NIST Special Publication 800-178

A Comparison of Attribute Based Access Control (ABAC) Standards for Data Service Applications

Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)

David Ferraiolo
Ramaswamy Chandramouli
Vincent Hu
Rick Kuhn

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-178

C O M P U T E R S E C U R I T Y

National Institute of Standards and Technology
U.S. Department of Commerce

NIST Special Publication 800-178

A Comparison of Attribute Based Access Control (ABAC) Standards for Data Service Applications

Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)

David Ferraiolo
Ramaswamy Chandramouli
Vincent Hu
Rick Kuhn
Computer Security Division
Information Technology Laboratory

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-178

October 2016

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

Authority

This publication has been developed by NIST in accordance with its statutory responsibilities under the Federal Information Security Modernization Act (FISMA) of 2014, 44 U.S.C. § 3551 *et seq.*, Public Law (P.L.) 113-283. NIST is responsible for developing information security standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

National Institute of Standards and Technology Special Publication 800-178
Natl. Inst. Stand. Technol. Spec. Publ. 800-178, 68 pages (October 2016)
CODEN: NSPUE2

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.800-178

Comments on this publication may be submitted to:

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: sp800-178@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems. The Special Publication 800-series reports on ITL's research, guidelines, and outreach efforts in information system security, and its collaborative activities with industry, government, and academic organizations.

Abstract

Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC) are very different attribute based access control (ABAC) standards with similar goals and objectives. An objective of both is to provide a standardized way for expressing and enforcing vastly diverse access control policies on various types of data services. However, the two standards differ with respect to the manner in which access control policies are specified and implemented. This document describes XACML and NGAC, and then compares them with respect to five criteria. The goal of this publication is to help ABAC users and vendors make informed decisions when addressing future data service policy enforcement requirements.

Keywords

access control; access control mechanism; access control model; access control policy; attribute based access control (ABAC); authorization; Extensible Access Control Markup Language (XACML); Next Generation Access Control (NGAC); privilege

Acknowledgements

The authors, David Ferraiolo, Ramaswamy Chandramouli, Vincent C. Hu, and Rick Kuhn of the National Institute of Standards and Technology (NIST), wish to thank their colleagues who reviewed drafts of this document, including the following: Karen Scarfone (Scarfone Cybersecurity), Wayne Jansen (Bayview Behavioral Consulting), Serban Gavrila (NIST), Indrakshi Ray (Colorado State University), Duminda Wijesekera (George Mason University), and Ram Krishnan (University of Texas at San Antonio).

The authors also gratefully acknowledge and appreciate the comments and contributions made by government agencies, private organizations, and individuals in providing direction and assistance in the development of this document.

Note to Readers

For purposes of transparency, one of the authors of this document, David Ferraiolo, is a member of the American National Standards Institute/International Committee for Information Technology (ANSI/INCITS) Next Generation Access Control (NGAC) working group. To mitigate the injection of bias, prior drafts have been circulated for review to multiple subject matter experts, as well as formally to the public at large. In addition, prior to publication, the final draft of this document was reviewed by multiple independent entities in compliance with the requirements of the NIST Editorial Review Board (ERB). Disposition and resolution of comments were considered by the authors at large.

Trademark Information

Executive Summary

Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC) are very different attribute based access control (ABAC) standards with similar goals and objectives. XACML, available since 2003, is an Extensible Markup Language (XML) based language standard designed to express security policies, as well as the access requests and responses needed for querying the policy system and reaching an authorization decision [1]. NGAC is a relations and architecture-based standard designed to express, manage, and enforce access control policies through configuration of its relations.

What are the similarities and differences between these two standards? What are their comparative advantages and disadvantages? These questions are particularly relevant because XACML and NGAC are different approaches to achieving a common access control goal—to allow applications with vastly different access policies to be expressed and enforced using the features of the same underlying mechanism in diverse ways. These are also important questions, given the prevalence of data services in computing. Data services include computational capabilities that allow the consumption, alteration, and management of data resources, and distribution of access rights to data resources. Data services can take on many forms, to include applications such as time and attendance reporting, payroll processing, and health benefits management, but also including system level utilities such as file management.

To answer these questions, this document first describes XACML and NGAC, then compares them with respect to five criteria. The first criterion is the relative degree to which the access control functionality of a data service can be separated from a proprietary operational environment. The other four criteria are derived from ABAC issues or considerations identified by NIST Special Publication (SP) 800-162 [2]: operational efficiency, attribute and policy management, scope and type of policy support, and support for administrative review and resource discovery.

Although NGAC is only now emerging as a national standard, it compares favorably in many respects with XACML and should be considered, along with XACML, by both users and vendors in addressing future data service policy enforcement requirements. Below is a summary of this comparison.

Separation of Access Control Functionality from Proprietary Operating Environments

Both XACML and NGAC achieve separation of access control functionality of data services from proprietary operating environments, but to different degrees. XACML's separation is partial. XACML does not envisage the design of a Policy Enforcement Point (PEP) that is data service agnostic. An XACML deployment consists of one or more data services, each with an operating environment-dependent PEP and operating environment-dependent operational routines and resource types that share a common Policy Decision Point (PDP) and access control information consisting of policies and attributes.

The degree of separation that can be achieved by NGAC is near complete. Although an NGAC deployment could include a PEP with an application programming interface (API) that recognizes operating environment-specific operations (e.g., send and forward operations for a

messaging system), it does not necessarily need to do so. NGAC includes a standard PEP with an API that supports a set of generic, operating environment-agnostic operations (read, write, create, and delete policy elements and relations). This API enables a common, centralized PEP to be implemented to serve the requests of multiple applications.

Operational Efficiency

An XACML request is a collection of attribute name, value pairs for the subject (user), action (operation), resource, and environment. XACML identifies relevant policies and rules for computing decisions through a search for Targets (conditions that match the attributes of the request). Because multiple Policies in a PolicySet and/or multiple Rules in a Policy may produce conflicting access control decisions, XACML resolves these differences by applying a policy combining algorithm from a set defined by the standard. The entire process includes collecting attributes, matching conditions, computing rules, and resolving conflicts involving at least two data stores. There are two phases of policy evaluation that need to be considered. The first and costliest is loading policy from disk to Policy Decision Point (PDP) main memory, and the second is request evaluation. In both phases, performance is directly related to the number of policies considered.

An NGAC request is composed of a process id, user id, operation, and a sequence of one or more operands mandated by the operation that affects either a resource or access control data. NGAC identifies relevant Policies and attributes by reference when computing a decision. NGAC computes decisions by applying a single combining algorithm over applicable Policies that do not conflict. Unlike XACML, NGAC does not need to load policy from disk into memory when evaluating a request. Instead, and as treated in NGAC reference implementation version 1.6 [11] all information necessary in computing an access decision can reside in memory. Memory is initially loaded when the PDP is initialized, and is updated when an administrative change occurs. The NGAC specification describes what constitutes a valid implementation, but does not provide implementation guidance, thereby leaving room for multiple competing approaches with different efficiencies. A measure of the operational efficiency is the complexity of algorithm used for arriving at a policy decision. In its reference implementation Version 1. 6 on GitHub [11], the NGAC computes a decision through an algorithm [30] that is linear. Furthermore, it is not linear in relation to the entire access control data set, but only to the portion relevant to a particular user.

Attribute and Policy Management

Proper enforcement of data resource policies is dependent on administrative policies. This is especially true in a federated or collaborative environment, where governance policies require different organizational entities to have different responsibilities for administering different aspects of policies and their dependent attributes.

XACML and NGAC differ dramatically in their ability to impose policy over the creation and modification of access control data (attributes and policies). NGAC manages attributes and policies through a standard set of administrative operations, applying the same enforcement interface and decision making function as it uses for accessing data resources. XACML does not recognize administrative operations, but instead manages policy content through a Policy

Administration Point (PAP) with an interface that is different from that for accessing data resources. XACML provides support for decentralized administration of some of its access policies. However, the approach is only a partial solution in that it is dependent on trusted and untrusted policies, where trusted policies are assumed valid, and their origin is established outside the delegation model. Furthermore, the XACML delegation model does not provide a means for imposing policy over modification of access policies, and offers no direct administrative method for imposing policy over the management of its attributes.

NGAC enables a systematic and policy-preserving approach to the creation of administrative roles and delegation of administrative capabilities, beginning with a single administrator and an empty set of access control data, and ending with users with data service, policy, and attribute management capabilities. NGAC provides users with administrative capabilities down to the granularity of a single configuration element, and it can deny users administrative capabilities down to the same granularity.

Scope and Type of Policy Support

Although resources may be protected under a wide variety of different access policies, these policies can be generally categorized as either discretionary or mandatory controls. Discretionary access control (DAC) is an administrative policy that permits system users to allow or disallow other users' access to resources that are placed under their control. Although XACML can theoretically provide users with administrative capabilities necessary to control and give away access rights to other users, the approach is complicated by the need to create and maintain additional metadata for each and every object/resource (e.g., Owner attribute). Conversely, NGAC has a flexible means of providing users with administrative capabilities to include those necessary for the establishment of DAC policies.

In contrast to DAC, mandatory access control (MAC) enables ordinary users' capabilities to execute operations on resources, but not administrative operations that may influence those capabilities. MAC policies unavoidably impose rules on users in performing operations on resources. MAC policies can be further characterized as controls that accommodate confinement properties to prevent indirect leakage of data to unauthorized users, and those that do not.

Expression of non-confinement MAC policies is perhaps XACML's strongest suit. XACML can specify rules and other conditions in terms of attribute values of varying types. There are undoubtedly certain policies that are expressible in terms of these rules that cannot be easily accommodated by NGAC. This is especially true when treating attribute values as integers. For example, to approve a purchase request may involve adding a person's credit limit to the person's account balance. Furthermore, XACML takes environmental attributes into consideration in expressing policy, and NGAC does not. However, there are some non-confinement MAC properties, including a variety of history-based policies, that NGAC can express but XACML cannot.

In contrast to NGAC, XACML does not recognize the capabilities of a process independent of the capabilities of its user. Without such features, XACML is ill-equipped to support confinement and as such is arguably incapable of enforcement of a wide variety of policies. These confinement-dependent policies include some instances of role-based access control

(RBAC), e.g., “only doctors can read the contents of medical records,” originator control (ORCON) and Privacy, e.g., “I know who can currently read my data or personal information”, or conflict of interest, e.g., “a user with knowledge of information within one dataset cannot read information in another dataset”. Through imposing process level controls in conjunction with event-response relations, NGAC has shown [3] support for these and other confinement-dependent MAC controls.

Administrative Review and Resource Discovery

A desired feature of access controls is review of capabilities of users and access control entries of objects [4] [18]. These features are often referred to as “before the fact audit” and resource discovery. “Before the fact audit” is one of RBAC’s most prominent features [5]. Being able to discover or see a newly accessible resource is an important feature of any access control system. NGAC supports efficient algorithms for both per-user and per-object review. Per-object review of access control entries is not as efficient as a pure access control list (ACL) mechanism, and per-user review of capabilities is not as efficient as that of RBAC. However, this is due to NGAC’s consideration of conducting review in a multi-policy environment. NGAC can efficiently support both per-object and per-user reviews of combined policies [30], where RBAC and ACL mechanisms can do only one type of review efficiently, and logical formula-based mechanisms such as XACML, although able to combine policies, cannot do either type of review efficiently [6].

Table of Contents

List of Appendices

List of Figures

List of Tables

1 Introduction

1.1 Purpose and Scope

The purpose of this document is to compare and contrast Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC) — two very different access control standards with similar goals and objectives. The document explains the basics of both standards and provides a comparative analysis based on attribute based access control (ABAC) considerations identified in NIST Special Publication (SP) 800-162, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* [2].

1.2 Audience

The intended audience for this document includes the following categories of individuals:

- Computer security researchers interested in access control and authorization frameworks;
- Security professionals, including security officers, security administrators, auditors, and others with responsibility for information technology (IT) security;
- Executives and technology officers involved in decisions about IT security products; and
- IT program managers concerned with security measures for computing environments.

This document, while technical in nature, provides background information and examples to help readers understand the topics that are covered. The material presumes that readers have a basic understanding of security and possess fundamental access control expertise.

1.3 Document Structure

The remainder of this document is organized into the following sections:

- Section 2 provides background information on the origins, makeup, and objectives of XACML and NGAC.
- Section 3 describes XACML's policy specification language and reference architecture for ABAC implementation.
- Section 4 describes NGAC's fundamentally different approach from XACML for representing requests, expressing and administering policies, representing and administering attributes, and computing and enforcing decisions.
- Section 5 provides an analysis of XACML and NGAC's similarities and differences based on five criteria.
- Section 6 contains the conclusion for the document.
- Appendix A— provides a list of acronyms used in the document.
- Appendix B— contains a list of references.
- Appendix C— provides a formal XACML policy specification for an abbreviated policy example in Section 3.

2 Background

Controlling and managing access to sensitive data has been an ongoing challenge for decades. Attribute-based access control (ABAC) represents the latest milestone in the evolution of logical access control methods. It provides an attribute-based approach to accommodate a wide breadth of access control policies and simplify access control management.

Most other access control approaches are based on the identity of a user requesting performance of an operation on a resource (e.g., read a file), with the capability enabled directly based on the user's identity, or indirectly through predefined attribute types such as roles or groups assigned to the user. Practitioners have noted that these forms of access control are often cumbersome to set up and manage, given their need to associate capabilities directly to users or their attributes, and the difficulty of doing so. Furthermore, the identity, group, and role qualifiers of a requesting user are often insufficient for expressing real-world access control policies. An alternative is to express policies in terms of attributes and to grant or deny user requests based on arbitrary attributes of users and resources, and optionally environmental attributes. This approach to access control is commonly referred to as attribute-based access control (ABAC) and is an inherent feature of both XACML and NGAC.

The access control logic of an ABAC engine that computes decisions based on assigned attributes of a requesting user, resource, and environment, and policies expressed in terms of those attributes is depicted in Figure 1.

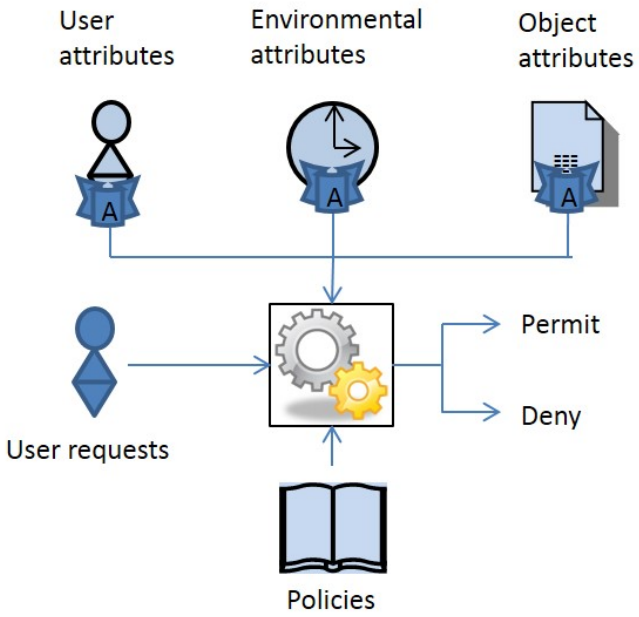

Figure 1: ABAC Overview

There are two techniques for specifying ABAC policies [6]. The most common approach is to define authorization policies by using logical formulas involving predicates expressed using logical operations (e.g., AND, OR, ≥, ≠) on attribute values. For example, can_access (u, a, o) →

Role(u) = "doctor" AND Ward (u) = Ward (o) AND (a = read OR a = write) specifies that any user with a Role of doctor can read or write any object where the Ward of the user is the same as the Ward of the object. XACML includes a policy specification language that falls into this category. Two other policy models that fall in this category are ABAC$_\alpha$ [7] and HGABAC (Hierarchical Group and Attribute Based Access Control) [8]. The second technique for expressing policy is by enumeration involving configurations of relations. NGAC and LaBAC (Label-Based Access Control) [6] fall under this category. For example, NGAC specifies polices in part by using combinations of association relations of the form (ua_i, ars_i, oa_i), with the meaning that the users in user attribute, ua_i have the access rights in access right set, ars_i on the objects in the object attribute, oa_i.

XACML and NGAC are ABAC standards for facilitating policy-preserving user executions of data service capabilities (data service operations on data service resources). In general, data services are a class of applications that provide users with capabilities to consume, manipulate, manage, and share data. Examples of data services include applications such as "time and attendance reporting", payroll processing, corporate calendaring, and health benefits management, all with a strong need for access control. XACML and NGAC provide a single generic access control facility for applications, resulting in a dramatic alleviation of many of the administrative, interoperability, and usability challenges otherwise faced by enterprises. This is achieved by removing the access control logic from the access control mechanism in the operating environment of each application. Instead the access control logic is implemented using a common set of access control modules that provide access decision functionality, which support these applications using a centralized policy and attribute repository.

Stated another way, from the perspective of access control, a data service can be conceptually viewed as an application with a presentation/logic layer and an operating environment layer delineated by their functionality and interfaces. The presentation layer provides users with an interface and methods for creating, displaying, and altering data. The presentation layer does not carry out operations that store, retrieve, or alter the state of stored data, or alter the access state to data (e.g., read, write/save, create and delete files, submit, approve, schedule), but instead issues requests to the operating environment layer to perform those operations. An operating environment implements operational routines to carry out access requests and provides access control to ensure that executions of processes involving operational routines are policy preserving.

Access control mechanisms comprise several components that work together to bring about policy-preserving data access. These components include access control data for expressing access control policies and representing attributes, and a set of functions for trapping access requests and for computing and enforcing access decisions over those requests. Most operating environments implement access control in different ways, each with a different scope of control (e.g., users, resources), and each with respect to different operation types (e.g., read, send, approve, select) and data types (e.g., files, messages, work items, records).

This heterogeneity introduces a number of administrative and policy enforcement challenges. Administrators are forced to contend with a multitude of security domains when managing access policies and attributes. Even if properly coordinated across operating environments,

global controls are hard to visualize and implement in a piecemeal fashion. Furthermore, because operating environments implement access control in different ways, it is difficult to exchange and share access control data across operating environments. XACML and NGAC seek to alleviate these challenges by creating a common and centralized way of expressing all access control data (policies and attributes) and computing decisions over the access requests of the presentation layer of applications (hereafter simply referred to as applications).

In 2014 NIST published SP 800-162, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations* [2], to serve two purposes. First, it provides federal agencies with an authoritative definition of ABAC and a description of its functional components. NIST SP 800-162 addresses ABAC as a mechanism comprising four layers of functional decomposition: Enforcement, Decision, Access Control Data, and Administration. Second, in light of the potential for numerous approaches to ABAC, NIST SP 800-162 highlights several considerations for selecting an ABAC system for deployment. These considerations pertain to operational efficiency, attribute and policy management, scope and type of policy support, and support for administrative review and resource discovery, among others. This report examines and compares XACML and NGAC based on these considerations. In addition, it compares the abilities of XACML and NGAC to separate the access control functionality necessary to support applications from proprietary operating environments. It is not intended to provide an implementation specification for either XACML or NGAC.

2.1 XACML

In 2003, with the emergence of Service Oriented Architecture (SOA), a new specification called XACML was published through the Organization for the Advancement of Structured Information Standards (OASIS). The specification presented the elements of what would later be considered by many to be ABAC. XACML employs three components in its authorization process:

- **XACML policy language**, for specifying access control requirements using rules, policies, and policysets, expressed in terms of subject (user), resource, action (operation), and environmental attributes and a set of algorithms for combining policies and rules.
- **XACML request/response protocol**, for querying a decision engine that evaluates subject access requests against policies and returns access decisions in response.
- **XACML reference architecture**, for deploying software modules to house policies and attributes, and computing and enforcing access control decisions based on policies and attributes.

Of the two ABAC standards, XACML is the oldest with the first version having been published in 2003. The current version is XACML 3.0 published in 2013. Through its evolution, it has received lots of scrutiny and acceptance by both research and vendor communities. Compared to the relatively new NGAC standard (parts originally published in 2013 and others still under development) with implementations still in their infancy, XACML has achieved much greater adoption with many open source and proprietary implementations covering all components of its standard. Although XACML products are too numerous to discuss individually, the Sun XACML open source is worth mentioning due to its pioneering implementation of all

components of XACML framework, thus establishing the standard's viability and making an impact on all subsequent XACML product development. Sun XACML is still the most widely used access decision engine both in enterprise applications and in research projects. Also, many of the other existing open source and proprietary XACML solutions are based on the functionality found in Sun XACML libraries.

2.2 NGAC

In 2003, NIST initiated a project in pursuit of a standardized ABAC mechanism referred to as the Policy Machine that allows changes to a fixed set of data elements and relations in the expression and enforcement of ABAC policies. The Policy Machine has evolved from a concept to a formal specification [9] and provisional patent[1] [10] to a reference implementation of NGAC and GitHub open source distribution [11]. The Policy Machine has served as a research component, in alignment with, and in support of a family of American National Standards Institute/International Committee for Information Technology Standards (ANSI/INCITS) standardization efforts under the title of "Next Generation Access Control" (NGAC) [12], [13]. In addition to the expression and enforcement of a wide variety of access control policies [14], [3], NGAC facilities can be used to effectuate security-critical portions of the program logic of arbitrary data services and enforce mission-tailored access control policies over data services [15]. NGAC standardization work has and continues to be conducted under three sub-projects:

- Project 2193–D: Next Generation Access Control – Implementation Requirements, Protocols and API Definitions
- Project 2194–D: Next Generation Access Control – Functional Architecture
- Project 2195–D: Next Generation Access Control – Generic Operations and Abstract Data Structures

An initial standard from this work was published in 2013 and is now available from the ANSI estandards store as INCITS 499 – NGAC Functional Architecture (NGAC–FA) [12]. However, based on experience with Project 2195-D and in reflection of Project 2193-D, work is underway to update this standard.

At this time the standard for Project 2195–D has been approved and is available from the ANSI estandards store as INCITS 526 – NGAC Generic Operations and Abstract Data Structures (NGAC-GOADS) [13].

Of the three NGAC projects, Project 2193–D is the least mature. A proposed approach and significant supporting material is expected by the end of the summer of 2016.

With the availability of the Policy Machine specification, the NGAC GitHub open-source distribution, and the emergence of NGAC (designed to be consistent with the Policy Machine specification) as a national standard, a number of efforts to develop product offerings are underway by commercial and academic institutions. At least one implementation of NGAC has

[1] U.S. Patent application 12/366,855 was withdrawn in lieu of ANSI/INCITS standard.

been commercially deployed to support a cloud solutions provider's in-house applications for clinical research in the life sciences. This access decision engine may represent the first live, in-production use of any Policy Machine/NGAC implementation, which is currently used to protect data in several clinical trials. In the near future, the remainder of the firm's products, which collectively manage a substantial portion of the world's clinical trial data, will be integrated with that access decision service. A significant portion of this Policy Machine/NGAC implementation has been contributed as open source [16].

2.3 Comparison of XACML and NGAC's Origins

Specifying and enforcing security policies (and in particular access control policies) in different components of an enterprise IT architecture made it expensive to maintain, modify, and demonstrate enterprise-wide compliance to regulations. XACML was developed to meet the need for a common language for expressing access control policies so as to manage the enforcement of all its elements in all components of information systems. The first version of the standard, XACML 1.0, was released by OASIS in 2003. The current version is XACML 3.0, which was ratified by OASIS in January 2013.

Since the XACML specification was lacking an access control model for the policies it contains, a policy administration model was needed. The XACML 3.0 Administration and Delegation Profile Version 1.0 [17] is a specification that addresses this need. Since administration in the context of policies makes sense only in a situation where multiple people possesses administrative rights to create policies, and since those always flow from a higher authority (e.g., centralized administrator), it is natural that the XACML delegation profile should have features for policy delegation and support for creating delegation chains.

The motivations for development of NGAC were different. While researchers, practitioners, and policymakers have specified a large variety of access control policies to address real-world security issues, only a relatively small subset of these policies could be enforced through off-the-shelf technology, and even a smaller subset could be enforced by any one mechanism. The goal was to devise a general-purpose access control framework that can express and enforce arbitrary, organization-specific, attribute-based access control policies through policy configuration settings. NGAC was devised to offer a new perspective on access control in terms of a fundamental and reusable set of data abstractions and functions that supports commonly known and implemented access control policies, as well as combinations of common policies, and policies for which no access control mechanism had existed.

Administration of policies as well as attributes (including delegation) were considered from the onset as an integral part of the framework. A proof of concept system that embodied the above principles is the Policy Machine that was first developed in 2003, with subsequent iterations improving the policy diversity and expressiveness of authorizations. Based on the need to standardize the access control model used in Policy Machine as well as the policy evaluation process, with a view to encourage large-scale adoption, NGAC was developed.

Though XACML and NGAC have different origins and motivations, the common theme is that authorizations in both are based on attributes associated with user, action, or resource. Thus both

XACML and NGAC are frameworks based on the ABAC model. However, in XACML, authorizations are expressed in logical formulas involving attribute values, while in NGAC they are enumerated using relations that contain attribute values as terms.

Both XACML and NGAC in their present state are intended for use in diverse application environments, such as file systems, and distributed applications with either web services or RESTful APIs.

3 XACML Specification

XACML defines a policy specification language and reference architecture for ABAC implementation. The standard encompasses syntax and semantics for representing requests, policies, attributes, and functions for computing decisions and enforcing policies in response to subject access requests to perform actions on resources.

For purposes of brevity and readability, this section presents a summary of the XACML specification that is intended to highlight XACML's salient features and should not be considered complete. In some instances, actual XACML details and terms are substituted with others to accommodate a simpler and more comprehensive presentation.

3.1 Attributes and Policies

An XACML access request consists of subject attributes (typically for the user who issued the request), resource attributes (the resource for which access is sought), action attributes (the operations to be performed on the resource), and environment attributes.

XACML attributes are specified as name-value pairs, where attribute values can be of different types (e.g., integer, string). An attribute name/ID denotes the property or characteristic associated with a subject, resource, action, or environment. For example, in a medical setting, the attribute name Role associated with a subject may have doctor, intern, and admissions nurse values, all of type string. Subject and resource instances are specified using a set of name-value pairs for their respective attributes. For example, the subject attributes used in a Medical Policy may include: Role = "doctor", Role = "consultant", Ward = "pediatrics", SubjectName = "smith"; an environmental attribute: Time = "12:11"; and resource attributes: Resource-id = "medical-records", WardLocation = "pediatrics", Patient = "johnson". Although XACML does not require any convention for naming attributes, the prefixes Subject, Resource, and Env are used for naming the subject, resource, and environment attributes, respectively, to enhance readability.

Subject and resource attributes are stored in their respective repositories and are retrieved through the Policy Information Point (PIP) at the time of an access request and prior to the computation of the decision. XACML formally defines an action as a component of a request with attribute values that specify operations such as read, write, submit, and approve.

Environmental attributes, which depend on the availability of system sensors that can detect and report values, are somewhat different from subject and resource attributes, which are administratively created. Environmental attributes are not properties of the subject or resources, but are measurable characteristics that pertain to the operational or situational context in which access requests occur. These environmental characteristics are subject and resource independent, and may include the current time, day of the week, or threat level.

This document uses a functional notation for reporting on attribute values with the format A(), where the parameter may be a subject, resource, action, or environment. For example, A(*e*), where *e* is the environment, may equal 09:00 (time) and low (threat level), and A(*s*), where *s* is a

subject, may equal smith (name) and doctor (role). A tuple notation to describe multiple attributes possessed by a subject, resource, or environment is used. For example, A(*s*1) = <smith, doctor>, where the first attribute corresponds to the name and the second one to the role possessed by subject *s*1.

As shown by Figure 2, XACML access policies are structured as PolicySets that are composed of Policies and optionally other PolicySets, and Policies that are composed of Rules. Policies and PolicySets are stored in a Policy Retrieval Point (PRP). Because not all Rules, Policies, or PolicySets are relevant to a given request, XACML includes the notion of a Target. A Target defines a simple Boolean condition that, if satisfied by the attributes (evaluates to True), establishes the need for subsequent evaluation by a Policy Decision Point (PDP). If no Target matches the request, the decision computed by the PDP is NotApplicable.

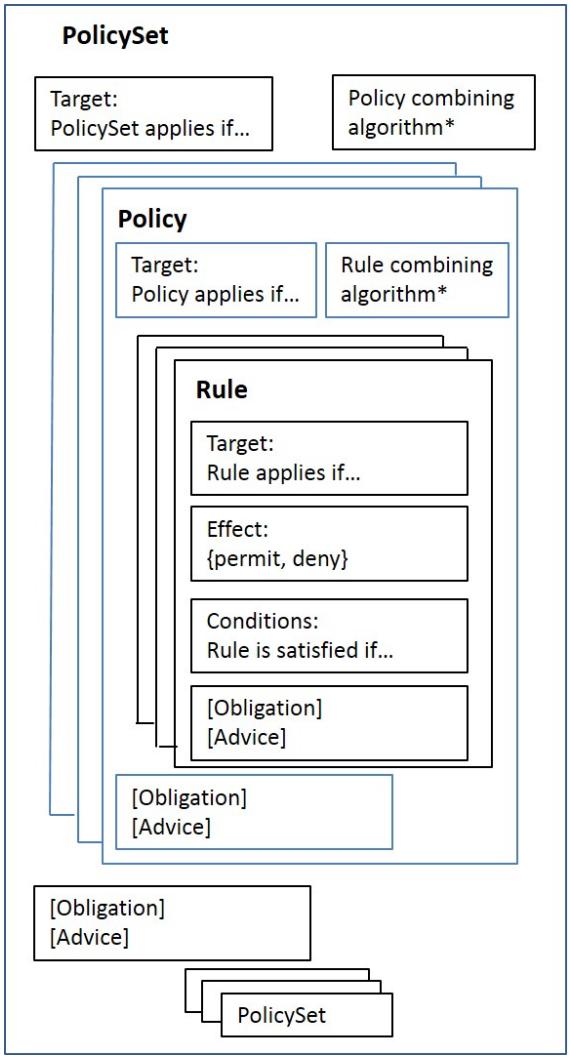

Figure 2: XACML Policy Constructs

In addition to a Target, a rule includes a series of Boolean conditions that if evaluated True have an effect of either Permit or Deny. If the target condition evaluates to True for a Rule and the

Rule's condition fails to evaluate for any reason, the effect of the Rule is Indeterminate. In comparison to the (matching) condition of a Target, the conditions of a Rule or Policy are typically more complex and may include functions involving logical operators (e.g., "greater-than-equal", "less-than", "string-equal") for the comparison of attribute values. Conditions can be used to express access control relations (e.g., a doctor can only view a medical record of a patient assigned to the doctor's ward) or computations on attribute values (e.g., sum(x, y) less-than-equal:250). To promote interoperability, all attributes have well-known datatypes and all functions that use those attributes are appropriately typed. While these standard data types and functions enable flexible policy expression, XACML further specifies extensions for defining additional data types and functions.

3.2 Combining Algorithms

Because a Policy may contain multiple Rules, and a PolicySet may contain multiple Policies or PolicySets, each Rule, Policy, or PolicySet may evaluate to different decisions (Permit, Deny, NotApplicable, or Indeterminate). XACML provides a way of reconciling these individual decisions. This reconciliation is achieved through a collection of combining algorithms. Each algorithm represents a different way of combining multiple local decisions into a single global decision. There are several standard combining algorithms defined, including the following:

- Deny-overrides: if any decision evaluates to Deny, or no decision evaluates to Permit, then the result is Deny. If all decisions evaluate to Permit, the result is Permit.
- Permit-overrides: if any decision evaluates to Permit, then the result is Permit, otherwise the result is Deny.
- First-applicable: the result is the result of the first decision (either Permit, Deny, or Indeterminate) when evaluated in their listed order.
- Only-one-applicable: if only one decision applies, then the result is the result of the decision, and if more than one decision applies, then the result is Indeterminate.

Combining algorithms are applied to Rules in a Policy and Policies within a PolicySet in arriving at an ultimate decision of the PDP. Combining algorithms can be used to build up increasingly complex policies. For example, given that a subject request is Permitted (by the PDP) only if the aggregate (ultimate) decision is Permit, the effect of the Permit-overrides combining algorithm is an "OR" operation on Permit (any decision can evaluate to Permit), and the effect of a Deny-overrides is an "AND" operation on Permit (all decisions must evaluate to Permit).

In addition to the standard set of combining algorithms, XACML includes a standard extension mechanism that can be used to define additional algorithms.

3.3 Obligation and Advice Expressions

XACML includes the concepts of obligation and advice expressions. An obligation optionally specified in a Rule, Policy, or PolicySet is a directive from the PDP to the Policy Enforcement Point (PEP) on what must be carried out before or after an access request is approved or denied. Advice is similar to an obligation, except that advice may be ignored by the PEP.

A few examples include:

- If Alice is denied access to document X: inform her manager via email that Alice tried to access document X.
- If a user is denied access to a file: inform the user why the access was denied.
- If a user is approved to view document X: watermark the document "DRAFT" before delivery.

A common use of an obligation applied after an access request is approved is for auditing and logging user access events.

It should be noted that the functionality to accommodate the directives of an obligation or advice is outside of the scope of XACML and must be implemented and executed by an application-specific PEP.

3.4 Example Policies

Consider the following two example XACML Policy specifications. For purposes of maintaining the same semantics as XACML, the same element names are used, but Policies and Rules are specified in pseudocode (instead of exact XACML syntax) for purposes of enhanced readability. A more formal XACML treatment of the first Policy (Policy 1) is included in Appendix C.

Policy 1 applies to "All read or write accesses to medical records by a doctor or intern" (the Target of the Policy) and includes three rules. As such, the Policy is considered "applicable" whenever a subject with a role of "doctor" or "intern" issues a request to read or write the "medical-records" resource. The Rules do not refine the Target, but describe the conditions under which read or write requests from doctors or interns to medical records can be allowed. Rule 1 will deny any access request (read or write) if the ward in which the doctor or intern is assigned is not the same ward where the patient is located. Rule 2 explicitly denies "write" access requests to interns under all conditions. Rule 3 permits read or write access to medical-records for "doctor", regardless of Rule 1, if an additional condition is met. This additional condition pertains to patients in critical status. Since the intent of the Policy is to allow access under these critical situations, a policy combining algorithm of "permit-overrides" is used, while still denying access if only the conditions stated in Rule 1 or Rule 2 apply.

```
<Policy PolicyId = "Policy 1" rule-combining-algorithm="permit-overrides">
            // Doctor Access to Medical Records //
     <Target>
        /* :Attribute-Category    :Attribute ID     :Attribute Value */
                 :access-subject      :Role                 :doctor
                 :access-subject      :Role                 :intern
                 :resource            :Resource-id          :medical-records
                 :action              :Action-id            :read
                 :action              :Action-id            :write
      </Target>

      <Rule RuleId = "Rule 1" Effect="Deny">
            <Condition>
          Function: string-not-equal
```

```
                /* :Attribute-Category   :Attribute ID   */
                     :access-subject       :WardAssignment
                     :resource             :WardLocation
            </Condition>
        </Rule>

        <Rule RuleId = "Rule 2" Effect="Deny">
            <Condition>
          Function: and
            Function: string-equal
               /* :Attribute-Category    :Attribute ID    :Attribute Value */
                     :access-subject       :Role            :intern
           Function: string-equal
              /* :Attribute-Category     : Attribute ID   :Attribute Value */
                     :action               :Action-id       :write
            </Condition>
         </Rule>

        <Rule RuleId = "Rule 3" Effect="Permit">
            <Condition>
          Function: and
            Function: string-equal
              /* :Attribute-Category    :Attribute ID     :Attribute Value */
                     :access-subject      :Role             :doctor
           Function: string-equal
              /* :Attribute-Category    :Attribute ID     :Attribute Value */
                     :resource            :PatientStatus    :critical
            </Condition>
        </Rule>
  </Policy>
```

Together policies (PolicySets and Policies) and attribute assignments define the authorization state. Table 1 defines the authorization state for Policy 1 by specifying attribute names and values.

Table 1. Attribute Names and Values and the Authorization State for Policy 1

Subject attribute names and their domains: Role = {doctor, intern} WardAssignment = {ward1, ward2}
Resource attribute names and their domains: Resource-id = {medical-records} WardLocation = {ward1, ward2} PatientStatus = {critical}
Action attribute names and their domains: Action-id = {read (r), write (w)}
Attribute value assignments when there are two subjects (s3, s4) and three resources (r5, r6, r7): A(s3) = <doctor, ward2>, A(s4) = <intern, ward1>, A(r5) = <medical-records, ward2>, A(r6) = <medical-records, ward1>, and A(r7) = <critical>.

Authorization state:
(s3, r, r5), (s3, w, r5), (s3, r, r7), (s3, w, r7), (s4, r, r6)

Policy 2 applies to “IRS-agents and auditor access to tax-returns” (Target of the Policy) and has two Rules. This Policy is an “applicable policy” whenever users with role “IRS-agent or auditor” access the resource “tax-returns” with a write request. The Rules do not refine the Target, but state the conditions under which write requests from IRS-agents or auditors to tax-returns records can be allowed. Rule 1 will permit an applicable access request if the access time (an environmental variable) is between 8 AM and 6 PM. Rule 2 will deny the request even if the condition in Rule 1 applies through an additional condition; the IRS-agent or auditor is attempting to write to his or her own tax return. Since the intent of the Policy is to disallow IRS employees from altering their own tax returns, a policy combining algorithm of “deny-overrides” is used, while still allowing access if the conditions stated in Rule 2 do not apply.

```
<Policy PolicyId = “Policy 2” rule-combining-algorithm=”deny-overrides”>
            // IRS Agent and Auditor Access to Tax Returns //
     <Target>
        /* :Attribute-Category   : Attribute ID   : Attribute Value */
                 :access-subject      :Role                 :IRS-agent
                 :access-subject      :Role                 :auditor
                 :resource            :Resource-id          :tax-returns
                 :action              :Action-id            :write
      </Target>

      <Rule RuleId = “Rule 1” Effect=”Permit”>
             <Condition>
           Function: and
           /* :Attribute-Category   : Attribute ID   : Attribute Value
                          :environment        : Time               : ≥ 08:00
                          :environment        : Time               : ≤ 18:00
             </Condition>
       </Rule>
       <Rule RuleId = “Rule 2” Effect=”Deny”>
             <Condition>
               Function: string-equal
                 /* :Attribute-Category   :Attribute ID
                          : access-subject     :SubjectName
                          : resource           :Resource-owner
             </Condition>
        </Rule>
  </Policy>
```

3.5 XACML Access Request

XACML specifies a format for conveying an access request and a related decision response. The format is referred to as XACML Context. The request and response formats represent a standard interface between a PDP, with standard behavior, and a standard PEP that issues requests and deals with the response. If the PEP does not generate requests in XACML Context, a context handler is required between the PEP and PDP. The handler converts application environment-specific requests (coming from PEP) to XACML Context requests before submitting them to PDP, and also converts XACML Context responses received from PDP to application environment-specific responses before forwarding them to PEP.

A Request Context consists of one or more attributes associated with elements: subject, resource, action, and environment. For example, if the IRS Agent Smith is making a request to write Brown's Tax Return at 9:30 a.m., the XACML access request will carry the values "smith" and "IRS-agent" for the Subject-id and Role attributes, values "tax-returns" and "brown" for the resource's Resource-id and Resource-owner attributes, value "write" for action's Action-id, and value "09:30." for environment's time attribute. XACML code for this access request is as follows.

```
<Request>
   <Subject>
       <Attribute AttributeId=”access-subject;Subject-id”>
          <AttributeValue>smith</AttributeValue>
       <Attribute AttributeId=”access-subject;Role”>
          <AttributeValue>IRS-agent</AttributeValue>
    </Subject>
    <Resource>
       <Attribute AttributeId=”resource;Resource-id”>
          <AttributeValue>tax-returns</AttributeValue>
       <Attribute AttributeId=”resource:Resource-owner”>
          <AttributeValue>brown</AttributeValue>
    </Resource>
    <Action>
       <Attribute AttributeId=”action;Action-id”>
          <AttributeValue>write</AttributeValue>
    </Action>
    <Environment>
       <Attribute AttributeId=”environment;time”>
          <AttributeValue>09:30</AttributeValue>
    </Environment>
</Request>
```

Policy 2 (Section 3.4) is applicable to this request, since its target includes any subject with Role=IRS-agent or auditor accessing the resource tax-returns in "write" mode. This policy contains a rule with a condition that specifies the time interval for allowed access as well as another rule with a condition that will deny access if the resource owner is the same as the access subject (an IRS-agent or auditor accessing his/her own tax-returns).

A Response Context contains one or more results obtained from the PDP's evaluation of the Access Request against the policy. Among these results is a Decision, Status, and optionally, Obligation or Advice. As discussed above the Decision can be one of the following: Permit, Deny, Not Applicable, or Indeterminate (in the event of an error). The Status returns optional information pertaining to an error. The Response may optionally include one or more Obligations or Advice if included in an applicable Policy or PolicySet.

The response, encoded in XML, when the above request is evaluated against Policy 2 is as follows:

```
<Response>
    <Result>
        <Decision>Permit</Decision>
        <Status>
             <StatusCode Value=”status:ok”/>
```

```
        </Status>
    </Result>
</Response>
```

3.6 Delegation

The XACML Policies discussed thus far have pertained to Access Policies that are created and may be modified by a single authority. This single authority (e.g., a centralized administrator) is trusted and hence these access policies carry no designation of an "Issuer" and are considered "trusted". Thus, based on the semantics of trust, the type or class to which these policies belong can then be called "Trusted Access Policies". But in the absence of any other type of policy, every policy in the XACML policy repository is simply called an "Access Policy". The advantage of a trusted policy is that it can be directly used by the PDP in rendering a decision.

In recognition of the need to create policies through delegation from a centralized administrator or single authority to a set of subordinate administrators (or delegated authorities), XACML has introduced a new type of policy called *administrative policies*. A feature enabled by this new type of policy is the ability to designate multiple authorities either for: (a) creating access policies or (b) designating further subordinate authorities through creation of one or more administrative policies. This designated authority is called a *delegate* and the domain over which the delegate has authority is called a *situation*. Collectively these capabilities result in the specification of a set of delegated policies and thus delegation chains – consisting of one or more administrative policies and terminating with an access policy.

The two new concepts of delegate and situation introduced by administrative policies are encapsulated within its *Target*. Thus the Target in an administrative policy is different in content and semantics from that found in an access policy. The delegate is an attribute category of the same type as a subject, representing the entity(s) that has (have) been given the authority to create either an access policy (thus completing the delegation chain) or an administrative policy (thus adding to the delegation chain). A situation provides the scope for that authority by designating a privilege domain. This privilege domain is expressed in terms of a combination of subject, resource, and action attributes. If the delegate creates a Policy granting access rights for resources covered by that privilege domain, then the type designation for such a Policy is *Untrusted Access Policy*. An Untrusted Access Policy can be distinguished from an *Access Policy* (found in policy repositories without delegated policies) in that the former should always have an Issuer. On the other hand, if the delegate decides to exercise the right of further delegation (instead of exercising the right to create access policies) within the scope of that privilege domain, the resulting administrative policy he/she creates will be of a type called *Untrusted Administrative Policy*.

Thus, in an XACML Policy repository that supports delegation, you can have the following four sub-types of Policies:

- (Trusted) Access Policies,
- Untrusted Access Policies,
- Trusted Administrative Policies, and
- Untrusted Administrative Polices.

The root, or the starting point for creating a set of delegated Policies (and hence a delegation chain), is a sub-type of administrative policy called a *Trusted Administrative Policy*. Trusted Administrative Policies are created under the same authority used to create (trusted) Access Policies. Thus a Trusted Administrative Policy does not have an Issuer tag since it is always created by a trusted centralized administrator.

Recall that the purpose of administrative policies is the ability to delegate the rights to create access policies to multiple authorities either directly or indirectly through a series of further delegations. Thus a Trusted Administrative Policy gives the delegate the authority to create Untrusted Administrative Policies or Untrusted Access Policies. Hence a delegation chain has to naturally start from a Trusted Administrative Policy and end with an Untrusted Access Policy. Consequently, a delegation chain can optionally include one or more intermediate Untrusted Administrative Policies. The situation for a newly created Untrusted Administrative Policy or Untrusted Access Policy is a subset (the same or narrower in scope) of that specified in the Trusted Administrative Policy. In addition, an Untrusted Administrative Policy or Untrusted Access Policy includes an Issuer tag with a value that is the same as that of the delegate in the administrative policy under which it was created. Both of these policies have at least one rule with a PERMIT or DENY effect.

3.6.1 Delegation Chain – An Example

An example of a delegation chain (going from bottom to top) is shown in Figure 3. As expected, the starting node of this delegation chain is a type of administrative policy called a Trusted Administrative Policy. Based on the description, this should have a delegate (the subordinate administrator) and a situation (the privilege domain over which the subordinate administrator can grant access (or in turn delegate further)) in its target. In our example, the Trusted Administrative Policy designates “Smith” as the subordinate administrator and “Situation1” as the privilege domain over which Smith can grant access rights to resources or do further delegation.

Recall that an Untrusted Administrative Policy should also have a delegate (his designated subordinate) and associated privilege domain for the subordinate. In addition, an Untrusted Administrative Policy should have an Issuer tag as well. The value of this Issuer tag must match with that of a Delegate tag in some administrative policy. Then only the authority under which this Untrusted Administrative Policy was created can be established. Figure 3 shows an Untrusted Administrative Policy issued by Smith (under the authority of the Trusted Administrative Policy discussed above) designating Jones as the delegate (subordinate administrator) with the associated privilege domain being “Situation2”. The scope of this privilege domain “Situation2” must be a subset of the privilege domain “Situation1” that was designated in its authority-granting Trusted Administrative Policy discussed above.

Just like the subordinate administrator Smith, his delegate Jones can exercise the right of further delegation or grant access rights to resources specified within her privilege domain. If Jones exercises the right of granting access rights to resources within her privilege domain (unlike Smith), the resulting access policy that she will create will be an Untrusted Access Policy. In Figure 3, there is an Untrusted Access Policy issued by Jones (under the authority granted to her by Smith) with the scope of access rights covered designated as “Situation3”. This scope must be a subset of the privilege domain (Situation2) that was delegated to her by Smith.

The number of delegated policies (forming the delegation chain) in an XACML policy repository is comparatively small. The majority of the policies are (trusted) access policies created by a single authority and this is shown in Figure 3.

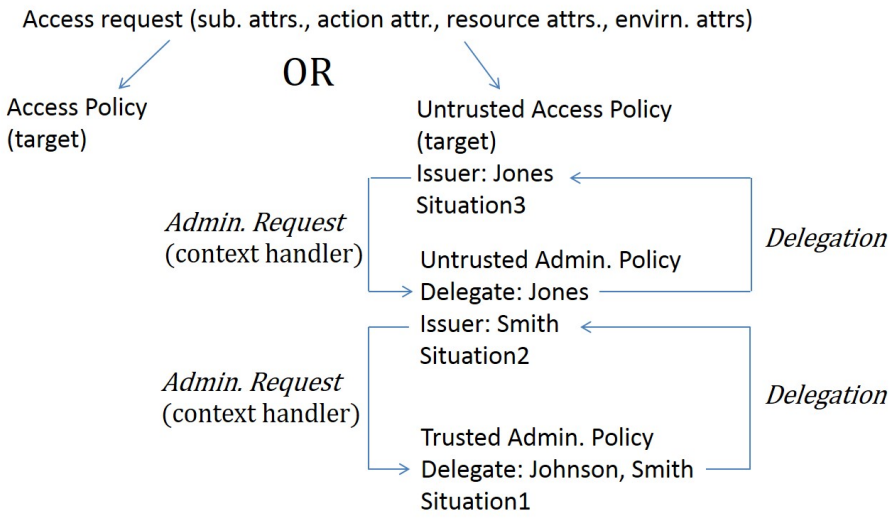

Figure 3: An Example of a Delegation Chain

3.6.2 Access Request Processing in Delegation Chains

In the absence of any delegation chain, all access policies in the system are (trusted) Access Policies. In these situations, if the PDP is able to match the attributes in the request with the target of an Access Policy, then that Access Policy is treated as an applicable policy and automatically included as a participating policy in the relevant policy-combining algorithm (since the policy is trusted). In the presence of one or more delegation chains, the PDP may discover that the matching target (for a request) is found in an Untrusted Access Policy. Since it is designated as an Untrusted Access Policy, it can be considered for inclusion in the policy-combining algorithm by the PDP only if the authority of the policy issuer is first verified (recall that an Untrusted Access Policy always has an Issuer tag). Authority is verified by finding a delegation chain that leads to a Trusted Administrative Policy. This chain may involve zero or more Untrusted Administrative Policies in the path to a Trusted Administrative Policy. When policies are considered as nodes of a graph, the process of finding a delegation chain translates to finding/constructing a path in that graph from a node representing an Untrusted Access Policy to a node representing a Trusted Administrative Policy. To construct each edge of the graph, the PDP (using the XACML context handler) formulates an Administrative Request as shown in Figure 3.

An Administrative Request has the same structure as an Access Request except that in addition to attribute categories – access-subject, resource, and action – it also uses two additional attribute categories, delegate and decision-info. If a policy Px happens to be one of the applicable (matched) Untrusted Access Policies, the Administrative Request is generated using policy Px to construct an edge to policy Py using the following:

- Convert all attributes (and attribute values) used in the original Access Request to attributes of category "delegated";
- Include the value under the *Issuer* tag of Px as the value for the subject-id attribute of the *delegate* attribute category; and
- Include the effect of evaluating policy Px as attribute value (PERMIT, DENY, etc.) for the Decision attribute of the *decision-info* attribute category.

The Administrative Request constructed using the above attributes is evaluated against the target for policy Py. If the result of the evaluation is "PERMIT", an edge is constructed between policies Px and Py. The overall logic involved is to verify the authority for issuance of policy Px. For this there should exist a policy with its "delegate" set to the policy issuer of Px. If that policy is Py, then it means policy Px has been issued under the authority found in policy Py. The edge construction then proceeds from policy Py until an edge to a Trusted Administrative Policy is found.

The process of selecting applicable policies for inclusion in the combining algorithm in delegation chains is illustrated in Figure 4. Assume that for an Access Request, the matching targets are found in three Untrusted Access Policies P31, P32, and P33. Each of these policies can become an applicable policy (and hence can be included in an associated combining algorithm) if a path can be constructed (through an edge or a series of edges) from each of them to a Trusted Administrative policy (thus verifying the authority under which each was issued). In Figure 4, such paths can be constructed/established from policies P31 and P32. From Policy P31, this path goes through an Untrusted Administrative Policy P21 to a Trusted Administrative Policy P11. From Policy P32, the path goes through an Untrusted Administrative Policy P22 to a Trusted Administrative Policy P12. However, no such path exists from the third Untrusted Access Policy P33. Hence, only Policies P31 and P32 will be used in the combining algorithm for evaluating the final access decision, and Policy P33 will be discarded since its authority could not be verified.

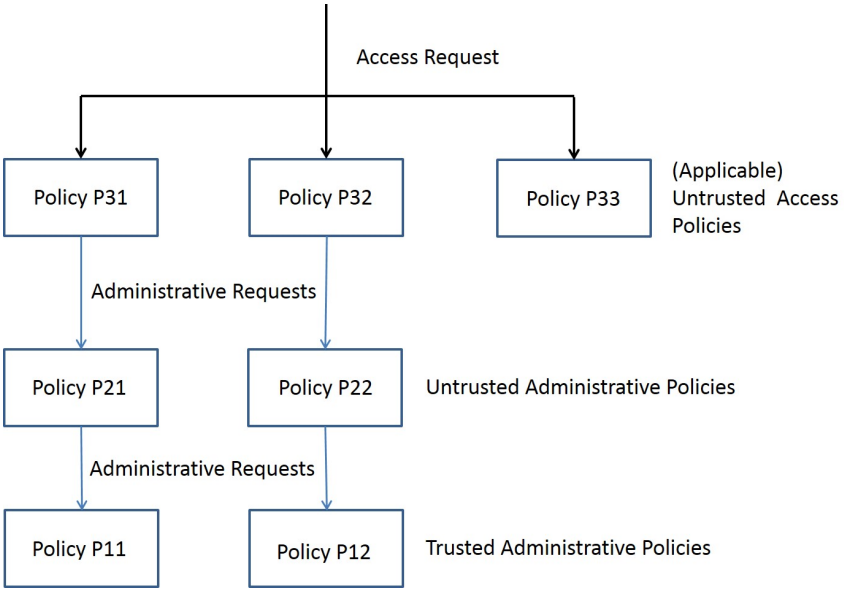

Figure 4: Utilizing Delegation Chains for Policy Evaluation

Below is a more concrete example that illustrates the use of delegation chains to select applicable policies that are used in combining algorithms for arriving at final access decisions. The example gives a Policy Set that consists of four policies:

- Policy P1: A Trusted Administrative Policy that gives John (the delegate) the authority to create policies for a situation involving reading of medical records to any user who has the role of Doctor.
- Policy P2: An Untrusted Administration Policy that is issued by John, under the authority of P1, to give Jessica (the delegate) the authority to create policies for a situation involving reading of medical records to any user who has the role of Doctor. Because of the matching of delegate of P1 to policy issuer of P2 and the fact that the situations in both policies P1 and P2 are the same, it is obvious that the authority to issue policy P2 has come from policy P1. Thus P1 and P2 form a delegation chain.
- Policy P3: An Untrusted Access Policy that is issued by Jeff to give Carol the capability to read medical records.
- Policy P4: An Untrusted Access Policy that is issued by Jessica to give Carol the ability to read medical records. Because of the matching of delegate of P2 to policy issuer of P4 and the fact that the situations in both policies P2 and P4 are the same, it is obvious that the authority to issue policy P4 has come from policy P2. Thus P2 and P4 form a delegation chain.

The four policies described above are given in the form of pseudocode below:

```
<Policy Set>
   <Policy P1> /* Trusted Administrative Policy */
       <Target>  /*:Attribute-Category  :Attribute ID  :Attribute Value */
                      :access-subject    :role           :doctor
                      :resource          :resource-id    :medical-records
                      :action            :action-id      :read
                      :delegate          :subject-id     :john
       </Target>
       <Rule R1>
           Effect:  PERMIT
       </Rule R1>
 </Policy P1>

 <Policy P2> /* Untrusted Administrative Policy */
       <Policy Issuer> john </Policy Issuer>
       <Target> /*:Attribute-Category   :Attribute ID  :Attribute Value */
                     :access-subject     :role          :doctor
                     :resource           :resource-id   :medical-records
                     :action             :action-id     :read
                     :delegate           :subject-id    :jessica
       </Target>
       <Rule R2>
           Effect:  PERMIT
       </Rule R2>
  </Policy P2>

  <Policy P3> /* Untrusted Access Policy */
       <Policy Issuer> Jeff </Policy Issuer>
```

```
      <Target>  /*:Attribute-Category   :Attribute ID   :Attribute Value */
                     :access-subject     :subject-id     :carol
                     :resource           :resource-id    :medical-records
                     :action             :action-id      :read
      </Target>
      <Rule R3>
          Effect:  PERMIT
      </Rule R3>
  </Policy P3>

  <Policy P4> /*  Untrusted Access Policy */
      <Policy Issuer> Jessica </Policy Issuer>
      <Target>  /*:Attribute-Category   :Attribute ID   :Attribute Value */
                     :access-subject     :subject-id     :carol
                     :resource           :resource-id    :medical-records
                     :action             :action-id      :read
      </Target>
      <Rule R4>
          Effect:  PERMIT
      </Rule R4>
   </Policy P4>
<Policy Set>
```

By matching the situation and delegate in one policy to the situation and policy issuer in another, one can see that P1, P2, and P4 form a delegation chain. P3 is not part of any delegation chain. Given the above delegation structure, consider how the following access request REQ1 will be resolved.

```
<Request REQ1>
      <Attributes>  /* :Attribute-Category   : Attribute ID   : Attribute
Value */
            :access-subject   :subject-id    :carol
            :access-subject   :role          :doctor
            :resource         :resource-id   :medical-records
            :action           :action-id     :read
      </Attributes>
</Request REQ1>
```

By matching the attributes (and values) in the request REQ1 with the attributes (and values) in the target of the policies in the policy set, one finds that only policies P3 and P4 match directly since policies P1 and P2 contain delegated attributes. Since both policies P3 and P4 are untrusted access policies, their respective authorities have to be verified by making administrative requests. Since policy P3 is not part of any delegation chain, its authority cannot be verified. However, the authority for policy P4 can be established by using the delegation chain P1, P2, P4.

The same PAP interface that is used to create access policies can be used to create the additional policies needed for supporting delegation – Untrusted Access Policies, Trusted Administrative Policies, and Untrusted Administrative Policies. This requires at least two classes of policy administrators. The first is a System-Administrator authorized to create Access Policies. The second is a Delegated-Administrator authorized to create Untrusted Administrative Policies or Untrusted Access Policies conforming to the situation or a subset of the situation authorized in any Trusted Administrative Policy currently in the policy repository.

3.7 XACML Reference Architecture

The XACML reference architecture defines necessary functional components (depicted in Figure 5) to achieve enforcement of its policies. The authorization process is a seven-step process that depends on four layers of functionality: Enforcement, Decision, Access Control Data, and Administration.

At its core is a PDP that computes decisions to permit or deny subject requests (to perform actions on resources). Requests are issued from, and PDP decisions are returned to, a PEP using a standardized request and response language. The PEP is implemented as a component of an operating environment that is tightly coupled with its application. A PEP may not generate requests in XACML syntax nor process XACML syntax-compliant responses. In order to convert access requests in native format (of the operating environment) to XACML access requests (or convert a PDP response in XACML to a native format), the XACML architecture includes a context handler. The context handler also provides additional attribute values for the access request context (retrieving them from PIP). In the reference architecture in Figure 5, the context handler is not explicitly shown as a component since it is assumed that it is an integral part of the PEP or PDP.

A request is comprised of attributes extracted from the PIP minimally sufficient for Target matching. The PIP is shown as one logical store, but in fact may comprise multiple physical stores. In computing a decision, the PDP queries policies stored in a PRP. If the attributes of the request are not sufficient for rule and policy evaluation, the PDP may request the context handler to search the PIP for additional attributes. Information and data stored in the PIP and PRP comprise the access control data and collectively define the current authorization state.

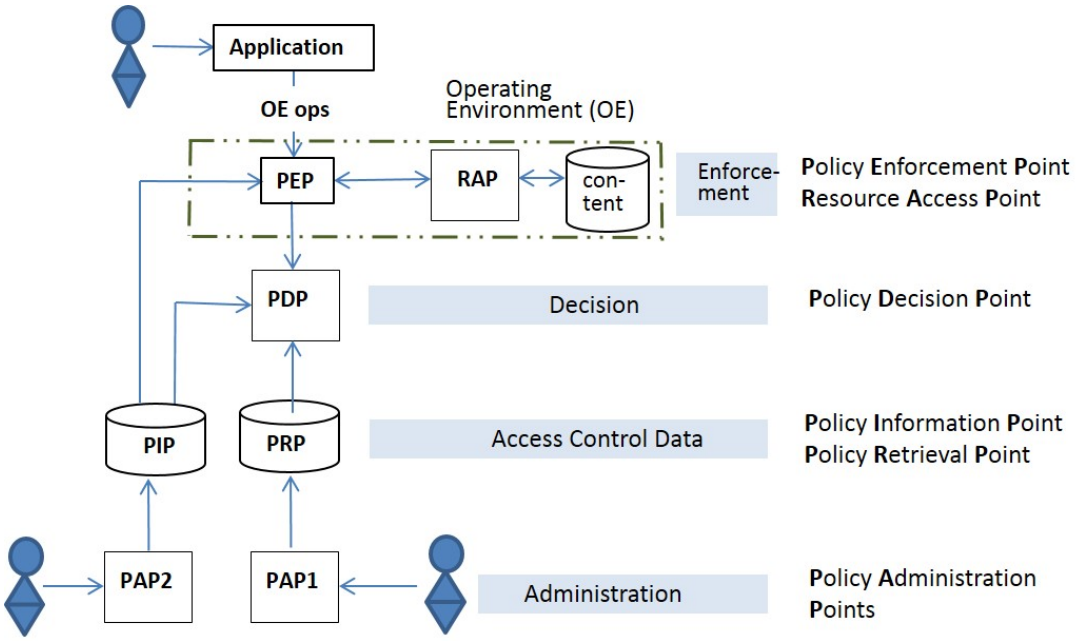

Figure 5: XACML Reference Architecture

A Policy Administration Point (PAP1) using the XACML policy language creates the access control data stored in the PRP in terms of rules for specifying Policies, PolicySets as containers of Policies, and rule and policy combining algorithms. The PRP may store trusted or untrusted policies. Although not included in the XACML reference architecture, a second Policy Administration Point (PAP2) is shown for creating and managing the access control data stored in the PIP. PAP2 implements administrative routines necessary for the creation and management of attribute names and values for users and resources. The Resource Access Point (RAP) implements routines for performing operations on a resource that is appropriate for the resource type. In the event that the PDP returns a permit decision, the PEP issues a command to the RAP for execution of an operation on resource content. As indicated by the dashed box in Figure 5, the RAP, in addition to the PEP, runs in an application's operating environment, independent of the PDP and its supporting components. The PDP and its supporting components are typically implemented as modules of a centralized Authorization Server that provides authorization services for multiple types of operations.

4 NGAC Specification

NGAC takes a fundamentally different approach from XACML for representing requests, expressing and administering policies, representing and administering attributes, and computing and enforcing decisions. NGAC is defined in terms of a standardized and generic set of relations and functions that are reusable in the expression and enforcement of policies.

For purposes of brevity and readability, this section presents a summary of the NGAC specification that highlights NGAC's salient features and should not be considered complete. In some instances, actual NGAC relational details and terms are substituted with others to accommodate a simpler presentation.

4.1 Basic Policy and Attribute Elements

NGAC's access control data is comprised of basic elements, containers, and configurable relations. While XACML uses the terms subject, action, and resource, NGAC uses the terms user, operation, and object with similar meanings. In addition to these, NGAC includes processes, administrative operations, and policy classes. Like XACML, NGAC recognizes user and object attributes; however, it treats attributes along with policy class entities as containers. These containers are instrumental in both formulating and administering policies and attributes.

NGAC treats users and processes as independent but related entities. NGAC processes can be thought of as simple representations of operating system processes. They have an id, memory, and descriptors for resource allocations (e.g., "handles"). Like an operating system, an NGAC process can utilize system resources (e.g., clipboard) for inter-process communication. Processes through which a user attempts access take on the same attributes as the invoking user.

Although an XACML resource is similar to an NGAC object, NGAC uses the term object as an indirect reference to its data content. Given this one-to-one correspondence, the object can also be identified as an object attribute with the same name. The set of objects reflects entities needing protection, such as files, clipboards, email messages, and record fields.

Similar to an XACML subject attribute value, NGAC user containers can represent roles, affiliations, or other common characteristics pertinent to policy, such as security clearances.

Object containers (attributes) characterize data and other resources by identifying collections of objects, such as those associated with certain projects, applications, or security classifications. Object containers can also represent compound objects, such as folders, inboxes, table columns, or rows, to satisfy the requirements of different data services. Policy class containers are used to group and characterize collections of policy or data services at a broad level, with each container representing a distinct set of related policy elements. Every user, user attribute, and object attribute must be contained in at least one policy class. Policy classes can be mutually exclusive or overlap to various degrees to meet a wide range of policy requirements.

NGAC recognizes a generic set of operations that include basic input and output operations (i.e., read and write) that can be performed on the contents of objects that represent data service resources, and a standard set of administrative operations that can be performed on NGAC

access control data representing policies and attributes. In addition, an NGAC deployment may consider and provide control over other types of data service operations besides the basic input/output operations. Resource operations can also be defined specifically for an operating environment. Administrative operations, on the other hand, pertain only to the creation and deletion of NGAC data elements and relations, and are a stable part of the NGAC framework, regardless of the operating environment.

4.2 Relations

NGAC does not express policies through rules, but instead through configurations of relations of four types: assignments (define membership in containers), associations (derive privileges), prohibitions (specify privilege exceptions), and obligations (dynamically alter access state).

4.2.1 Assignments and Associations

NGAC uses a tuple (*x*, *y*) to specify the assignment of element *x* to element *y*. In this publication the notation $x \rightarrow y$ is used to denote the same assignment relation. The assignment relation always implies containment (*x* is contained in *y*). A chain of one or more assignment relations is denoted by "$\rightarrow^{+}$". The set of entities used in assignments includes users, user attributes, object attributes (which include all objects), and policy classes.

To be able to carry out an operation, one or more access rights are required. As with operations, two types of access rights apply: non-administrative and administrative.

Access rights to perform operations are acquired through associations. An association is a triple, denoted by *ua---ars---at*, where *ua* is a user attribute, *ars* is a set of access rights, and *at* is either a user attribute or an object attribute. The attribute *at* in an association is used as a referent for itself and the policy elements contained by the attribute. Similarly, the first term of the association, attribute *ua*, is treated as a referent for the users and user attributes contained in *ua*. The meaning of the association *ua---ars---at* is that the users contained in *ua* can execute the access rights in *ars* on the policy elements referenced by *at*. The set of policy elements referenced by *at* is dependent on (and meaningful to) the access rights in *ars*.

Figure 6 illustrates two example assignment and association relations depicted as graphs—one an access control policy configuration with policy class "Project Access" (Figure 6a), and the other a data service configuration with "File Management" as its policy class (Figure 6b). Users and user attributes are on the left side of the graphs, and objects and object attributes are on the right. The arrows represent assignment relations and the dashed lines denote associations. Remember that the set of referenced policy elements is dependent on the access rights in *ars*. Note that the *at* attribute of each association is an object attribute and the access rights are read/write. In the association Division---{r}---Projects, the policy elements referenced by Projects are objects o1 and o2, meaning that users u1 and u2 can read objects o1 and o2. If there was an association Division---{create assign-to}---Projects, then the policy elements referenced by Projects would be Projects, Project1, and Project2, meaning that users u1 and u2 may (administratively) create assignment relations to Projects, Project1, and Project2.

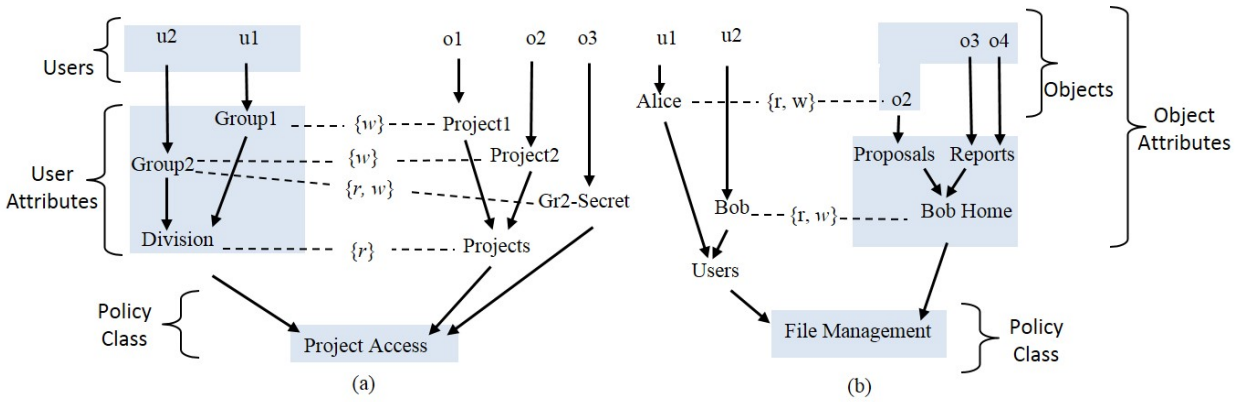

Figure 6: Two Example Assignment and Association Graphs

4.2.2 Derived Privileges

Collectively associations and assignments indirectly specify privileges of the form (*u*, *ar*, *e*), with the meaning that user *u* is permitted (or has a capability) to execute the access right *ar* on element *e*, where *e* can represent a user, user attribute, or object attribute. Determining the existence of a privilege (a derived relation) is a requirement of, but not sufficient for, computing an access decision, as discussed later.

NGAC includes an algorithm for determining privileges with respect to one or more policy classes and associations. Specifically, (*u*, *ar*, *e*) is a privilege if and only if, for each policy class *pc* in which *e* is contained, the following is true:

- The user *u* is contained by the user attribute of an association;
- The element *e* is contained by the attribute *at* of that association;
- The attribute *at* of that association is contained by the policy class *pc*; and
- The access right *ar* is a member of the access right set of that association.

Note that the algorithm for determining privileges applies to configurations that include one or more policy classes. The left and right columns of Table 2 list derived privileges for Figure 6a and 6b, when considered independent of one another.

Table 2: Derived Privileges for the Independent Configuration of Figures 6a and 6b

(u1, r, o1), (u1, w, o1), (u1, r, o2), (u2, r, o1), (u2, r, o2), (u2, w, o2), (u2, r, o3), (u2, w, o3)	(u1, r, o2), (u1, w, o2), (u2, r, o2), (u2, w, o2), (u2, r, o3), (u2, w, o3), (u2, r, o4), (u2, w, o4)

Figure 7 is an illustration of the graphs in Figure 6a and 6b when considered in combination. Note that for the purposes of deriving privileges, user attribute to policy class assignments are not considered, and as such are not shown.

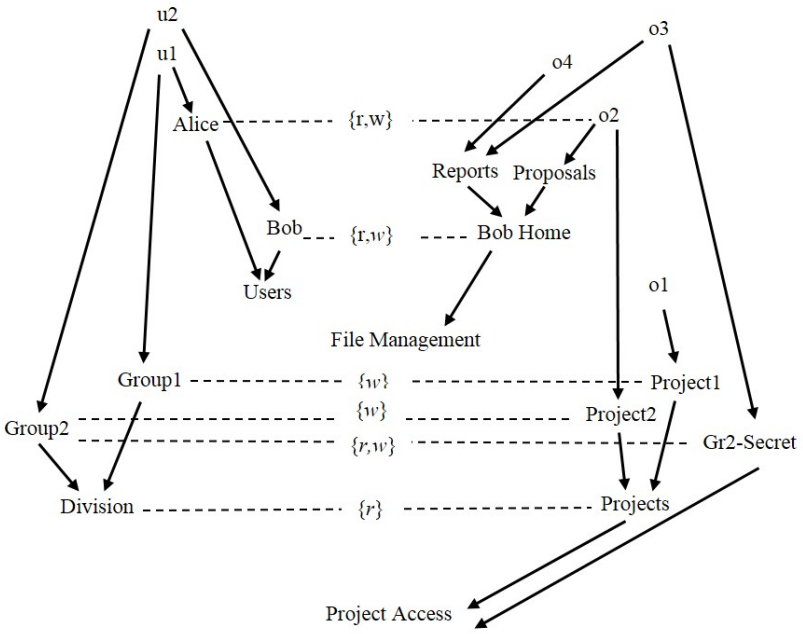

Figure 7: Graphs from Figures 6a and 6b in Combination

Table 3 lists the derived privileges for the graphs from Figure 6a and 6b when considered in combination.

Table 3: Derived Privileges for the Combined Configuration of Figures 6a and 6b

(u1, r, o1), (u1, w, o1), (u1, r, o2), (u2, r, o1), (u2, r, o2), (u2, w, o2), (u2, r, o3), (u2, w, o3), (u2, r, o4), (u2, w, o4)

Note that (u1, r, o1) is a privilege in Table 3 because o1 is only in policy class “Project Access” and there exists an association Division---{r}--- Projects, where u1 is in Division, r is in {r}, and o1 is in Projects. Note that (u1, w, o2) is not a privilege in Table 3 because o2 is in both Project Access and File Management policy classes, and although there exists an association Alice---{r, w}---o2, where u1 is in Alice, w is in {r, w}, and o2 is in o2 and File Management, no such association exists with respect to Project Access.

NGAC configurations indirectly specify rules. The access control policy of Figure 6a specifies that users assigned to either Group1 or Group2 can read objects contained in Projects, but only Group1 users can write to Project1 objects and only Group2 users can write to Project2 objects. The Policy further specifies that Group2 users can read/write data objects in Gr2-Secret. While Figure 6a specifies policies for how its objects can be read and written, the configuration is considered incomplete in that it does not specify how its users, objects, policy elements, assignments, and associations were created and can be managed.

Figure 6b depicts an access policy for a File Management data service. User u2 (Bob) has read/write access to objects assigned to object attributes (Proposals and Reports representing

folders) that are contained in Bob Home (representing his home directory). The configuration also shows user u1 (Alice) with read/write access to object o2. This configuration is also incomplete in that one would expect a File Management data service with capabilities for users to create and manage their folders and to create and assign objects to their folders. Another feature common to a File Management data service is the capability for users to grant or give away access rights to objects that are under their control to other users.

The missing management capabilities for the Project Access policy are specified in Section 4.4.1 and File Management data service in Section 4.5.

Although the graph depicted in Figure 7 pertains to the intersection of policies, NGAC employs the Boolean logics of AND and OR to express the combinations of policies [27]. Figure 8 is a depiction of an NGAC equivalent configuration of the XACML Policy1 specified in Section 3.4. Both policies specify that users assigned to Intern can read AND Doctor can read and write Medical Records that are assigned to the same Ward as the user OR Doctors can read and write Medical Records assigned to Critical regardless of the Ward in which the Medical Record is assigned.

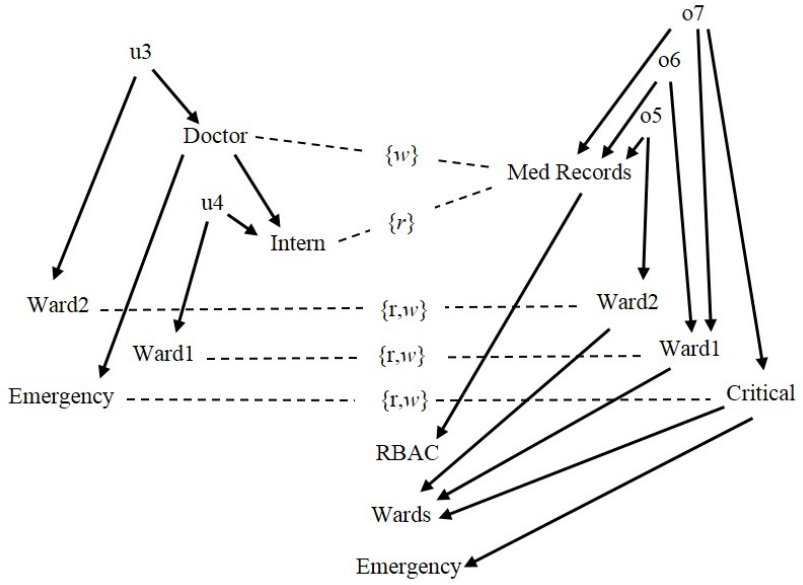

Figure 8: NGAC's Equivalent Expression of XACML Policy1

Figure 8 shows NGAC users and objects that correspond to the XACML subjects and resources in Table 1 and are assigned to the same attribute values in Table 1. As a consequence, the derived privileges of Figure 8 (listed in Table 4) are the same as the authorization state specified in Table 1.

Table 4: Derived Privileges for the Configuration of Figure 8

(u3, r, o5), (u3, w, o5), (u3, r, o7), (u3, w, o7), (u4, r, o6)

4.2.3 Prohibitions (Denies)

In addition to assignments and associations, NGAC includes three types of prohibition relations: user-deny, user attribute-deny, and process-deny. In general, deny relations specify privilege exceptions. User deny, user attribute deny, and process deny relations are respectively denoted by u_deny(*u*, *ars*, *pe*), ua_deny(*ua*, *ars*, *pe*), and p_deny(*p*, *ars*, *pe*), where *u* is a user, *ua* is a user attribute, *p* is a process, *ars* is an access right set, and *pe* is a policy element used as a referent for itself and the policy elements contained by the policy element. The respective meanings of these relations are that user *u*, users in *ua*, and process *p* cannot execute access rights in *ars* on policy elements in *pe*. User-deny relations and user attribute-deny relations can be created directly by an administrator or dynamically as a consequence of an obligation (see Section 4.2.4). An administrator, for example, could impose a condition where no user is able to alter his or her own Tax Return, in spite of the fact that the user is assigned to an IRS Auditor user attribute with capabilities to read/write all tax returns. When created through an obligation, user-deny and user attribute-deny relations can take on dynamic policy conditions. Such conditions can, for example, provide support for separation of duty policies (if a user executed capability x, that user would be immediately precluded from being able to perform capability y). In addition, the policy element component of each prohibition relation can be specified as its complement, denoted by ¬. The respective meaning of u_deny(*u*, *ars*, ¬*pe*), ua_deny(*ua*, *ars*, ¬*pe*), and p_deny(*p*, *ars*, ¬*pe*) is that the user *u*, any user assigned to *ua*, and process *p* cannot execute the access rights in *ars* on policy elements not in *pe*.

Process-deny relations are exclusively created using obligations. Their primary use is in the enforcement of confinement conditions (e.g., if a process reads Top Secret data, preclude that process from writing to any object not in Top Secret).

4.2.4 Obligations

Obligations consist of a pair (*ep*, *r*) (usually expressed as **when** *ep* **do** *r*) where *ep* is an *event pattern* and *r* is a sequence of administrative operations, called a *response*. The event pattern specifies conditions that, if matched by the context surrounding a process's successful execution of an operation on an object (an event), cause the administrative operations of the associated response to be immediately executed. The context may pertain to and the event pattern may specify parameters like the user of the process, the operation executed, and the attribute(s) of the object.

Obligations can specify operational conditions in support of history-based policies and data services. Such conditions include conflict of interest (if a user reads information from a sensitive data set, that user is prohibited from reading data from a second data set) and workflow (approving (writing to a field of)) a work item enables a second user to read and approve the work item). Also, included among history-based policies are those that prevent leakage of data to unauthorized principals. The use of an obligation to prevent data leakage is discussed in Section 4.5.

4.3 NGAC Decision Function

The NGAC access decision function controls accesses in terms of processes. The user on whose behalf the process operates must hold sufficient authority over the policy elements involved. The function process_user(*p*) denotes the user associated with process *p*.

Access requests are of the form (*p*, *op*, *argseq*), where *p* is a process, *op* is an operation, and *argseq* is a sequence of one or more arguments that is compatible with the scope of the operation. That is, an access request comprises an operation and a list of enumerated arguments that have their number, type, and order dictated by the operation.

The access decision function to determine whether an access request can be granted requires a mapping from an operation and argument sequence pair to a set of access rights and policy element pairs (i.e., {(*ar*, *pe*)}) the process's user must hold for the request to be granted.

When determining whether to grant or deny an access request, the authorization decision function takes into account all privileges and restrictions (denies) that apply to a user and its processes, which are derived from relevant associations and denies, giving restrictions precedence over privileges:

> A process access request (*p*, *op*, *argseq*) with mapping (*op*, *argseq*)→{(*ar*, *pe*)}) is granted iff for each (ar_i, pe_i) in {(*ar*, *pe*)}, there exists a privilege (*u*, ar_i, pe_i) where *u* = process_user(*p*), and (ar_i, pe_i) is not denied for either *u* or *p*.

In the context of Figure 7, an access request may be (p, read, o1) where p is u1's process. The pair (read, o1) maps to (r, o1). Because there exists a privilege (u1, r, o1) in Table 3 and (r, o1) is not denied for u1 or p, the access request would be granted. Assume the existence of associations Division---{create assign-to}---Projects, and Bob---{create assign-from}---Bob Home in the context of Figure 7, and an access request (p, assign, <o4, Project1>) where p is u2's process. The pair (assign, <o4, Project1>) maps to {(create assign-from, o4), (create assign-to, Project1)}. Because privileges (u2, create assign-from, o4) and (u2, create assign-to, Project1) would exist under the assumption, and (create assign-from, o4) and (create assign-to, Project1) are not denied for u2 or p, the request would be granted.

4.4 Administrative Considerations

Many access rights categorized as administrative access rights, such as those needed to create a file and assign it to a folder, arguably seem non-administrative from a usage standpoint. Nevertheless, from a policy specification standpoint, they are considered administrative (e.g., in this case, an association with access rights for creating an object and assigning the object to an object attribute is needed). The main difference between the two types of access rights is that non-administrative actions pertain to activities on protected resources represented as objects, while administrative actions pertain to activities on the policy and attribute representation comprising the policy elements and relationships defined within and maintained by NGAC.

4.4.1 Administrative Associations

In order to execute an administrative operation, the requesting user must possess appropriate access rights. Just as access rights to perform read/write operations on objects representing resources are defined in terms of associations, so too are capabilities to perform administrative operations on policy elements and relations. In comparison with non-administrative access rights, where resource operations are synonymous with the access rights needed to carry out those operations (e.g., a "read" operation corresponding to an "r" access right), the authority associated with an administrative access right is not necessarily synonymous with an administrative operation. Instead, the authority stemming from one or more administrative access rights may be required for a single operation to be authorized.

Some administrative access rights are explicitly divided into two parts, as denoted by the "from" and "to" suffixes. Both parts of the authority must be held to carry out the implied administrative operation.

For example, consider the following two associations that provide administrative capabilities in support of the "Project Access" policy configuration depicted in Figure 6a:

ProjectAccessAdmin --- {create-u-to, delete-u-from, create-ua-to, delete-ua-from, create-uua-from, create-uua-to, delete-uua-from, create-uaua-from, create-uaua-to, delete-uaua-from, delete-uaua-to }---Division

ProjectAccessAdmin --- {create-o-to, delete-o-from, create-oa-to, delete-oa-to, create ooa-from, create ooa-to, delete-ooa-from, create-oaoa-from, create-oaoa-to, delete-oaoa-from, delete-oaoa-to }--- Projects

The meaning of the first association is that users in ProjectAccessAdmin can create and delete users, user attributes, user to user-attribute (uua), and user-attribute to user-attribute (uaua) assignments in Division. The second association similarly establishes privileges to create and delete objects(o), object attributes(oa), object to object-attribute (ooa), and object-attribute to object-attribute (oaoa) assignments in Projects.

With the preceding two associations, the next two associations complete the configuration begun by the configuration of Figure 6a, enabling complete administration. The associations enable users in ProjectAccessAdmin to create and delete associations from user attributes in Division to object attributes in Projects, with allocated read and/or write access rights.

ProjectAccessAdmin --- {create-assoc-from, delete-assoc-from} --- Division.
ProjectAccessAdmin --- {create-assoc-to, delete-assoc-to, r-allocate, w-allocate} --- Projects.

4.4.2 Delegation

The question remains, how are administrative capabilities created? The answer begins with a superuser with capabilities to perform all administrative operations on all access control data. The initial state consists of an NGAC configuration with empty data elements, attributes, and relations. A superuser either can directly create administrative capabilities or more practically can create administrators and delegate to them capabilities to create and delete administrative

privileges. Delegation and rescinding of administrative capabilities is achieved through creating and deleting associations. The principle followed for allocating access rights via an association is that the creator of the association must have been allocated the access right over the attribute in question (as well as the necessary create-assoc-from and create-assoc-to rights) in order to delegate them. The strategy enables a systematic approach to the creation of administrative attributes and delegation of administrative capabilities, beginning with a superuser and ending with users with administrative and data service capabilities.

4.4.3 NGAC Administrative Commands and Routines

Administrative commands and routines are the means by which policy specifications are formed. Each access request involving an administrative operation corresponds on a one-to-one basis to an administrative routine, which uses the sequence of arguments in the access request to perform the access. As described earlier in this section, the access decision function grants the access request (and initiation of the respective administrative routine) only if the process holds all prohibition-free access rights over the items in the argument sequence needed to carry out the access. The administrative routine, in turn, uses one or more administrative commands to perform the access.

Administrative commands describe rudimentary operations that alter the policy elements and relationships of NGAC, which comprise the authorization state. An administrative command is represented as a parameterized procedure, with a body that describes state changes to policy that occur when the described behavior is carried out (e.g., a policy element or relation Y changes state to Y′ when some function f is applied). Administrative commands are specified using the following format:

cmdname (x_1: $type_1$, x_2: $type_2$, …, x_k: $type_k$)
…*preconditions* …
{
$Y' = f(Y, x_1, x_2, \ldots, x_k)$
}

Consider, as an example, the administrative command CreateAssoc shown below, which specifies the creation of an association. The preconditions here stipulate membership of the x, y, and z parameters respectively to the user attributes (UA), access right sets (ARS), and policy elements (PE) elements of the model. The body describes the addition of the tuple (x, y, z) to the set of associations (ASSOC) relation, which changes the state of the relation to ASSOC′.

createAssoc (x, y, z)
$x \in UA \wedge y \in ARS \wedge z \in PE \wedge (x, y, z) \notin ASSOC$
{
$ASSOC' = ASSOC \cup \{(x, y, z)\}$
}

Each administrative command entails a modification to the NGAC configuration that involves the creation or deletion of a policy element, an assignment between policy elements, or an association, prohibition, or obligation.

Compared to administrative routines, administrative commands are elementary. That is, administrative commands provide the foundation for the NGAC framework, while administrative routines use one or more administrative commands to carry out their functions.

An administrative routine consists mainly of a parameterized interface and a sequence of administrative command invocations. Administrative routines build upon administrative commands to define the protection capabilities of the NGAC model. The body of an administrative routine is executed as an atomic transaction—an error or lack of capabilities that causes any of the constituent commands to fail execution causes the entire routine to fail, producing the same effect as though none of the commands were ever executed. Administrative routines are specified using the following format:

rtnname (x_1: $type_1$, x_2: $type_2$, …, x_k: $type_k$)
… *preconditions* …
{
cmd_1;
$condition_a$ cmd_2, cmd_3;
. . .
$condition_z$ cmd_n;
}

The name of the administrative routine, rtnname, precedes the routine's declaration of formal parameters, x1: type1, x2: type2, …, xk: typek ($k \geq 0$). Each formal parameter of an administrative routine can serve as an argument in any of the administrative command invocations, cmd1, cmd2, …, cmdn ($n \geq 0$), that make up the body of the routine, and also in any condition prepended to a command. As with an administrative command, the body of an administrative routine is prefixed by preconditions, which in general ensure that the arguments supplied to the routine are valid, and that certain properties on which the routine relies are maintained. As illustrated above, an optional condition can precede one or more of the commands.

For example, when a new user is created, an administrator typically creates a number of containers, links them together, and grants the authority for the user to access them as its work space. Rather than manually performing each step of this sequence of administrative actions for each new user, the entire sequence of repeated actions can be defined as a single administrative routine and executed in its entirety as an atomic action.

To execute the routine, the user (administrative) must possess the necessary capabilities to execute each administrative command.

4.5 Arbitrary Data Service Operations and Policies

NGAC recognizes administrative operations for the creation and management of its data elements and relations that represent policies and attributes, and basic input and output operations (e.g., read and write) that can be performed on objects that represent data service resources. In accommodating data services, NGAC may establish and provide control over other types of operations, such as send, submit, approve, and create folder. However, it does not

necessarily need to do so. This is because the basic data service capabilities to consume, manipulate, manage, and distribute access rights on data can be attained as combinations of read/write operations on data and administrative operations on data elements, attributes, and relations that may alter the access state for which users can read/write data.

Consider the following administrative routine that creates a "file management" user and provides the user with capabilities to create and manage objects and folders, and control and share access to objects in the context of Figure 6b. The routine assumes the pre-existence of the user attribute "Users" assigned to the "File Management" policy class as shown in Figure 6b.

```
create-file-mgmt-user(user-id, user-name, user-home) {
     createUAinUA(user-name, Users);
     createUinUA(user-id, user-name);
     createOAinPC(user-home, File Management);
     createAssoc(user-name, {r, w}, user-home);
     createAssoc(user-name, {create-o-to, delete-o-from}, user-home);
     createAssoc(user-name, {create-ooa-from, create-ooa-to,
                             delete-ooa-from, create-oaoa-from,
                             create-oaoa-to, delete-oaoa-from}, user-home);
     createAssoc(user-name, {create-assoc-from, delete-assoc-from}, Users);
     createAssoc(user-name, {create-assoc-to, delete-assoc-to, r-allocate,
                             w-allocate}, user-home);}
```

This routine with parameters (*u*1, *Bob*, and *Bob Home*) could have been used to create "file management" data service capabilities for user *u*1 already in Figure 6b. Through the routine the user attribute "Bob" is created and assigned to "Users", and user *u*1 is created and assigned to "Bob". In addition, the object attribute "Bob Home" is created and assigned to policy class "File Management". In addition, user *u*1 is delegated administrative capabilities to create, organize, and delete object attributes (presented folders) in Bob Home, and *u*1 is provided with capabilities to create, read, write, and delete objects that correspond to files and place those files into his folders. Finally, *u*1 is provided with discretionary capabilities to "grant" to other users in the "Users" container capabilities to perform read/write operations on individual files or to all files in a folder in his Home.

As already indicated by Figure 6b, and subsequent to the execution of this administrative routine, user *u*1 can grant user *u*2 (Alice) read/write access to object *o*2 by using the following routine:

```
grant(user-name, rights, file/folder) {
   createAssoc(user-name, rights, file/folder)}
```

Through this routine Bob could, under his discretion, "grant" Alice read access to o3. However, even if Bob were to do so, Alice would not be able to read o3. This is because of a lack of a privilege (u1, r, o3) due to o3's containment in the "Project Access" policy class. Although Bob cannot successfully provide Alice read access to object o3 through his delegated "grant" capability, Bob could "leak" the capability to read the content of o3 to Alice. This could be achieved by Bob first reading the content of o3 and then writing that content to o2. Even if Bob was trusted not to perform such actions, a malicious process acting on Bob's behalf could do so, without Bob's knowledge. To prevent leakage, the following obligation is added to the configuration:

When any process *p* performs (*r*, *o*) where $o \rightarrow^{+}$ Gr2-Secret **do** create p-deny(*p*, {*w*}, ¬Gr2-Secret)

The effect of this obligation will prevent a process (and its user) from reading the contents of any object in Gr2-Secret and writing it to an object in a different container (not in Gr2-Secret).

4.6 NGAC Functional Architecture

NGAC's functional architecture (shown in Figure 9), like XACML's, encompasses four layers of functional decomposition: Enforcement, Decision, Administration, and Access Control Data. This architecture involves several components that work together to bring about policy-preserving access and data services. Among these components is a PEP that traps application requests. An access request includes a process id, user id, operation, and a sequence of one or more operands mandated by the operation that pertain to either a data resource or an access control data element or relation. Administrative operational routines are implemented in the PAP and read/write routines are implemented in the RAP.

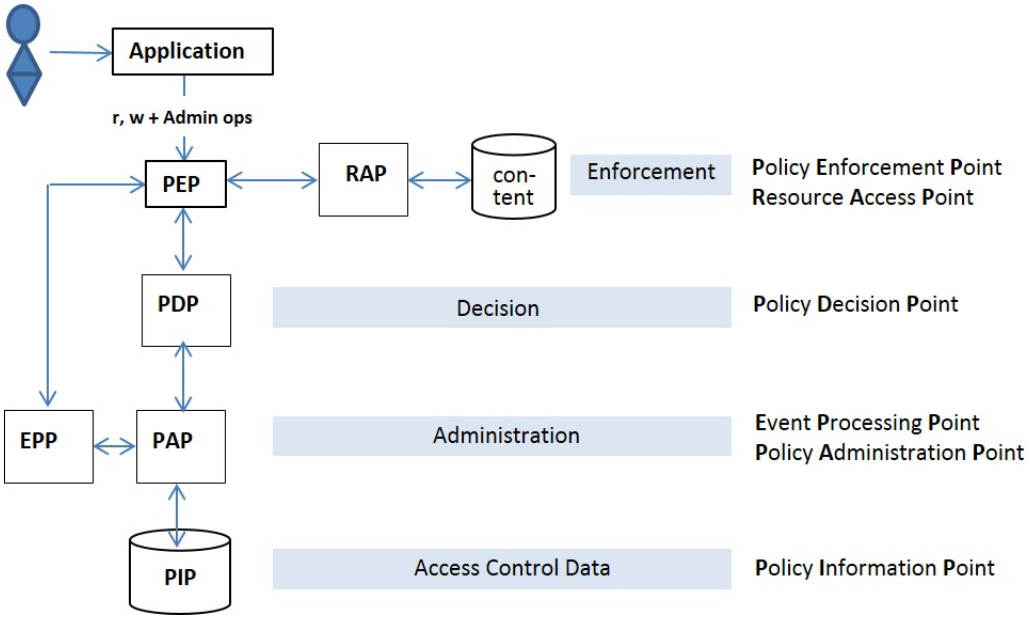

Figure 9: NGAC Standard Functional Architecture

To determine whether to grant or deny, the PEP submits the request to a PDP. The PDP computes a decision based on the current configuration of data elements and relations stored in the PIP, via the PAP. Unlike the XACML architecture, the access request information from an NGAC PEP together with the NGAC relations (retrieved by the PDP) provide the full context for arriving at a decision. The PDP returns a decision of grant or deny to the PEP. If access is granted and the operation was read/write, the PDP also returns the physical location where the object's content resides, the PEP issues a command to the appropriate RAP to execute the operation on the content, and the RAP returns the status. In the case of a read operation, the RAP also returns the data type of the content (e.g., PowerPoint) and the PEP invokes the correct data service application for its consumption. If the request pertained to an administrative operation and the decision was grant, the PDP issues a command to the PAP for execution of the operation

on the data element or relation stored in the PIP, and the PAP returns the status to the PDP, which in turn relays the status to the PEP. If the returned status by either the RAP or PAP is "successful", the PEP submits the context of the access to the Event Processing Point (EPP). If the context matches an event pattern of an obligation, the EPP automatically executes the administrative operations of that obligation, potentially changing the access state. Note that NGAC is data type agnostic. It perceives accessible entities as either data or access control data elements or relations, and it is not until after the access process is completed that the actual type of the data matters to the application.

5 Analysis

XACML is similar to NGAC insofar as they both provide flexible, mechanism-independent representations of policy rules that may vary in granularity, and they employ attributes in computing decisions. However, XACML and NGAC differ significantly in their expression of policies, treatment of attributes, computation of decisions, and representation of requests. In this section, these similarities and differences are analyzed with respect to the degree of separation of access control functionality from proprietary operating environments and four ABAC considerations identified in NIST SP 800-162: scope and type of policy support, operational efficiency, attribute and policy management, and support for administrative review and resource discovery.

For the purposes of comparison, some of XACML and NGAC's terminology is normalized.

5.1 Separation of Access Control Functionality from Proprietary Operating Environments

XACML and NGAC both separate access control functionality of data services from proprietary operating environments, but to different degrees. An XACML deployment may consist of multiple operating environments, each hosting one or more applications and sharing a common authorization infrastructure. Each of these operating environments implements its own method of authentication, and in support of its applications implements its own operational routines. Application specific operations included in XACML access requests correspond one-to-one with operational routines implemented in supporting operating environments. It is for this reason that an XACML-enabled application is dependent on an operating environment PEP. Requests are issued from, and decisions are returned to, an operating environment-specific PEP.

Although an NGAC deployment could include a PEP with an application programming interface (API) that recognizes operating environment-specific operations (e.g., send and forward operations for a messaging system), it does not necessarily need to do so. NGAC includes a PEP with an API that supports a set of generic, operating environment-agnostic operations (read, write, create, and delete policy elements and relations). This API enables a common, centralized PEP to be implemented to serve the requests of multiple applications. Although the generic operations may not meet the requirements of every application (e.g., transactions that perform computations on attribute values), calls from many applications can be accommodated. This includes operations that generically pertain to consumption, manipulation, and management of data, and distribution of access rights on data. For example, the "send" operation of a messaging data service could be implemented through a series of administrative operations on NGAC data elements and relations, where "inboxes" and "outboxes" are represented as object attributes. The administrative operations create and assign a message (an object) to the "outbox" of the sender and the "inbox" of the recipient, where the sender and recipient have read access rights to objects contained in their respective "outbox" and "inbox". The file management data service described in Section 4 is another example of a data service that supports application specific operations for creating and managing files and folders implemented though NGAC generic operations. Still others could include operations in support of workflow, calendar, record management, and time and attendance.

XACML does not envisage the design of a PEP that is data service agnostic. In other words, a PEP under the XACML architecture is tightly coupled to a specific operating environment for which it was designed to enforce access. However, based on the deployment feature described above, it is possible for the NGAC PEP to provide a level of abstraction between application calls and underlying object types and their associated privileges.

As a consequence of this abstraction capability, NGAC can completely displace the need for an access control mechanism of an operating environment in that through the same API, set of operations, access control data elements and relations, and functional components, arbitrary data services can be delivered to users, and arbitrary, mission-tailored access control policies can be expressed and enforced over executions of application calls.

5.2 Scope and Type of Policy Support

Access control policy is a broad term that pertains to many types of controls. For purposes of this report, these controls are divided into two broad categories: discretionary access control (DAC) and mandatory access control (MAC). In addition, MAC is further categorized into two subcategories: MAC policies that support confinement and MAC policies that do not.

DAC is an administrative policy that permits system users to allow or disallow other users' access to resources/objects under their control. The means of restricting access to objects is often based on the identities of users and/or the attributes to which they are assigned. The controls are discretionary in the sense that a user with access to a resource is capable of passing that access on to other users without the intercession of a system administrator [18]. Although XACML can theoretically implement DAC policies, it is not efficient. Consider the propagation feature of DAC. DAC permits owners/creators of objects to grant some or all of their capabilities to other users, and the grantees can further propagate those capabilities on to other users. The overall DAC feature to grant privileges to another user and the ability of the grantee to propagate those privileges cannot be supported in XACML syntax using "Access Policies" alone. XACML is geared for specifying global access policies in terms of attributes. Since the only user attribute designator is "access-subject", there is no predefined attribute category to denote the owner/creator of an object.

Therefore, all the capabilities of the owner/creator of an object together with administrative capabilities to grant those privileges have to be specified using a Trusted Administrative policy. The capabilities held by owner/creator can be captured by designating the owner/creator of the object as the "access-subject", and the administrative capability to grant privileges to others can be captured by designating the owner/creator as a delegate in that policy type. The creation of this trusted administrative policy, in turn, enables creation of derived administrative policies with the owner/creator as the policy issuer with the specified set of capabilities. Further, the specification of a "delegate" in this derived administrative policy (labeled NOT TRUSTED) provides a means for the owner/creator to grant capabilities to other users, as well as the ability for the grantee to propagate those capabilities to other users. However, while it is theoretically possible to implement DAC by leveraging XACML's delegation feature, this approach involves significant administrative overhead. The solution requires the specification of a trusted administrative policy and a set of derived administrative policies for every object owner/creator, and for all grantees of the capabilities.

NGAC offers a flexible means of providing users with administrative capabilities, to include those necessary for the implementation of different flavors of DAC. As shown by the execution of the administrative routine "create-file-mgmt-user(user-id, user-name, user-home)" in Section 4.5, user *u*1 (Bob) is created and given "File Management" data service capabilities. These capabilities include being able to create objects and assign them to his home, and consequently, having read/write access to those objects. In addition, Bob is given ownership and control capabilities over objects in his home (i.e., Bob can grant other users (e.g., Alice) read/write access to any object in his home). Because Alice is also a "File Management" user, Alice could create a copy of the object, place it in her home, and grant other users access to her copy.

In contrast to DAC, MAC enables ordinary users' capabilities to execute resource operations on resource objects, but not administrative capabilities that may influence those capabilities. MAC policies unavoidably impose rules on users in performing operations on resource objects.

Expression of MAC policies is perhaps XACML's strongest suit. XACML can specify rules in terms of attribute values that can be of varying types, such as strings and integers. There are undoubtedly certain policies that are expressible in terms of these rules that cannot be easily accommodated by NGAC. For example, a financial transaction may pertain to adding a person's credit limit to their account balance. XACML also takes into consideration environmental attributes in expressing policies, and NGAC does not directly support such policies. These environmental-driven policies are dynamic in nature in that the authorization state can change without the involvement of any administrative action. For instance, the threat level can change from "Low" to "High". XACML also includes the notion of an obligation that directs a PEP to take an action prior to or after an access request is approved or denied. XACML obligation can complement and refine MAC policies in a number of ways. While NGAC also uses the term obligation, an NGAC obligation refers to a different policy construct.

MAC policies are often dependent on and include administrative policies. This is especially true in a federated or collaborative environment, where governance policies require different organizational entities to have different responsibilities for administering different aspects of policies and their dependent attributes. It is also often desirable to be able to express policies that prevent combinations of resource capabilities and administrative capabilities—for example, a policy that would prevent an administrator from granting him/herself access to sensitive resources. XACML is ill suited to naturally express such policies. Consider the MAC policy depicted by Figure 6a. Although XACML can certainly express and enforce this policy, it cannot easily express policies as to who can assign users to the various groups (attributes), while NGAC can. NGAC can create administrative attributes and provide users with administrative capabilities down to the granularity of a single configuration element. Furthermore, NGAC can deny administrative capabilities down to the same granularity.

Although XACML has been shown to be capable of expressing aspects of standard RBAC [19] through an XACML profile [20], the profile falls short of demonstrating support for dynamic separation of duty, a key feature used for accommodating the principle of least privilege, and static separation of duty, a key feature for combatting fraud. Annex B of draft standard Next Generation Access Control – Generic Operations and Data Structures (NGAC-GOADS) [13] demonstrates NGAC support for all aspects of the RBAC standard. The appendix also

demonstrates support for the Chinese wall policy [21], which cannot be entirely accommodated by XACML.

NGAC has shown support for history-based separation of duty [3]. Simon and Zurko, in their seminal paper on separation of duty [22], describe history-based separation of duty as the most accommodating form of separation of duty, subsuming the policy objectives of other forms. Other history-based policies that can be accommodated by NGAC include two-person control, workflow, and conflict-of-interest.

Despite the use of attributes, the policies discussed thus far have resulted in a user-based authorization state. In other words, the policies and attributes together constitute an authorization state of the form {(*u*, *ar*, *o*)}, where user *u* is authorized to access object *o* under the access right *ar*. Such policies ignore the fact that processes, not users, actually access object content. In general, user-based authorization controls (whether MAC or DAC) share a weakness: their inability to prevent the "leakage" of data to unauthorized principals through malware, or through malicious or complacent user actions.

To illustrate this weakness, assume the following authorization state {(u1, r, o1), (u1, w, o2), and (u2, r, o2)}. Note that it is impossible to determine if u2 can read the content of o1. Under one scenario, u1 can read and subsequently write the contents of o1 to o2. Even if policy depended on "trust in users", the existence of a Trojan horse that can easily thwart policy must be assumed. This threat exists because, in reality, users do not perform operations on objects, but under a user's capabilities, processes perform operations (actions) on the content of objects (resources). Therefore, a program executed by u1 can read the contents of o1 and, without u1's further action or knowledge, write that content to o2. Note that one cannot prevent this leakage even with the addition of a user-based deny condition or relation NOT (u2, r, o1). The importance of preventing inappropriate leakage of data (often called confinement) was recognized as early as the 1970s, with the establishment of the Bell and LaPadula security model [23] and the specific MAC policy defined in Trusted Computer Security Evaluation Criteria (TCSEC) [24].

Because XACML does not allow the specification and enforcement of policies that pertain to processes in isolation of their users, it excludes or imposes undue constraints on users in regard to MAC confinement policies. Another drawback of XACML is that its PDP is stateless, which places limitations on the policies that can be specified and enforced. Although XACML includes the concept of an obligation, it is not used to alter authorization state.

Consider the following XACML TCSEC MAC policy specification:

```
<Policy PolicyId = "Policy 3" rule-combining-algorithm="only-one-applicable">
            // TCSEC MAC Policy Specification //
     <Target> /* Policy applies to all subjects with clearance levels – Top-
Secret, Secret, or Unclassified and resources with classification levels –
Top-Secret, Secret, or Unclassified for both "read" and "write" actions */
        /* :Attribute-Category    :Attribute ID    :Attribute Value */
              :access-subject      :Clearance        :Top-Secret
              :access-subject      :Clearance        :Secret
              :access-subject      :Clearance        :Unclassified
              :resource            :Classification   :Top-Secret
              :resource            :Classification   :Secret
```

```
          :resource          :Classification    :Unclassified
          :action            :action-id         :read
          :action            :action-id         :write
 </Target>

/* Rule 1 and Rule 2 apply to permissible and non-permissible “reads” */
<Rule RuleId = “Rule 1” Effect=”Permit”>
       <Target>
           /*:Attribute-Category   :Attribute ID    :Attribute Value */
                 :action             :action-id        :read
        </Target>
       <Condition>
     Function: string-greater-or-equal
           /*:Attribute-Category   :Attribute ID
                 :access-subject     :Clearance
                 :resource           :Classification
       </Condition>
  </Rule>
   <Rule RuleId = “Rule 2” Effect=”Deny”>
       <Target>
          /*:Attribute-Category    :Attribute ID    :Attribute Value */
                :action             :action-id        :read
        </Target>
       <Condition>
     Function: string-less
         /*:Attribute-Category     :Attribute ID
               :access-subject       :Clearance
               :resource             :Classification
       </Condition>
  </Rule>

     /* Rule 3 & Rule 4 apply to permissible and non-permissible
      “writes” */
   <Rule RuleId = “Rule 3” Effect=”Permit”>
       <Target>
         /*:Attribute-Category     :Attribute ID    :Attribute Value */
               :action               :action-id        :write
        </Target>
       <Condition>
     Function: string-less-or-equal
         /*:Attribute-Category     :Attribute ID
               :access-subject       :Clearance
               :resource             :Classification
       </Condition>
  </Rule>
   <Rule RuleId = “Rule 4” Effect=”Deny”>
       <Target>
        /*:Attribute-Category      :Attribute ID    :Attribute Value */
              :action               :action-id        :write
        </Target>
       <Condition>
     Function: string-greater
        /*:Attribute-Category      :Attribute ID
              :access-subject       :Clearance
              :resource             :Classification
       </Condition>
  </Rule>
```

```
</Policy>
```

Assuming that a user was assigned to Top Secret, Secret, or Unclassified, Policy3 would indeed enforce the TCSEC MAC policy, but would prevent a user from ever writing to a resource below the user's clearance level.

Now consider NGAC's specification of the same MAC policy, shown in Figure 10, where it is assumed that users (not shown) are directly assigned to Top Secret or Secret (on the left side) and objects are directly assigned to Top Secret or Secret (on the right side).

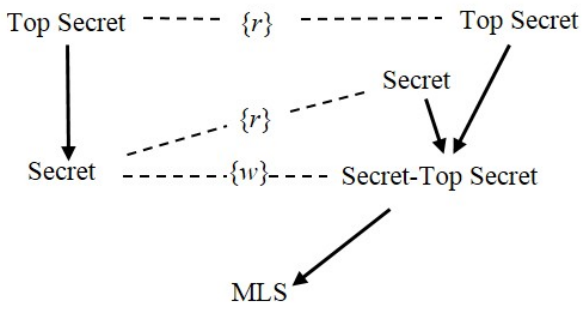

Figure 10: NGAC's Partial Expression of TCSEC MAC

The assignments and associations of the graph specify Top Secret users can read and write Secret and Top Secret objects, and Secret users can read Secret objects and write to Secret and Top Secret objects. Note that the assignments and associations alone do not prevent the leakage of data of a higher classification to a lower classification. With the following two obligations, NGAC can prevent illicit leakage of data, while allowing the user the full set of capabilities permitted by the assignments and associations. In other words, a user could read Top Secret data and write to Secret data in the same session, but through two different processes.

(1) when process *p* reads $o \rightarrow^{+} TopSecret$ **do** create p-deny(*p*, {*w*},¬*TopSecret*);
(2) when process *p* reads $o \rightarrow^{+} Secret$ **do** create p-deny(*p*, {*w*}, ¬*Secret-Top Secret*).

The first obligation specifies: when a process reads an object contained in Top Secret, deny the process from writing to any object outside the Top Secret (object attribute) container. Similarly, the second obligation specifies: when a process reads an object contained in the Secret container, deny the process from writing to any object outside the Secret-Top Secret container.

Without support for confinement, XACML is arguably incapable of enforcement of a wide variety of policies. These confinement-dependent policies include some instances of RBAC, e.g., "only doctors can read medical records", ORCON and Privacy [25], e.g., "I know who can currently read my data or personal information", or conflict of interest [21], e.g., "a user with knowledge of information within one dataset cannot read information in another dataset". Through imposing process level controls in conjunction with obligations, NGAC has shown [3] support for these and other confinement-dependent MAC controls.

Although XACML and NGAC have the ability to combine policies, their motivations are different. XACML's motivation is to resolve conflicts. That is, policies and rules may have

different Effects (Permit or Deny), which must be resolved during evaluation by selectively applying one of several combining algorithms. NGAC's motivation is to ensure the adherence of combinations of multiple policies when computing a decision (e.g., DAC and RBAC).

5.3 Operational Efficiency

While XACML and NGAC are similar in that they selectively identify and evaluate policies and conditions that pertain to a request, they differ significantly in their approach. An XACML request is a collection of attribute name-value pairs for the subject (user), resource, action, and environment that maybe translated to an XACML canonical form for PDP consumption. XACML identifies applicable policies and rules within policies by matching attributes to Targets. The entire process involves collecting attributes and matching Target conditions over all policies (trusted and untrusted access policies) and all rules in applicable policies, issuing administrative requests (for determining a chain of trust for applicable untrusted access policies). If the attributes are not sufficient for the evaluation of an applicable policy or rule, the PDP may search for additional attributes. The access process involves searching at least two data stores (PIP and PRP). The PDP evaluates each applicable rule in a policy and applies a combining algorithm in rendering a policy level decision. The process continues over all applicable policies and renders an ultimate decision by applying a combining algorithm over the evaluation results of the policies.

In response to an access request, an NGAC decision is computed using access control data in a single store represented as either as a collection of sets or more typically as a graph. In the graphical representation the policy elements user, object, user attribute, object attribute and policy class form the five types of nodes. The links between these policy elements are represented as directed edges. In addition, the links from a user attribute to an object attribute are labeled with the allowed set of operations. Unlike XACML, NGAC does not have distinct processes for identification of applicable policies and combining local decision results in reaching a final decision. Instead, all of these processes are an integral part of the path search algorithms applied on the graphical representation of NGAC access control data, to determine the existence of an appropriate privilege. If such a privilege does exist and no exceptions (prohibitions) exist, the request is granted, otherwise it is denied. Like policies and attributes, prohibitions are found through relations and not search. NGAC does not include a context handler for converting requests and decisions to and from its canonical form or for retrieving attributes. Although considered a component of its access control process, obligations do not come into play until after a decision has been rendered and data has been successfully altered or consumed.

There are potentially three phases that need to be considered in computing an access decision [29]. These include:

- loading policy from disk to memory
- Finding applicable policies, and
- Policy evaluation

The factors that impact performance for each phase are discussed for both XACML and NGAC.

5.3.1 Loading Policies

This phase involves loading policies into memory from disk (or any persistent storage medium) and converting them into an in-memory representation or structure that is suitable for efficient computation. In regard to XACML, the main components of a policy are: Target (the combination of subject, object, action and environmental attribute values) and a set of one or more rules. For XACML, this phase has been found to be the most expensive phase in access decision computation with significant increase in processing time when working with policy stores that contain over 100 policies [26]. Regardless of the number of policies, there is a fundamental need to parse and convert the external XML representation of policy to memory structures. Performance optimization strategies have been proposed to include caching frequently loaded policies (using "Policy Identifier-Loaded Policy" combination), loading just proxy policies (policies that contain only the Target) instead of the full policy contents and using an efficient parser (Java DOM-based or StAX-based) for conversion into in-memory structures. In regard to NGAC, policies, both on disk and in memory can be represented as a graph [11]. Consequently, the conversion from the representational format on disk to the in-memory structure is minimal and very efficient. In addition, for purposes of computing a decision, all information that is needed can reside in memory. In the NGAC reference implementation Version 1.6 [11] access control information is loaded into memory when the PDP is initialized, and updated when an administrative change occurs.

5.3.2 Finding Applicable Policies

In this phase, the set of memory-resident applicable policies are identified and returned after matching the request against the loaded policies. For XACML, results are available to show that the speed for matching is not correlated to the number of rules in the policy store, but only to the complexity of policy's Target [29]. Since the process of matching a request against a policy requires just the information that is in the policy Target, having proxy policies alone in the memory will meet this need. The advantage of having proxy policies, instead of full policies, is that it speeds up the process of identifying the applicable policies. After matching is done using a proxy policy to identify an applicable policy, the rest of the policy content has to be loaded from disk into memory. The overhead involved in this disk operation (loading from policy store to memory) becomes significant when the number of policies is large. Another performance optimization measure used in this phase is the caching of "Request-Applicable Policies" combinations. In regard to NGAC, the discovery of applicable policies, or more accurately stated, applicable policy elements, is an inherent aspect of its decision computation. When computing a decision NGAC only considers the graph nodes reachable from the nodes representing the user and target object in a request**.**

5.3.3 Decision Processing

The decision process involves evaluating the request against all applicable policies in the memory representation and computing the final decision. The most significant task in XACML for this phase is the evaluation of logical expressions in the various rules in the applicable policies. Secondary tasks are: Retrieval of dynamic attributes (e.g., environmental attribute such as time, IP address etc.) through the Context Handler or PIP and application of combining algorithms. As such, performance of this phase is impacted by the number of rules in the policy

store. Performance optimization is achieved by caching "Request-Access Decision" combinations and using efficient algorithms for evaluating logical expressions. The impact of caching is pronounced in application environments where access requests are similar (instead of being completely random), where the number of rules using dynamic attributes is limited and the policy store is relatively stable (no frequent addition/deletion of policies or policy sets) [29]. Although there is no formal analysis of the complexity of the XACML decision process that we are aware of, empirical studies have shown that the most significant factor impacting performance is the number of policies and/or rules which beyond a certain level present scalability issues [26]. This level is dependent on a number of factors in the deployment environment such as sensitivity and number of applications and desired granularity of access control.

In regard to NGAC, the conditions necessary for the existence of a valid privilege (right to perform an operation) can be expressed as follows in the graphical representation [30]:

A user is allowed to perform an operation on an object if and only if there exists a set of edges labeled with the desired operation such that the tail of each edge is reachable from the user, and the head of each edge is reachable from the target object, and the set of policy class nodes reachable from the set of head nodes is a superset of the set of policy class nodes reachable from the target object.

The NGAC specification describes what constitutes a valid implementation using set theoretic notation, but does not provide implementation guidance. This leaves room for multiple competing approaches and implementations with decision processing algorithms of varying efficiencies. In the method adopted in [30], an algorithm that is linear with regard to the size of the policy graph (i.e., O(m + n) where m is the number of edges and n is the number of nodes) has been developed and implemented. The linearity is not in relation to the entire access control graph, but only to the portion of the graph relevant to a particular user.

5.4 Attribute and Policy Management

XACML and NGAC both offer a delegation mechanism in support of decentralized administration of access policies. Both allow an authority (delegator) to delegate all or parts of its own authority or someone else's authority to another user (delegate). Unlike NGAC, XACML's delegation method is a partial solution. It is dependent on trusted and untrusted access policies, where trusted access policies are assumed valid, and their origin is established outside the delegation model. XACML enables policy statements to be written by multiple writers. Although XACML facilitates the independent writing, collection, and combination of policy components, XACML does not describe any normative way to coordinate the creation and modification of policy components among these writers. NGAC enables a systematic approach to the creation of administrative responsibilities. The approach begins with a single administrator that can create and delegate administrative capabilities to include further delegation authority to intermediate administrators. The process ends with users with data service, policy, and attribute management capabilities.

Although one could imagine a means of administering attributes through the use of XACML policies, in practice the creation of attribute values and subject and resource assignments to those

attributes is typically performed in different venues without any notion of coordination or governance.

Because XACML is implemented in XML, it inherits XML's benefits and drawbacks. The flexibility and expressiveness of XACML, while powerful, make the specification of policy complex and verbose [27]. Applying XACML in a heterogeneous environment requires fully specified data type and function definitions that produce a lengthy textual document, even if the actual policy rules are trivial. In general, platform-independent policies expressed in an abstract language are difficult to create and maintain by resource administrators [28]. Unlike XACML, NGAC is a relations-based standard, which avoids the syntactic and semantic complexity in defining an abstract language for expressing platform-independent policies [27]. NGAC policies are expressed in terms of configuration elements that are maintained at a centralized point and typically rendered and manipulated graphically. For example, to describe hierarchical relations between attributes, NGAC requires only the addition of links representing assignment relations between them; in XACML, relations need to be inserted in precise syntactic order.

NGAC's ability to express policies graphically aids in the management of policy expressions; administrators can "see" how the managed attributes are related to each other, as well as the policies under which the attributes are covered.

XACML does not allow policies to be modified by ordinary users. NGAC manages its access control data (policies and attributes) through a standard set of administrative operations, applying the same PEP interface and decision making function it uses for accessing its objects (resources). In other words, NGAC does not make a distinction between ordinary users and administrators; users possess varying flavors of capabilities to access resource objects and access control data objects. On one extreme, users may have only capabilities for administering a mandatory policy, and are denied the ability to provision their access to resources governed by that policy. On the other extreme, users may have total control over their own data and be responsible for setting up their own policies. Examples of the latter extreme include social networking, messaging, and calendar application capabilities.

XACML's ability to specify policies as conditions provides policy expression efficiency. Consider the NGAC expression, shown in Figure 8, of the equivalent XACML Policy1 specified in Section 3.4. NGAC expresses the policy using five association relations, while XACML uses just three rules. Note that as the number of Wards that are considered by the policy increases, so will the number of NGAC association relations, but the number of XACML rules will always remain the same. Recognize that for this policy, the number of attribute assignments is the same for XACML and NGAC. On the other hand, for some policies, the number of XACML attribute assignments can far exceed those necessary for an NGAC equivalent policy. Consider the TCSEC MAC Policy expressed using XACML rules and NGAC relations specified in Section 5.2. Note that under the NGAC configuration there is no need to directly specify policy or attributes regarding uncleared users or unclassified objects. More significantly, NGAC requires far fewer attribute assignments. For the XACML TCSEC MAC policy to work, all resources are required to be assigned to Unclassified, Secret, or Top Secret attributes. For the NGAC TCSEC MAC policy to work, only objects that are actually classified are required to be assigned to Secret or Top Secret attributes.

5.5 Administrative Review and Resource Discovery

A desired feature of access controls is review of capabilities of a user/subject and access control entries of an object/resource [18], [4]. This feature is also referred to as "before the fact audit" and resource discovery. "Before the fact audit" has been suggested by some as one of RBAC's most prominent features [5], and it includes being able to review the capabilities of a user or the consequences of assigning a user to a role. It also includes the capability for a user to discover or see accessible resources. Being able to review the access control entries of an object/resource is equally important. Who are the users/subjects that can access this object/resource and what are the consequences of assigning an object/resource to an attribute or deleting an assignment?

NGAC supports efficient algorithms for both per-user and per-object review. Biswas, Sandhu, and Krishnan report in [6] that conducting a review using enumerated policies such as that of NGAC is inherently simple. In the method identified in [30], linear bounded algorithms for the retrieval of user accessible objects are offered. However, the algorithms are not linear in relation to the entire access control database of data sets and relations, but only to the portion relevant to a particular user. Thus, empirically the algorithms execute faster than the theoretical analysis might indicate. Logical formula-based policy models, such as XACML, are not able to conduct policy review efficiently [6]. Conducting a policy review under such mechanisms is equivalent to the satisfiability problem in propositional logic, which is NP complete [6]. Determining an authorization for a subject to perform an action on a resource can only be determined by issuing a request. In other words, there exists no method of determining the authorization state without testing all possible decision outcomes.

6 Conclusion

XACML is similar to NGAC in that they both employ attributes in computing decisions, and both provide flexible, mechanism-independent representations of policy rules that may vary in granularity. However, XACML and NGAC differ significantly in their expression and management of policies, treatment of attributes, computation of decisions, and representation of requests. Many of these differences stem from the methods with which they represent policies and attributes. XACML's approach is to define policies by using logical formulas involving attribute values, while NGAC uses enumeration involving configurations of relations. As a consequence of this and other factors, XACML and NGAC have comparative advantages and disadvantages.

XACML has great expressive power and flexibility. XACML specifies policy rules in terms of attribute values that can vary in type, while NGAC treats all attributes as containers. As a consequence, there are certain policies that are expressible in XACML that cannot be accommodated by NGAC. XACML takes into consideration environmental attributes in expressing policies, and NGAC does not. Furthermore, XACML includes the notion of an obligation or advice that directs a PEP to take an action prior to or after an access request is approved or denied, which can complement and refine policies in a number of ways.

NGAC also has substantial expressive power and can express many of the same policies as XACML. However, there are some policies, including a variety of history-based policies that only NGAC can express. In contrast to NGAC, XACML does not recognize the capabilities of a process independent of the capabilities of its user. Without such features, XACML is arguably incapable of enforcing a variety of policies whose intent is to prevent "leakage" of data to unauthorized users.

Certain policies are more efficiently expressed using logical formula rather than enumeration. An NGAC configuration of an equivalent XACML policy may require the use of an explicit relation for each possible value that an attribute may take on in an XACML rule of the policy. This can have a multiplicative effect on the number of relations that need to be created using NGAC as opposed to XACML. Because XACML is implemented in XML, it inherits XML's benefits and drawbacks. The flexibility and expressiveness of XACML, while powerful, make the specification of policy complex and verbose [27]. Applying XACML in a heterogeneous environment requires fully specified data type and function definitions that produce a lengthy textual document, even if the actual policy rules are trivial. Unlike XACML, NGAC is a relations-based standard, which avoids the syntactic and semantic complexity of defining an abstract language for expressing policies [27]. NGAC policies are expressed in terms of configuration elements that are typically rendered and manipulated graphically.

In addition to policy expression, policy review is an important ABAC feature. Policy review can pertain to a number of circumstances, including the ability to identify the accessible resources of a user, the consequences of assigning a user to an attribute, or providing the capability for a user to discover or see accessible resources. Logical formula-based mechanisms such as XACML cannot conduct policy review efficiently. Conducting a policy review under logical-based mechanisms is equivalent to the satisfiability problem in propositional logic [6] (essentially this means it may become exponentially more difficult to analyze as the size of the policy increases).

In contrast, conducting a policy review using enumerated policies such as that of NGAC is relatively simple [6], [30].

Although some policies are more efficiently expressed using XACML rather than NGAC, there are not necessarily any negative consequences on NGAC's operational performance. The efficiency of evaluating a request is not directly related to the number of relations stored in the NGAC policy database. Instead, policies are evaluated only with respect to the policy elements that pertain to the user and object in the request. The two phases of XACML policy evaluation involve the costly loading of policy from disk to PDP main memory and the request evaluation. In both phases, performance is directly related to the number of policies considered. Conversely, NGAC neither loads its relations into PDP memory for each request evaluation, nor does it have to consider all policy related relations.

XACML and NGAC differ in their ability to impose policy over the creation and modification of access control data (attributes and policies). XACML (through an optional profile) provides support for decentralized administration of untrusted access policies and untrusted administrative policies. However, the approach is only a partial solution in that it is dependent on trusted and untrusted policies where trusted policies are assumed valid, and their origin is established outside the delegation model. Furthermore, the XACML delegation model does not provide a means for imposing policy over modifications to access policies and offers no direct administrative method for imposing policy over the management of its attributes. NGAC enables a systematic and policy-preserving approach to the creation of administrative roles and delegation of administrative capabilities, beginning with a single administrator and an empty set of access control data and ending with users with data service, policy, and attribute management capabilities.

In support of interoperability, XACML and NGAC replace the data service-specific, access control logic in underlying operating environments with centralized policy decision functionality. An XACML deployment consists of one or more data services, each with an operating environment-resident PEP that issues requests consisting of data service-specific operations and data types to a common PDP. NGAC can go one step further. Although an NGAC deployment could include a PEP that recognizes data service-specific operations (e.g., sending and forwarding operations for a messaging system), it does not necessarily need to do so. NGAC can accommodate a centralized PEP that can generate a set of generic, data service-agnostic operations.

Although the criteria used in this document to compare XACML and NGAC are significant factors for users to consider in choosing future ABAC deployments and for vendors considering future product offering, this set is by no means exhaustive. From a user's perspective, NGAC is new with few products available for testing and evaluation. From a vendor's perspective, the NGAC specification describes only what constitutes a valid implementation using set theoretic notation, thereby leaving room for multiple competing implementations. XACML, on the other hand, has served as a basis for a number of proprietary and open-source product offerings that cover virtually all aspects of its deployment.

Appendix A—Acronyms

Selected acronyms and abbreviations used in this document are defined below.

ABAC	Attribute Based Access Control
ACL	Access Control List
ANSI/INCITS	American National Standards Institute/International Committee for Information Technology Standards
API	Application Programming Interface
DAC	Discretionary Access Control
EPP	Event Processing Point
FISMA	Federal Information Security Modernization Act
IR	Interagency Report
IT	Information Technology
ITL	Information Technology Laboratory
MAC	Mandatory Access Control
NGAC	Next Generation Access Control
NGAC-FA	Next Generation Access Control Functional Architecture
NGAC-GOADS	Next Generation Access Control Generic Operations and Abstract Data Structures
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
OMB	Office of Management and Budget
ORCON	Originator Controlled
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PM	Policy Machine
PRP	Policy Retrieval Point
RAP	Resource Access Point
RBAC	Role-Based Access Control
RS	Resource Server
SAML	Security Assertion Markup Language
SOA	Service Oriented Architecture
SP	Special Publication
TCSEC	Trusted Computer Security Evaluation Criteria
XACML	Extensible Access Control Markup Language
XML	Extensible Markup Language

Appendix B—References

[1] OASIS, *eXtensible Access Control Markup Language (XACML) Version 3.0*, January 22, 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf [accessed 8/29/16]

[2] V.C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*, National Institute of Standards and Technology (NIST) Special Publication (SP) 800-162, January 2014. http://dx.doi.org/10.6028/NIST.SP.800-162

[3] D.F. Ferraiolo, V. Atluria, and S.I. Gavrila, "The Policy Machine: A Novel Architecture and Framework for Access Control Policy Specification and Enforcement," *Journal of Systems Architecture*, vol. 57, no. 4, pp. 412-424, April 2011. http://dx.doi.org/10.1016/j.sysarc.2010.04.005

[4] V.C. Hu, D.F. Ferraiolo, and D.R. Kuhn, *Assessment of Access Control Systems*, National Institute of Standards and Technology (NIST) Interagency Report (IR) 7316, September 2006. http://dx.doi.org/10.6028/NIST.IR.7316

[5] A.C. O'Connor and R.J. Loomis, *2010 Economic Analysis of Role-Based Access Control*, RTI Number 0211876, Research Triangle Institute, December 2010. http://www.nist.gov/tpo/upload/No-10-Role-Based-Access-Control-2010.pdf [accessed 8/29/16]

[6] P. Biswas, R. Sandhu, R. Krishnan, "Label-Based Access Control: An ABAC Model with Enumerated Authorization Policy," in *ABAC '16: Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, ACM, 2016, pp. 1-12. http://dx.doi.org/10.1145/2875491.2875498

[7] X. Jin, R. Krishnan, and R. Sandhu, "A Unified Attribute-Based Access Control Model Covering DAC, MAC, and RBAC," *Data and Applications Security and Privacy XXVI: Proceedings of the 26th Annual IFIP WG 11.3 Conference, DBSEC 2012, Paris, France, July 11-13, 2012*, Lecture Notes in Computer Science 7371, Springer, 2012, pp. 41-55. http://dx.doi.org/10.1007/978-3-642-31540-4_4

[8] D. Servos and S.L. Osborn, "HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control," *Foundations and Practice of Security: 7th International Symposium, FPS 2014, Montreal, Quebec, Canada, November 3-5, 2014, Revised Selected Papers*, Lecture Notes in Computer Science 8930, Springer, 2014, pp. 187-204. http://dx.doi.org/10.1007/978-3-319-17040-4_12

[9] D. Ferraiolo, S. Gavrila, and W. Jansen, *Policy Machine: Features, Architecture, and Specification*, National Institute of Standards and Technology (NIST) Internal Report (IR) 7987 Revision 1, October 2015. http://dx.doi.org/10.6028/NIST.IR.7987r1

[10] D. Ferraiolo and S. Gavrila, "Method and system for the specification and enforcement of arbitrary attribute-based access control policies", U.S. Patent App. 12/366,855, 2009. https://scholar.google.com/citations?view_op=view_citation&hl=en&user=_6b33c0AAAAAJ&citation_for_view=_6b33c0AAAAAJ:kNdYIx-mwKoC [accessed 8/29/16]

[11] NIST Policy Machine Versions 1.5 and 1.6 - Harmonia [Website], https://github.com/PM-Master [accessed 9/26/16]

[12] American National Standards Institute, *Information technology - Next Generation Access Control - Functional Architecture (NGAC-FA)*, INCITS 499-2013, American National Standard for Information Technology, March 2013.

[13] American National Standards Institute, *Information technology – Next Generation Access Control – Generic Operations and Data Structures (GOADS)*, INCITS 526-2016, American National Standard for Information Technology, January 2016.

[14] D.F. Ferraiolo, S.I. Gavrila, V.C. Hu, and D.R. Kuhn, "Composing and Combining Policies Under the Policy Machine," *SACMAT '05: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden,* ACM, 2005, pp. 11-20. http://dx.doi.org/10.1145/1063979.1063982

[15] D. Ferraiolo, S. Gavrila, and W. Jansen, "On the Unification of Access Control and Data Services," in *Proceedings of the 2014 IEEE 15th International Conference of Information Reuse and Integration*, IEEE, 2014, pp. 450 – 457. http://dx.doi.org/10.1109/IRI.2014.7051924

[16] Medidata Policy Machine version 1.1.0 [Website], https://github.com/mdsol/the_policy_machine [accessed 8/29/16]

[17] OASIS, *XACML v3.0 Administration and Delegation Profile Version 1.0*, Committee Specification Draft 04 / Public Review Draft 02, November 13, 2014. http://docs.oasis-open.org/xacml/3.0/administration/v1.0/csprd02/xacml-3.0-administration-v1.0-csprd02.doc [accessed 8/29/16]

[18] National Computer Security Center, *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003, Version-1, Fort George G. Meade, Maryland, USA, September 30, 1987, 29 pp. http://csrc.nist.gov/publications/secpubs/rainbow/tg003.txt [accessed 8/29/16]

[19] American National Standards Institute, *Information technology – Role-Based Access Control (RBAC)*, INCITS 359-2004, American National Standard for Information Technology, 2004.

[20] OASIS, *XACML Profile for Role Based Access Control (RBAC)*, Committee Draft 01, February 13, 2004, https://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf [accessed 8/29/16]

[21] D.F.C. Brewer and M.J. Nash, "The Chinese Wall Security Policy," in *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, Oakland, California, United States, May 1-3, 1989, pp. 206-214. http://dx.doi.org/10.1109/SECPRI.1989.36295

[22] R. Simon and M. Zurko, "Separation of duty in role-based environments," in *Proceedings of the 10th Computer Security Foundations Workshop, Rockport, Massachusetts, June 10-12, 1997*, IEEE Computer Society, 1997, pp. 183-194. http://dx.doi.org/10.1109/CSFW.1997.596811

[23] D. Bell and L. La Padula, "Secure computer system: unified exposition and MULTICS interpretation," Report ESD-TR-75-306, The MITRE Corporation, Bedford, Massachusetts, March 1976, 129 pp. http://www.dtic.mil/docs/citations/ADA023588 [accessed 8/29/16]

[24] DoD Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," DoD Directive 5200.28, December 1985.

[25] R. Graubart, "On the need for a third form of access control," in *Proceedings of the 12th National Computer Security Conference, Baltimore, Maryland, October 10-13, 1989*, National Institute of Standards and Technology / National Computer Security Center, 1989, pp. 296-304. http://csrc.nist.gov/publications/history/nissc/1989-12th-NCSC-proceedings.pdf [accessed 8/29/16]

[26] F. Turkmen and B. Crispo, "Performance Evaluation of XACML PDP Implementations," in *Proceedings of the 2008 ACM Workshop on Secure Web Services*, ACM, 2008, pp. 37-44. http://dx.doi.org/10.1145/1456592.1456499

[27] V.C. Hu, D.F. Ferraiolo, and K. Scarfone, "Access Control Policy Combinations for the Grid Using the Policy Machine," in *Proceedings. Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, IEEE Computer Society, 2007, pp. 225-232. http://dx.doi.org/10.1109/CCGRID.2007.15

[28] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shah, "First Experience Using XACML for Access Control in Distributed Systems," in *Proceedings of the 2003 ACM Workshop on XML Security (XMLSEC '03), Fairfax, Virginia*, ACM, 2003, pp. 25-37. http://dx.doi.org/10.1145/968559.968563

[29] Ö.M. Ilhan D. Thatmann, and A. Küpper, "A Performance Analysis of the XACML Decision Process and the Impact of Caching," in *Proceedings of the 2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS 2015)*, IEEE, 2015, pp. 216 -223. http://dx.doi.org/10.1109/SITIS.2015.83

[30] P. Mell, J. Shook, S. Gavrila, Restricting Insider Access through Efficient Implementation of Multi-Policy Access Control Systems. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*. Vienna, Austria, October 24-26, 2016.

Appendix C—XACML 3.0 Encoding of Medical Records Access Policy

```
/* This policy pertains to Medical Record (Read or Write) Access
by users with role "Doctor" or "Intern".

Rule 1 denies access if the WardAssignment of the doctor or intern
does not match the WardLocation of the patient.
Rule 2 denies write access to intern unconditionally.
Rule 3 permits access if the subject is a doctor and the PatientStatus
is Critical without any other conditions. */


<Policy PolicyId="Medical-Record-Access-by-Doctors-and-Interns"
   RuleCombiningAlgId = "permit-overrides">

   <Target>
      /* Policy Target covers all subjects with Doctor or Intern role,
      resources with medical-records as Resource-id,
      and actions either read or write */

      <AnyOf>
        <AllOf>
            /* Specifying the subject match -
             subjects with role-id equal to Doctor or Intern */
            <Match MatchId="string-equal">
               /* Subject role = Doctor */
               <AttributeValue>
                  Doctor
               </AttributeValue>
               <AttributeDesignator Category="access-subject"
                AttributeId="role-id"/>
            </Match>
        </AllOf>

        <AllOf>
            /* Specifying the subject match - subjects with role-id equal to
             Doctor */
            <Match MatchId="string-equal">
               /* Subject role = Intern */
               <AttributeValue>
                  Intern
               </AttributeValue>
               <AttributeDesignator Category="access-subject"
                AttributeId="role-id"/>
            </Match>
         </AllOf>
      </AnyOf>


      <AnyOf>
         <AllOf>
            /* Specifying the resource match -
             resource with resource-id equal to medical records */
            <Match MatchId="string-equal">
               <AttributeValue>
                  medical-records
```

```
            </AttributeValue>
            <AttributeDesignator Category="resource"
             AttributeId="resource-id"/>
         </Match>
      </AllOf>
   </AnyOf>

   <AnyOf>
      /* Specifying action match action with either read or write value */
      <AllOf>
         /* read action */
         <Match MatchId="string-equal">
            <AttributeValue>
               read
            </AttributeValue>
            <AttributeDesignator Category="action"
             AttributeId="action-id"/>
         </Match>
      </AllOf>

      <AllOf>
         /* write action */
         <Match MatchId="string-equal">
            <AttributeValue>
               write
            </AttributeValue>
            <AttributeDesignator Category="action"
             AttributeId="action-id"/>
         </Match>
      </AllOf>
   </AnyOf>
</Target>

<Rule RuleId="Rule 1"
   Effect="Deny">
   /* denial of access to medical record for all subjects if the patient
   is not in the same ward to which the doctor or intern is assigned */

   <Condition>
      <Apply FunctionId="string-not-equal">
         <Apply FunctionId="string-one-and-only">
            <AtributeDesignator Category="access-subject"
             AttributeId="WardAssignment">
         </Apply>

         <Apply FunctionId="string-one-and-only">
            <AtributeSelector Category="resource"
             Path="medical-records/patient/WardLocation/text( )"/>
         </Apply>
      </Apply>
   </Condition>
</Rule>



<Rule RuleId="Rule 2"  Effect="Deny">
   /* unconditional denial of write access to Interns */
```

```
      <Condition>
         <Apply FunctionId="string-equal">
            <Apply FunctionId="string-one-and-only">
               <AttributeValue>
                  Intern
               </AttributeValue>
               <AttributeDesignator Category="access-subject"
                AttributeId="role-id"/>
            </Apply>

            <Apply FunctionId="string-one-and-only">
               <AttributeValue>
                  write
               </AttributeValue>
               <AtributeDesignator Category="action" AttributeId="action-id">
            </Apply>
         </Apply>
      </Condition>
   </Rule>



   <Rule RuleId="Rule 3" Effect="Permit">
      /* unconditional access to medical records for doctor if the patient
       Status is critical irrespective of the location of the patient */

      <Condition>
         <Apply FunctionId="and">
            /* combines subject role value and patient status value */
            <Apply FunctionId="string-one-and-only">
               /* retrieves the subject role */
               <AttributeValue>
                  doctor
               </AttributeValue>
               <AttributeDesignator Category="access-subject"
                AttributeId="role-id"/>
            </Apply>

            <Apply FunctionId="string-equal">
              /*looks for medical records where patient status is critical */
              <Apply FunctionId="string-one-and-only">
                  <AttributeSelector Category="resource"
                  Path="medical-records/patient/PatientStatus/text( )"/>
               </Apply>
               <AttributeValue>
                  Critical
               </AttributeValue>
            </Apply>
         </Apply>
      </Condition>
   </Rule>
</Policy>
```