (Nouvelles Technologies de la Répartition)*, 2005, pp. 85–91.

[77] M. A. El Hadj, M. Ayache, Y. Benkaouz, A. Khoumsi, and M. Erradi, "Clustering-based approach for anomaly detection in xacml policies," in *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE), Madrid, Spain, July 24-26, 2017*, 2017, pp. 548–553.

[78] S. Pina Ros, M. Lischka, and F. Gómez Mármol, "Graph-based xacml evaluation," in *Proceedings of the 2012 17th ACM Symposium on Access Control Models and Technologies (SACMAT), Newark, NJ, USA, June 20-22, 2012*. ACM, 2012, pp. 83–92.

[79] A. Shaikh Riaz, K. Adi, L. Logrippo, and S. Mankovski, "Detecting incompleteness in access control policies using data classification schemes," in *Digital Information Management (ICDIM), 2010 Fifth International Conference on*. IEEE, 2010, pp. 417–422.

[80] J. Ma, D. Zhang, G. Xu, and Y. Yang, "Model checking based security policy verification and validation," in *Proceedings of the 2010 Second International Workshop on Intelligent Systems and Applications (ISA)*. IEEE, 2010, pp. 1–4.

[81] V. Kolovski, J. Hendler, and B. Parsia, "Analyzing web access control policies," in *Proceedings of the 16th International Conference on World Wide Web (WWW), Banff, Alberta, Canada, May 8-12, 2007*. ACM, 2007, pp. 677–686.

[82] D. Lin, P. Rao, E. Bertino, and J. Lobo, "An approach to evaluate policy similarity," in *Proceedings of the 2007 12th ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2007.

[83] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo, "Exam: a comprehensive environment for the analysis of access control policies," *International Journal of Information Security*, vol. 9, no. 4, pp. 253–273, 2010.

[84] R. Craven, J. Lobo, E. Lupu, J. Ma, A. Russo, M. Sloman, and A. Bandara, "A formal framework for policy analysis," *Imperial College London, Tech. Rep*, 2008.

[85] R. Craven, J. Lobo, J. Ma, A. Russo, E. Lupu, and A. Bandara, "Expressive policy analysis with enhanced system dynamicity," in *Proceedings of the 4th ACML Symposium on Information, Computer, and Communications Security (ASIACCS), Sydney, Australia, March 10-12, 2009*. ACM, 2009, pp. 239–250.

[86] P. Mazzoleni, E. Bertino, B. Crispo, and S. Sivasubramanian, "Xacml policy integration algorithms: not to be confused with xacml policy combination algorithms!" in *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT), Lake Tahoe, California, USA, June 7-9, 2006*. ACM, 2006, pp. 219–227.

[87] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino, "Xacml policy integration algorithms," *ACM Transactions on Information and System Security (TISSEC)*, vol. 11, no. 1, p. 4, 2008.

[88] D. J. Power, M. Slaymaker, and A. Simpson, "Conformance checking of dynamic access control policies," in *Proceedings of the 13th International Conference on Formal Engineering Methods (ICFEM), Durham, UK, October 26-28, 2011*. Springer, 2011, pp. 227–242.

[89] Y. Elrakaiby, T. Mouelhi, and Y. Le Traon, "Testing obligation policy enforcement using mutation analysis," in *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), Montreal, QC, Canada, April 17-21, 2012*. IEEE, 2012, pp. 673–680.

[90] J. Y. Halpern and V. Weissman, "Using first-order logic to reason about policies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 11, no. 4, p. 21, 2008.

[91] A. K. Bandara, E. C. Lupu, and A. Russo, "Using event calculus to formalise policy specification and analysis," in *Proceedings of 2003 IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, June 4-6, 2003*. IEEE, 2003, pp. 26–39.

[92] G. Boella, J. Hulstijn, and L. Van Der Torre, "Argumentation for access control," *AI* IA 2005: Advances in Artificial Intelligence*, pp. 86–97, 2005.

[93] G. Boella, J. Hulstijn, and L. van der Torre, "Argument games for interactive access control," in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence, Compiegne, France, 19-22 September 2005*. IEEE, 2005, pp. 751–754.

[94] L. Perrussel, S. Doutre, J.-M. Thévenin, and P. McBurney, "A persuasion dialog for gaining access to information," in *Proceedings of the 4th International Workshop on Argumentation in Multi-Agent Systems (ArgMAS), Honolulu, HI, USA, May 15, 2007*. Springer, 2007, pp. 63–79.

[95] G. Hughes and T. Bultan, "Automated verification of access control policies using a sat solver," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, no. 6, pp. 503–520, 2008.

[96] F. Alberti, A. Armando, and S. Ranise, "Efficient symbolic automated analysis of administrative attribute-based rbac-policies," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 165–175.

[97] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005, pp. 196–205.

[98] R. A. Shaikh, K. Adi, L. Logrippo, and S. Mankovski, "Inconsistency detection method for access control policies," in *Information Assurance and Security (IAS), 2010 Sixth International Conference on*. IEEE, 2010, pp. 204–209.

[99] R. A. Shaikh, K. Adi, and L. Logrippo, "A data classification method for inconsistency and incompleteness detection in access control policy sets," *International Journal of Information Security*, vol. 16, no. 1, pp. 91–113, 2017.

[100] M. Aqib and R. A. Shaikh, "Policy validation tool for access control policies," *Journal of Internet Technology*, 2018.

[101] L. Bauer, S. Garriss, and M. K. Reiter, "Detecting and resolving policy misconfigurations in access-control systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 2, 2011.

[102] W. Xu, X. Zhang, and G.-J. Ahn, "Towards system integrity protection with graph-based policy analysis," in *Proceedings of the Data and Applications Security XXIII, 23rd Annual (IFIP) (WG) 11.3 Working Conference, Montreal, Canada, July 12-15, 2009*. Springer, 2009, pp. 65–80.

[103] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "Grids-a graph based intrusion detection system for large networks," in *Proceedings of the 19th national information systems security conference*, vol. 1. Baltimore, 1996, pp. 361–370.

[104] S. Alves and M. Fernández, "A graph-based framework for the analysis of access control policies," *Theoretical Computer Science*, 2016.

[105] E. S. Al-Shaer and H. H. Hamed, "Modeling and management of firewall policies," *IEEE Transactions on Network and Service Management*, vol. 1, no. 1, pp. 2–10, 2004.

[106] S. Davy, B. Jennings, and J. Strassner, "Efficient policy conflict analysis for autonomic network management," in *Proceedings of the IEEE 5th Workshop on Engineering of Autonomic and Autonomous Systems (EASE)*. IEEE, 2008, pp. 16–24.

[107] J. W. Raymond, E. J. Gardiner, and P. Willett, "Rascal: Calculation of graph similarity using maximum common edge subgraphs," *The Computer Journal*, vol. 45, no. 6, pp. 631–644, 2002.

[108] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM computing surveys (CSUR)*, vol. 38, no. 2, p. 6, 2006.

[109] O. F. A. Specification, "Business process modeling notation specification," 2006.

[110] A.-W. Scheer, O. Thomas, and O. Adam, "Process modeling using event-driven process chains," *Process-aware information systems*, pp. 119–146, 2005.

[111] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte *et al.*, "Business process execution language for web services," 2003.

[112] W. M. Van Der Aalst and A. H. Ter Hofstede, "Yawl: yet another workflow language," *Information systems*, vol. 30, no. 4, pp. 245–275, 2005.

[113] P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera, "Designing data-intensive web applications," 2002.

[114] "Provone: A prov extension data model for scientific workflow provenance (2014)." [Online]. Available: http://purl.org/provone

[115] Q. Shao, P. Sun, and Y. Chen, "Wise: A workflow information search engine," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on.* Ieee, 2009, pp. 1491–1494.

[116] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome biology*, vol. 11, no. 8, p. R86, 2010.

[117] P. Mates, E. Santos, J. Freire, and C. T. Silva, "Crowdlabs: Social analysis and visualization for the sciences," in *International Conference on Scientific and Statistical Database Management*. Springer, 2011, pp. 555–564.

[118] T. Jin, J. Wang, and L. Wen, "Efficient retrieval of similar workflow models based on behavior," in *Asia-Pacific Web Conference*. Springer, 2012, pp. 677–684.

[119] Z. Yan, R. Dijkman, and P. Grefen, "Fast business process similarity search with feature-based similarity estimation," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2010, pp. 60–77.

[120] R. Bergmann and Y. Gil, "Similarity assessment and efficient retrieval of semantic workflows," *Information Systems*, vol. 40, pp. 115–127, 2014.

[121] N. Friesen and S. Rüping, "Workflow analysis using graph kernels," in *Proceedings of the ECML/PKDD Workshop on Third-Generation Data Mining: Towards Service-Oriented Knowledge Discovery (SoKD 2010), Barcelona, Spain*, 2010.

[122] E. Santos, L. Lins, J. P. Ahrens, J. Freire, and C. T. Silva, "A first study on clustering collections of workflow graphs," in *International Provenance and Annotation Workshop*. Springer, 2008, pp. 160–173.

[123] V. Silva, F. Chirigati, K. Maia, E. Ogasawara, D. Oliveira, V. Braganholo, L. Murta, and M. Mattoso, "Similarity-based workflow clustering," in *JCIS*, vol. 2, no. 1, 2011, pp. 23–35.

[124] J. Stoyanovich, B. Taskar, and S. Davidson, "Exploring repositories of scientific workflows," in *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*. ACM, 2010, p. 7.

[125] A. Goderis, P. Li, and C. Goble, "Workflow discovery: the problem, a case study from e-science and a graph-based solution," in *Proceedings of the International Conference on Web Services (ICWS)*. IEEE, 2006, pp. 312–319.

[126] X. Xiang and G. Madey, "Improving the reuse of scientificworkflows and their by-products," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 792–799.

[127] V. Cuevas-Vicenttín, B. Ludäscher, and P. Missier, "Provenance-based searching and ranking for scientific workflows," in *International Provenance and Annotation Workshop*. Springer, 2014, pp. 209–214.

[128] D. Shasha, J. T. Wang, and R. Giugno, "Algorithmics and applications of tree and graph searching," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2002, pp. 39–52.

[129] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 2004, pp. 335–346.

[130] S. Zhang, M. Hu, and J. Yang, "Treepi: A novel graph indexing method," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 966–975.

[131] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: towards verification-free query processing on graph databases," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 857–872.

[132] P. Zhao, J. X. Yu, and P. S. Yu, "Graph indexing: tree+ delta¡= graph," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 938–949.

[133] L. Zou, L. Chen, H. Zhang, Y. Lu, and Q. Lou, "Summarization graph indexing: beyond frequent structure-based approach," in *International Conference on Database Systems for Advanced Applications*. Springer, 2008, pp. 141–155.

[134] D. W. Williams, J. Huan, and W. Wang, "Graph database indexing using structured graph decomposition," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 976–985.

[135] H. He and A. K. Singh, "Closure-tree: An index structure for graph queries," in *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*. IEEE, 2006, pp. 38–38.

[136] Z. Xu and S. D. Stoller, "Mining attribute-based access control policies from logs," in *IFIP DBSec*. Springer, 2014, pp. 276–291.

[137] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD Record*, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[138] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *VLDB*, vol. 1215, 1994, pp. 487–499.

[139] R. V. Krejcie and D. W. Morgan, "Determining sample size for research activities," *Educational and psychological measurement*, vol. 30, no. 3, pp. 607–610, 1970.

[140] C. Cotrini, T. Weghorn, and D. Basin, "Mining abac rules from sparse logs," in *EuroS&P*. IEEE, 2018, pp. 31–46.

[141] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo, "Mining roles with semantic meanings," in *SACMAT*. ACM, 2008, pp. 21–30.

[142] Z. Xu and S. D. Stoller, "Algorithms for mining meaningful roles," in *SACMAT*. ACM, 2012, pp. 57–66.

[143] E. Medvet, A. Bartoli, B. Carminati, and E. Ferrari, "Evolutionary inference of attribute-based access control policies," in *EMO*. Springer, 2015, pp. 351–365.

[144] D. Mocanu, F. Turkmen, and A. Liotta, "Towards abac policy mining from logs with deep learning," in *IS*, 2015, pp. 124–128.

[145] L. Karimi and J. Joshi, "An unsupervised learning based approach for mining attribute based access control policies," in *Big Data*. IEEE, 2018, pp. 1427–1436.

[146] B. Kavšek and N. Lavrač, "Apriori-sd: Adapting association rule learning to subgroup discovery," *Applied Artificial Intelligence*, vol. 20, no. 7, pp. 543–583, 2006.

[147] M. W. Sanders and C. Yue, "Mining least privilege attribute based access control policies," in *Proceedings of the 2019 Annual Computer Security Applications Conference (ACSAC)*, 2019.

[148] C. Efstratiou, A. Friday, N. Davies, and K. Cheverst, "Utilising the event calculus for policy driven adaptation on mobile systems," in *3rd International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), 5-7 June 2002, Monterey, CA, USA. IEEE Computer Society.*

[149] A. Singla, E. Bertino, and D. Verma, "Preparing network intrusion detection deep learning models with minimal data using adversarial domain adaptation," in *Proceedings of the 2020 ACM Asia Conference on Computer and Communications Security, AsiaCCS 2020, Taipei, Taiwan, October 05-08, 2020. ACM.*

[150] ——, "Overcoming the lack of labeled data: Training intrusion detection models using transfer learning." in *Proceedings of the IEEE International Conference on Smart Computing, SMARTCOMP 2019, Washington, DC, USA, June 12-15, 2019. IEEE.*

[151] J. Zhao, S. Shetty, J. W. Pan, C. Kamhoua, and K. Kwiat, "Transfer learning for detecting unknown network attacks," *EURASIP Journal on Information Security*, vol. 2019, no. 1, p. 1, 2019.

[152] J. Zhao, S. Shetty, and J. W. Pan, "Feature-based transfer learning for network security," in *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 2017, pp. 17–22.

[153] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7167–7176.

[154] A. Abu Jabal and E. Bertino, "A comprehensive query language for provenance information," *International Journal of Cooperative Information Systems*, vol. 27, no. 03, p. 1850007, 2018.

[155] A. Abu Jabal, B. Elisa, J. Lobo, A. Russo, S. Calo, and D. Verma, "Flap - a federated learning framework for
attribute-based access control policies," 2020, manuscript submitted for publication.

[156] D. Lin, P. Rao, R. Ferrini, E. Bertino, and J. Lobo, "A similarity measure for comparing xacml policies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 9, pp. 1946–1959, 2013.

[157] D. Metzler, Y. Bernstein, W. B. Croft, A. Moffat, and J. Zobel, "Similarity measures for tracking information flow," in *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 2005, pp. 517–524.

[158] T. C. Hoad and J. Zobel, "Methods for identifying versioned and plagiarized documents," *Journal of the Association for Information Science and Technology*, vol. 54, no. 3, pp. 203–215, 2003.

[159] M. Ehrig, P. Haase, M. Hefke, and N. Stojanovic, "Similarity for ontologies-a comprehensive framework," *ECIS 2005 Proceedings*, p. 127, 2005.

[160] O. Biton, S. Cohen-Boulakia, and S. B. Davidson, "Zoom* userviews: Querying relevant provenance in workflow systems," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 1366–1369.

[161] D. Garijo, O. Corcho, and Y. Gil, "Detecting common scientific workflow fragments using templates and execution provenance," in *Proceedings of the seventh international conference on Knowledge capture*. ACM, 2013, pp. 33–40.

[162] P. Missier, J. Bryans, C. Gamble, V. Curcin, and R. Danger, *Provenance graph abstraction by node grouping*. Computing Science, Newcastle University, 2013.

[163] T. H. Cormen, *Introduction to Algorithms*. MIT press, 2009.

[164] M. Fujita, P. C. McGeer, and J. Yang, "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," *Formal methods in system design*, vol. 10, no. 2-3, pp. 149–169, 1997.

[165] "Number of tweets per day?" [Online]. Available: https://www.dsayce.com/social-media/tweets-day/

[166] "The top 20 valuable facebook statistics." [Online]. Available: https://zephoria.com/top-15-valuable-facebook-statistics/

[167] S. Wakamiya, R. Lee, and K. Sumiya, "Crowd-based urban characterization: extracting crowd behavioral patterns in urban areas from twitter," in *Proceedings of the 3rd ACM SIGSPATIAL international workshop on location-based social networks*. ACM, 2011, pp. 77–84.

[168] L. Mitchell, M. R. Frank, K. D. Harris, P. S. Dodds, and C. M. Danforth, "The geography of happiness: Connecting twitter sentiment and expression, demographics, and objective characteristics of place," *PloS one*, vol. 8, no. 5, p. e64417, 2013.

[169] B. J. Jansen, M. Zhang, K. Sobel, and A. Chowdury, "Twitter power: Tweets as electronic word of mouth," *Journal of the American society for information science and technology*, vol. 60, no. 11, pp. 2169–2188, 2009.

[170] Y. Liu, X. Huang, A. An, and X. Yu, "Arsa: a sentiment-aware model for predicting sales performance using blogs," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2007, pp. 607–614.

[171] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe, "Predicting elections with twitter: What 140 characters reveal about political sentiment." in *Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media (ICWSM)*, 2010, pp. 178–185.

[172] M. J. Paul and M. Dredze, "You are what you tweet: Analyzing twitter for public health." in *Proceedings of the Fifth International Conference on Weblogs and Social Media*. AAAI Press, 2011, pp. 265–272.

[173] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes twitter users: real-time event detection by social sensors," in *Proceedings of the 19th International Conference on World Wide Web (WWW)*. ACM, 2010, pp. 851–860.

[174] A. Kongthon, C. Haruechaiyasak, J. Pailai, and S. Kongyoung, "The role of twitter during a natural disaster: Case study of 2011 thai flood," in *Proceedings of Technology Management for Emerging Technologies (PICMET)*. IEEE, 2012, pp. 2227–2232.

[175] S. E. Vieweg, "Situational awareness in mass emergency: A behavioral and linguistic analysis of microblogged communications," Ph.D. dissertation, University of Colorado at Boulder, 2012.