InterNational Committee for Information Technology Standards (INCITS)
Secretariat: Information Technology Industry Council (ITI)
1101 K Street NW, Suite 610, Washington, DC 20005
www.INCITS.org

eb-2017-00576

Document Date:	11/15/2017
To:	INCITS Members
Reply To:	Rachel Porter
Subject:	Public Review and Comments Register for the Approval of:

INCITS 525: Information technology - Next Generation Access Control - Implementation Requirements, Protocols and API Definitions (NGAC-IRPADS)

Due Date: **The public review is from December 1, 2017 to January 30, 2018**

Action: The InterNational Committee for Information Technology Standards (INCITS) announces that the subject-referenced document(s) is being circulated for a 60-day public review and comment period. Comments received during this period will be considered and answered. Commenters who have objections/suggestions to this document should so indicate and include their reasons.

All comments should be forwarded not later than the date noted above to the following address:

INCITS Secretariat/ITI
1101 K Street NW - Suite 610
Washington DC 20005-3922
Email: comments@standards.incits.org (preferred)

This public review also serves as a call for patents and any other pertinent issues (copyrights, trademarks). Correspondence regarding intellectual property rights may be emailed to the INCITS Secretariat at patents@itic.org.

Working Draft
Standard

DPANS
INCITS 525

Revision 0.75
26 October 2017

Information technology - Next Generation Access Control - Implementation Requirements, Protocols and API Definitions (NGAC-IRPAD)

This is an internal working document of CS1, a Technical Committee of Accredited Standards Committee INCITS (InterNational Committee for Information Technology Standards). As such this is not a completed standard and has not been approved. The contents may be modified by the CS1 Technical Committee. The contents are actively being modified by CS1. This document is made available for review and comment only.

CS1 Technical Editor:

Wayne Jansen
Bayview Behavioral Consulting
1574 Gulf Rd., #237
Point Roberts, WA 98281
USA
Telephone: (360) 306-5263
Email: jansen@computer.org

Reference number
INCITS 525:201x

Points of Contact

InterNational Committee for Information Technology Standards (INCITS) CS1 Technical Committee

CS1 Chair
Eric Hibbard

Telephone:
Email: Eric.Hibbard@hitachivantara.com

CS1 Vice-Chair
Sal Francomacaro
NIST
Stop 8930
Gaithersburg, MD 20899-8930
USA

Telephone: (301) 975-6414
Email: salvatore.francomacaro@nist.gov

CS1 Web Site: http://www.incits.org/committees/cs1

INCITS Secretariat
c/o Information Technology Industry Council
1101 K Street NW
Suite 610
Washington, DC 20005
USA

Telephone: (202) 737-8888
Web site: http://www.incits.org
Email: incits@itic.org

Revision Information

Version 0.70 (20 September 2017)
Revised draft of standard.
Version 0.75 (26 October 2017)
Revised draft addressing comments from INCITS management review of version 0.70

Draft

American National Standards
for Information Systems

Next Generation Access Control - Implementation Requirements, Protocols and API Definitions (NGAC-IRPAD)

Secretariat

InterNational Committee for Information Technology Standards

Approved mm.dd.yy

American National Standards Institute, Inc.

Abstract

Next Generation Access Control (NGAC) is a fundamental reworking of traditional access control to meet the needs of the modern, distributed, interconnected enterprise. NGAC is based on a flexible infrastructure that can provide access control services for a number of different types of resources, accessed by a number of different types of applications and users. The design is scalable, capable of supporting different types of policies simultaneously, and manageable in the face of changing technology, organizational restructuring, and increasing data volumes. This standard specifies key implementation aspects of the NGAC framework that allow functional entities within the architecture to operate in a correct, effective, and accordant manner.

Draft

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that effort be made towards their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give interpretation on any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken periodically to reaffirm, revise, or withdraw this standard. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

Caution: The developers of this standard have requested that holders of patents that may be required for the implementation of the standard disclose such patents to the publisher. However, neither the developers nor the publisher have undertaken a patent search in order to identify which, if any, patents may apply to this standard. As of the date of publication of this standard and following calls for the identification of patents that may be required for the implementation of the standard,

No further patent search is conducted by the developer or publisher in respect to any standard it processes. No representation is made or implied that licenses are not required to avoid infringement in the use of this standard.

Published by
American National Standards Institute
11 W. 42nd Street, New York, New York 10036

Printed in the United States of America

Table of Contents

List of Figures

Figure **Page**

Foreword

(This foreword is not part of American National Standard INCITS.***:201x.)

Technical Committee CS1 of Accredited Standards Committee INCITS developed this standard during 2011-201x. The standards approval process started in 201x.

Next Generation Access Control (NGAC) is a fundamental reworking of traditional access control into a form suited to the needs of the modern, distributed, interconnected enterprise. NGAC is based on a flexible infrastructure that can provide access control services for a number of different types of resources, accessed by a number of different types of applications and users. The NGAC infrastructure design is scalable, supports policies of different types simultaneously, and remains tractable in the face of changing technology, organizational restructuring, and increasing data volumes. Functional components of the reference architecture can be supported by products from different manufacturers.

NGAC diverges from traditional approaches to access control in that it defines a generic architecture that is separate from any particular policy or type of policy. NGAC is not an extension or adaption of any existing access control model, but rather a redefinition of access control in terms of a fundamental and reusable set of data abstractions and functions. NGAC provides a unifying framework capable of supporting current access control approaches as well as novel types of policy that have been conceived yet never implemented due to the lack of a suitable means of expression and enforcement.

NGAC follows an attribute-based construction in which characteristics or properties are used to control access to resources and to describe and manage policy. NGAC accommodates combinations of different policies and can support several types of policies concurrently in a manner that is both deterministic and manageable. NGAC is also suitable for applications in which some information is stored locally and some is stored in a grid or cloud, since different policies can be asserted in each context. Even more generally, NGAC supports situations in which policy determined by a central organization is able to operate concurrently with a local, specific, and more ad hoc policy.

Through its support of access control policies, NGAC is also able to protect data services, such as e-mail, workflow, and records management. Support for data services is effected through the use of NGAC access control information to mediate data service operations.

The family of NGAC standards specifies the architecture, functions, operations, and interfaces at a level of detail necessary to ensure their realization in different types of implementation environments at a range of scalability levels. This standard specifies, in detail, key interfaces of the reference architecture and provides guidance and recommendations regarding aspects of implementation.

This standard contains the following items:

a) detailed specifications of the interfaces between key entities of the NGAC functional architecture;

b) implementation considerations and requirements for the functional entities of the architecture; and

c) operational and security considerations and requirements for the interfaces between key functional entities.

Requests for interpretation, suggestions for improvement and addenda, or defect reports are welcome. They should be sent to the INCITS Secretariat, InterNational Committee for Information Technology Standards, Information Technology Institute, 1101 K Street NW, Suite 610, Washington, DC 20005.

Users of this standard are encouraged to determine if there are standards in development or new versions of this standard that may extend or clarify technical information contained herein.

This standard was processed and approved for submittal to ANSI by the InterNational Committee for Information Technology Standards (INCITS). Committee approval of the standard does not necessarily imply that all committee members voted for approval. At the time it approved this standard, INCITS had the following members:

Organization Represented **Name of Representative**

Editor's Note 1: <<Insert INCITS member list>>

Technical Committee CS1 on Cyber Security, which reviewed this standard, had the following members:

Dan Benigni, Chair
Sal Francomacaro, Vice-Chair
Eric Hibbard, International Representative

Organization Represented	**Name of Representative**

CS1 Ad Hoc on Next Generation Access Control, which developed and reviewed this standard, had the following members:

Sal Francomacaro, Chair
Wayne Jansen, Editor

Organization Represented	Name of Representative
Hitachi Data Systems	Eric Hibbard
NIST	David Ferraiolo
	Sal Francomacaro
	Serban Gavrila
	Wayne Jansen
VHA CIO	Mike Davis
	Richard Grow

Introduction

Next Generation Access Control (NGAC) is a fundamental reworking of traditional access controls to meet the needs of the modern, distributed, and interconnected enterprise. NGAC provides a unifying framework capable of supporting both current and novel types of access control policies simultaneously. NGAC defines access control in terms of a fundamental and reusable set of data abstractions and functions, following an attribute-based access control model in which authorization are defined in terms of attributes. Security-relevant properties of users, processes and objects (e.g., role, sensitivity, affiliation and class) can be expressed as attributes.

The family of NGAC standards specifies the architecture, functions, operations, and interfaces necessary to enable conforming implementations to interact in an effective manner. This standard, NGAC-IRPAD, explicates features that are required of the NGAC framework and its functional entities as well as implementation considerations that may accompany them.

NGAC-IRPAD is divided into the following clauses and annexes:

Clause 1 defines the scope of this standard.

Clause 2 enumerates the normative references that apply to this standard.

Clause 3 defines the definitions, symbols, abbreviations, and notation used in this standard.

Clause 4 identifies the critical interfaces between key entities of the functional architecture, and specifies them in detail.

Clause 5 defines the implementation requirements for the functional entities that comprise the architecture and also for the interfaces between key functional entities.

Clause 6 discusses other aspects of the implementation and use of the NGAC framework.

Annex A describes algorithms related to various computations involving the policy.

AMERICAN NATIONAL STANDARD **BSR INCITS.***:201x**

**American National Standard
for Information Technology -**

Next Generation Access Control - Implementation Requirements, Protocols and API Definitions (NGAC-IRPAD)

1 Scope

NGAC follows an attribute-based construction in which characteristics or properties are used to describe and manage policy and to control access to resources. The family of NGAC standards specifies the architecture, functions, operations, and interfaces necessary to ensure their realization in different types of implementation environments at a range of scalability levels. This standard contains the information needed to realize the NGAC Functional Architecture and obtain the requisite level of cohesion and functionality to operate correctly and effectively at the system level.

2 Normative References

The following ANSI and ISO standards contain provisions that, by reference in the text, constitute provisions of this standard. At the time of publication, the editions listed were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

Copies of the documents may be obtained from ANSI, an ISO member organization:

Approved ANSI standards;
approved international and regional standards (ISO and IEC); and
approved foreign standards (including JIS and DIN).

For further information, contact the ANSI Customer Service Department:

Phone: +1 212-642-4900
Fax: +1 212-302-1286
Web: http://www.ansi.org
E-mail: ansionline@ansi.org

or the InterNational Committee for Information Technology Standards (INCITS):

Phone 202-737-8888
Web: http://www.incits.org
E-mail: incits@itic.org

The following are approved references that pertain to this standard:

ACF: ISO/IEC 10181-3:1996, *Open Systems Interconnection – Security frameworks for open systems: Access control framework*

FAM: ISO/IEC 29146:2016, Information technology – *Security techniques – A framework for access management*

NGAC-FA: INCITS 499-2017, Information technology – *Next Generation Access Control – Functional Architecture (NGAC-FA)*

NGAC-GOADS: INCITS 526-2016, Information technology – *Next Generation Access Control – Generic Operations And Data Structures (NGAC-GOADS)*

ZNOT: ISO/IEC 13568:2002, Information technology – *Z formal specification notation – Syntax, type system and semantics*

Note that the status of the referenced American National Standards and ISO standards may have changed since the time of publication. For information about the current status of a document, or regarding availability, contact the relevant standards body.

3 Definitions, Symbols, Abbreviations, and Conventions

3.1 Definitions

For the purposes of this document, the terms and definitions in NGAC-FA and NGAC-GOADS apply, as do the following terms and definitions.

Client Application: A program that accesses protected resources or manipulates policy on behalf of a user. A process is a running instance of a client application.

3.2 Symbols and acronyms

Symbol / Acronym	Meaning
ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
CA	Client Application
DAG	Directed Acyclic Graph
EPP	Event Processing Point
NGAC	Next Generation Access Control
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RAP	Resource Access Point

3.3 Keywords

Invalid: A keyword used to describe an illegal or unsupported bit, byte, word, field or code value. Receipt of an invalid bit, byte, word, field or code value shall be reported as an error.

Mandatory: A keyword indicating an item that is required to be implemented as defined in this standard to claim compliance with this standard.

May: A keyword that indicates flexibility of choice with no implied preference.

Optional: A keyword that describes features that are not required to be implemented by this standard. However, if any optional feature defined by this standard is implemented, it shall be implemented as defined in this standard.

Reserved: A keyword referring to bits, bytes, words, fields and code values that are set aside for future standardization. Their use and interpretation may be specified by future extensions to this or other standards. A reserved bit, byte, word or field shall be set to zero, or in accordance with a future extension to this standard. Recipients are not required to check reserved bits, bytes, words or fields for zero values. Receipt of reserved code values in defined fields shall be reported as an error.

Shall: A keyword indicating a mandatory requirement. Designers are required to implement all such requirements to ensure conformance with this standard.

Should: A keyword indicating flexibility of choice with a preferred alternative; equivalent to the phrase “it is recommended”.

3.4 Conventions

Certain words and terms used in this American National Standard have a specific meaning beyond the common English meaning. These words and terms are defined either in clause 3 or in the text where they first appear.

Numbers that are not immediately followed by lower-case b or h are decimal values.

Numbers immediately followed by lower-case b (xxb) are binary values.

Numbers immediately followed by lower-case h (xxh) are hexadecimal values.

Hexadecimal digits that are alphabetic characters are upper case (i.e., ABCDEF, not abcdef).

Hexadecimal numbers may be separated into groups of four digits by spaces. If the number is not a multiple of four digits, the first group may have fewer than four digits (e.g., AB CDEF 1234 5678h)

An alphabetic list (e.g., a, b, c or A, B, C) of items indicate the items in the list are unordered.

A numeric list (e.g., 1, 2, 3) of items indicate the items in the list are ordered (i.e., item 1 shall occur or complete before item 2).

In the event of conflicting information the precedence for requirements defined in this standard is

1) text,
2) tables, then
3) figures.

4 Interface Specifications

4.1 Background

The implementation requirements in this standard are intended to support the exchange of access control data between entities of the functional architecture. Each functional entity of the NGAC framework exposes a set of Application Programming Interfaces (APIs), which are identified and summarized in NGAC-FA. Those interfaces are illustrated in Figure 1. A black line depicts an interface supported by a functional entity. An arrow through an interface depicts the direction of invocation of the exposed interface with the interface provider at the arrow head and the interface consumer at the tail. The NGAC framework does not specify how functional entities should be grouped or packaged together in an implementation.

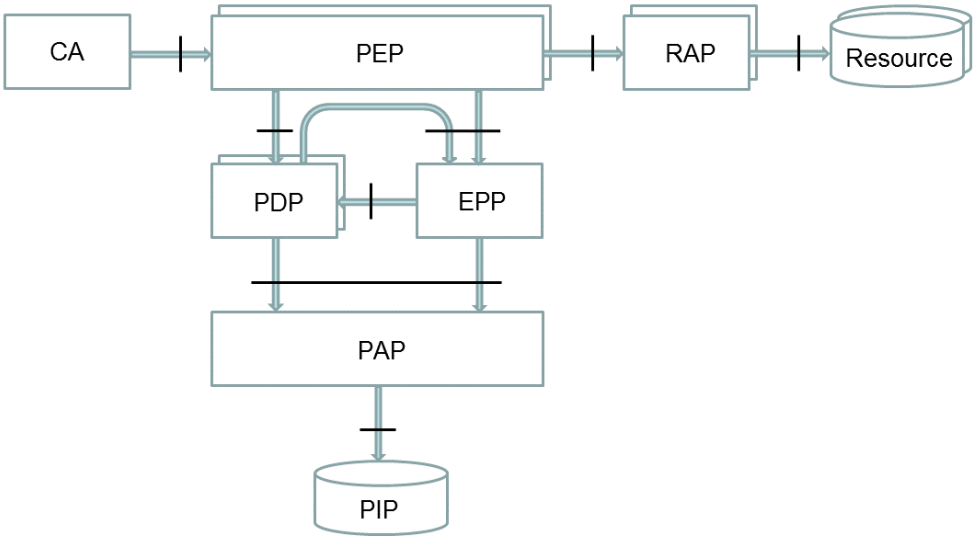

Figure 1: Interfaces Between Functional Entities

The following interfaces are depicted:

a) A Policy Enforcement Point (PEP) exposes a set of interfaces for use by a Client Application (CA) to access resources and policy information;
b) A Policy Decision Point (PDP) exposes a standard set of interfaces for use by a PEP to obtain a decision regarding an access, and by the Event Processing Point (EPP) to obtain a decision regarding the response of a matched obligation and have it carried out if deemed valid;
c) The EPP exposes a standard set of interfaces for use by a PEP or a PDP to indicate the occurrence of an event and to communicate its context;
d) The Policy Access Point (PAP) exposes a standard set of interfaces for use by the EPP to match an event context with defined obligations and to resolve the event response of a defined obligation, and by a PDP to request information on which to base access decisions and to manage the contents of the Policy Information Point (PIP);
e) The PIP exposes a set of interfaces for use by the PAP to search and manipulate the basic elements, containers and relations that are persisted by the PIP; and
f) A Resource Access Point (RAP) exposes a set of interfaces for use by a PEP to access protected resources and transfer data as necessary to and from those resources.

The interfaces between key functional entities, namely those enumerated in items (b), (c) and (d) above, are specified in greater detail in NGAC-IRPAD. The remaining interfaces are not elaborated further within the NGAC family of standards due to variability in the type of resources protected by the policy, the

translation of abstract operations performed on objects to concrete operations on the resources they represent, and the potential for diversity in the data model and services offered by various types of data stores.

An interface is a boundary across which two entities meet or communicate with each other. Documenting an interface consists of identifying it, assigning a name and specifying its syntactic and semantic information. The syntactic information consists of the signature of the methods that constitute the interface. The signature specifies the name of the method and its parameters. Parameters are defined by stating their order and type. The semantic information is in the form of a description of the behavior of each method. Because an entity may both consume and produce information through an interface it provides, the order and type of the values returned by each method of the interface are also specified.

4.2 Interface descriptions

Interfaces are described at an abstract level that specifies the information conveyed via the interface and indicates the behavior the functional entity carries out when the interface is invoked. The mathematical notation used to describe an interface corresponds to the subset of the Z formal specification notation used in NGAC-GOADS. This approach allows conformant functional entities to be developed free from constraints on the implementation environment of an entity such as the programming language or operating system features. In addition, the specifications of basic elements, containers, relations and other entities formally defined in NGAC-GOADS also apply to this standard.

Besides the interfaces specified in this standard, an implementation may support additional interfaces for functional entities to provide supplemental services. An implementation may also build upon a defined interface, extending and adjusting it to meet the design of the implementation and the practicalities of the computational environment.

The interface specifications defined herein apply only to exposed interfaces. Implementations in which two or more functional entities have been amalgamated and are treated as a unit may use appropriate internal interfaces that are not exposed and differ from the interface specifications in IRPAD. However, it is recommended that the IRPAD interface specifications be followed to the greatest extent possible in such instances.

4.3 PDP interfaces

4.3.1 Overview

Two distinct interfaces are supported by a PDP:

a) The adjudication of access requests received from a PEP; and
b) The evaluation and processing of event responses received from the EPP.

The Resource and Administrative Access Information Flows in NGAC-FA describe the behavior of a PDP for the former, and the Event Context Information Flow therein describes a PDP’s behavior for the latter.

4.3.2 Access request adjudication

The access request adjudication interface is for the sole use of PEPs. Two similar but distinct types of PEP-issued access requests are adjudicated by a PDP through this interface:

a) resource access; and
b) administration access.

Although the behavior of a PDP in each case is different, the access request construct, AREQ, is the same for both types of access (see NGAC-GOADS). The methods that constitute the access request adjudication interface are specified in Table 1.

Table 1: Access Request Adjudication Interface

Method	Description
AdjudicateResAccess (request: AREQ): RES_RESP	Check the authorization of the user/process to perform a resource access and return the access decision in the response. If the decision is to grant access to an object, return the locator for the associated resource along with the decision in the response.
AdjudicateAdmAccess (request: AREQ): ADM_RESP	Check the authorization of the user/process to perform an administration access and return the access decision. If the decision is to grant access, perform the access and return the access decision and the result of administrative access in the response.

Access requests should be well formed with valid identifiers for the referenced policy entities. An adjudication response to a resource access contains the decision rendered by the PDP and the locator for the resource denoted by the object in the access request. The decision may be positive, negative, or otherwise indicate an error condition. The contents and semantics of the locator can differ depending on the computational environment of the implementation are not defined in further detail in this standard.

RES_RESP ⊆ DECISION × LOC

DECISION = {Yes, No, Error}

LOC = seq_1 {00000000b, …, 11111111b}

An adjudication response to an administrative access conveys the decision rendered by the PDP. If a positive decision was rendered, the result of the administrative action taken and the identifier of a basic element, container, relation or other policy entity created by the administrative action are also conveyed. An administrative action is not attempted if there is insufficient authorization, and could fail when attempted due to a processing error. Otherwise, the action is reported as successful. The PDP should raise a system alert for any critical, actionable problems encountered that could affect the integrity and coherence of the policy. The details of any problems encountered should be entered into the audit log.

ADM_RESP ⊆ DECISION × RESULT × ID

RESULT = {Success, Failure}

4.3.3 Event response evaluation

For each obligation that matches an event context, the EPP communicates with a PDP to evaluate whether the authorization held by the obligation's creator for the actions of the event response is adequate and process the event response accordingly. The PDP utilizes the interface afforded by the PAP to carry out the event response.

The method defining this interface is specified in Table 2.

Table 2: Event Response Evaluation Interface

Method	Description
EvaluateResponse (evresp: EVNT_RESP): seq_1 ADM_RESP	Check the authorization of the user that defined the obligation and its event response. If the authorization is sufficient, carry out the administrative actions of the response and return the results.

The event response, EVNT_RESP, conveys the identifier of the user responsible for defining the event response and the actions to be taken. The actions are represented as a non-empty sequence of paired administrative operations and conformant operands.

EVNT_RESP $\subseteq$ U $\times$ seq_1 (AOP $\times$ seq_1 Arg), where Arg = { x | x $\in$ PE $\vee$ x $\in$ 2^{PE} $\vee$ x $\in$ 2_1^{AR}}

4.4 EPP interface

4.4.1 Overview

The EPP interface accepts and processes event contexts that are issued from both PEPs and PDPs, which pertain to successfully completed accesses to protected resources or to policy information persisted at the PIP, respectively.

The Event Context Information Flow in NGAC-FA describes the behavior of the EPP, which is the same regardless of the source of issue. The single interface supports both types of functional entities.

4.4.2 Event context processing

The EPP uses the information from an event context to process obligations. It matches the event context with the event pattern of each obligation persisted in the PIP, and carries out the event response of each matched obligation as a single atomic action. Table 3 specifies the method that defines this interface.

Table 3: Event Context Processing Interface

Method	Description
ProcessEventContext (context: EC): EC_RESP	If the event context is not well formed, return a negative response. Otherwise, match the event context against the event pattern of each defined obligation. For each matched pattern, process the corresponding event response using the PDP interface defined for this purpose. If the event responses of all matched obligation patterns are processed without complications, return a positive result; otherwise, return a negative result.

The event context, EC, includes identifier for the user of the process, the process, and the access operation and operands, which describe the associated event. Other additional information employed in the EPP's matching process, such as the containers of a policy element targeted by the access (e.g., the object attributes of an object that was deleted), are also conveyed via the event context. The additional information conveyed is dependent upon the type of event that occurred and may be attuned to the language grammar and semantics for an obligation's event pattern and response.

$EC \subseteq U \times P \times OP \times seq_1\ ID \times seq\ ID$

For each event context it receives, the EPP returns a response, EC_RESP, to the functional entity that initiated the request. If the event context is malformed or irregular such that it precludes the resolution of event patterns necessary for matching, no processing occurs and the result is reported as a failure. When processing of all matched obligations has completed, if no problems were encountered, the result is reported as a success. If any problems were encountered, the result is reported as a failure. The EPP should raise a system alert for any critical, actionable problems encountered, such as resolution errors, insufficient authorization, response incongruity or service errors, which could affect the integrity and coherence of the policy. The details of any problems encountered should be entered into the audit log.

$EC\_RESP \subseteq U \times P \times RESULT$

The EPP may be assigned a PDP for its exclusive use in processing the event responses of matched obligations to avoid contention with PEPs performing access request adjudication with the same PDP as the EPP.

4.5 PAP interfaces

4.5.1 Overview

The following two distinct interfaces are supported by the PAP:

a) The processing of inquiries concerning the defined policy, which are needed by a PDP to form an access decision, and by the EPP to act upon an event context with regard to any relevant defined obligation; and
b) The processing of directives affecting the defined policy, which are needed by a PDP to carry out successfully adjudicated administrative access requests and the event responses of matched obligations.

The PAP interfaces are designed to preserve the properties of the policy model. Policy inquiries shall not affect any policy entity, and policy modifications shall not have any side effect on the policy persisted in the PIP, other than the outcome defined for the interface. The naming conventions used for PAP interfaces prepends each method with a prefix, a letter followed by a dash, to indicate the type of task performed by the method. The prefixes used are as follows: C - create, D - delete, G - get and Q - question.

4.5.2 Policy inquiry

The PAP interface for PDP and EPP inquiries is defined to meet the distinct needs of each functional entity. In the case of the PDP, the focus of inquiries is on the privileges and restrictions that pertain to a process and the user of the process. For the EPP, the focus is on obligations and aspects of event patterns and event responses related to the details of an event context and the defined policy.

The methods defining this interface are listed in Table 4 below. Not all of the methods may be needed in an implementation, and some may be combined to define more complex methods. Where possible, table entries whose methods serve a related purpose are grouped together, demarcated from others by a double horizontal bar.

Table 4: Policy Inquiry Interface

Method	Description
G-AccessibleObjects(user: U): 2^{PE}	Return the set of policy elements that user U has access to, and the access rights U has on each. Used to review the capabilities of a user.
G-UsersWithAccess(element: PE): 2^{U}	Return the set of users that have access to element PE, and the access rights each user has on PE. Used to review the access control entries of an object.
G-PermittedARs (process: P, element: PE): 2^{AR}	Return the set of access rights a process holds for the given policy element for all policy classes that contain the policy element. Used to
G-DeniedARs (process: P, element: PE): 2^{AR}	Return the set of access rights a process is restricted from exercising for the given policy element. Used to compute an access decision.
G-BaseATs (user: U): 2^{AT}	Return the set of nethermost attributes for the user (i.e., attributes to which the user holds an association through a containing user attribute, which are not contained by any other attribute to which the user has an association).
G-AdjacentAscendants (element: PE): 2^{PE}	Return the set of policy elements that have an assignment emanating to the given policy element. Used to navigate policy element assignments.
G-AdjacentDescendants (element: PE): 2^{PE}	Return the set of policy elements that have an assignment emanating from the given policy element. Used to navigate policy element assignments.
G-PermittedARs (u: U, element: PE): 2^{AR}	Return the set of access rights a user holds for the given policy element. Used to compute an event response evaluation.
G-DeniedARs (u: U, element: PE): 2^{AR}	Return the set of access rights a user is restricted from exercising for the given policy element. Used to compute an event response evaluation.
G-OBLIGs (): 2^{OBLIG}	Return the set of all defined obligations. Used to retrieve the OBLIG relation for further processing.
G-EventResponses (context: EC): seq_1 EVNT_RESP	Return the set of all resolved event responses for which the given event context matched the respective event pattern. Used to process an event context against the defined obligations.
G-ATContainers (element: PE): 2^{PE}	Return the set of attributes that contain the given policy element.
G-PCContainers (element: PE): 2^{PC}	Return the set of policy classes that contain the given policy element.
G-ASSOCs (user: U): 2^{ASSOC}	Return the set of associations whose first term pertains to the given user (i.e., contains the user).
G-ASSOCs (user: UA): 2^{ASSOC}	Return the set of associations whose first term pertains to the given user attribute (i.e., is or contains the user attribute).
G-ASSOCs (at: AT): 2^{ASSOC}	Return the set of associations whose third term pertains to the given attribute (i.e., is or contains the attribute).
G-ConjPDENYs (process: P): $2^{P\_DENY\_CONJ}$	Return the set of conjunctive prohibitions that reference the given process.
G-DisjPDENYs (process: P): $2^{P\_DENY\_DISJ}$	Return the set of disjunctive prohibitions that reference the given process.
G-ConjUDENYs (user: U): $2^{U\_DENY\_CONJ}$	Return the set of conjunctive prohibitions that reference the given user.

Method	Description
G-DisjUDENYs (user: U): $2^{U\_DENY\_DISJ}$	Return the set of disjunctive prohibitions that reference the given user.
G-ConjUADENYs (ua: UA): $2^{UA\_DENY\_CONJ}$	Return the set of conjunctive prohibitions that reference the given user attribute.
G-DisjUADENYs (ua: UA): $2^{UA\_DENY\_DISJ}$	Return the set of disjunctive prohibitions that reference the given user attribute.
G-OBLIG (user: U): 2^{OBLIG}	Return the set of obligations defined by the given user.
G-OBLIG (ua: UA): 2^{OBLIG}	Return the set of obligations whose pattern references the given user attribute.
G-OBLIG (op: OP): 2^{OBLIG}	Return the set of obligations whose pattern references the given operation.
G-OBLIG (ua: UA): 2^{OBLIG}	Return the set of obligations whose response references the given user attribute.
Q-Contains (e1: PE, e2: PE): Boolean	Return a logical value (i.e., TRUE or FALSE) indicating whether the first policy element contains a second policy element.
Q-UserHasAttribute (u: U, ua: UA): Boolean	Return a logical value indicating whether the given user is contained by the user attribute.
Q-UserHasPC(u: U, pc: PC): Boolean	Return a logical value indicating whether the given user is contained by the policy class.
Q-ObjectHasAttribute (o: O, oa: OA): Boolean	Return a logical value indicating whether the given object is contained by the object attribute.
Q-ObjectHasPC(o: O, pc: PC): Boolean	Return a logical value indicating whether the given object is contained by the policy class.

4.5.3 Policy adjustment

The policy adjustment interface accepts and processes directives that are issued exclusively from PDPs and involve administration of the policy information maintained at the PIP. The authorization of the user and the process in question shall be verified as sufficient by a PDP prior to the invocation of any policy adjustment interface method.

The methods defining this interface are listed in Table 5 below. Where possible, table entries whose methods serve a related purpose are grouped together, demarcated from others by a double horizontal bar.

Table 5: Policy Adjustment Interface

Method	Description
C-Session (user: U): RESULT	Signal the start of a session for the user. Used to assemble any intermediate policy representations for the user, which may be needed by the implementation.
D-Session (user: U): RESULT	Signal the end of a session for the user. Used to dissolve any intermediate policy representations assembled for the user.
C-PC (): PC	Create a policy class and return its identifier. A null value for the identifier indicates that the action failed.

Method	Description
C-UAinPC (pc: PC): UA	Create a user attribute and assign it to the given policy class. Return the identifier of the user attribute or if the action fails, a null value.
C-UAinUA (ua: UA): UA	Create a user attribute and assign it to the given user attribute. Return the identifier of the user attribute or if the action fails, a null value.
C-UinUA (ua: UA): U	Create a user, assign it to the given user attribute, and return the identifier of the user or if the action fails, a null value.
C-OAinPC (pc: PC): OA	Create an object attribute and assign it to the given policy class. Return the identifier of the object attribute or if the action fails, a null value.
C-OAinOA (oa: OA): OA	Create an object attribute and assign it to the given object attribute. Return the identifier of the object attribute or if the action fails, a null value.
C-OinOA (oa: OA): O	Create an object and assign it to the given object attribute. Return the identifier of the object attribute or if the action fails, a null value.
D-PC (pc: PC): RESULT	Delete the given policy class. Return a result indicating whether the action succeeded.
D-UAinPC (ua: UA, pc: PC): RESULT	Delete the given user attribute and its assignment to the policy class. Return a result indicating whether the action succeeded.
D-UAinUA (ua1: UA, ua2: UA): RESULT	Delete the given user attribute and its assignment to the user attribute. Return a result indicating whether the action succeeded.
D-UinUA (u,: U, ua: UA): RESULT	Delete the given user and its assignment to the user attribute. Return a result indicating whether the action succeeded.
D-OAinPC (oa: OA, pc: PC): RESULT	Delete the given object attribute and its assignment to the policy class. Return a result indicating whether the action succeeded.
D-OAinOA (oa1: OA, oa2: OA): RESULT	Delete the given object attribute and its assignment to the object attribute. Return a result indicating whether the action succeeded.
D-OinOA (o: O, oa: OA): RESULT	Delete the given object and its assignment to the object attribute. Return a result indicating whether the action succeeded.
C-UtoUA (u: U, ua: UA): RESULT	Create an assignment from the given user to the user attribute. Return a result indicating whether the action succeeded.
C-UAtoUA (ua1: UA, ua2: UA): RESULT	Create an assignment from the first user attribute given to the second user attribute. Return a result indicating whether the action succeeded.
C-UAtoPC (ua: UA, pc: PC): RESULT	Create an assignment from the given user attribute to the policy class. Return a result indicating whether the action succeeded.
C-OAtoOA (oa1: OA, oa2: OA): RESULT	Create an assignment from the first object attribute given to the second object attribute. Return a result indicating whether the action succeeded.
C-OAtoPC (oa: OA, pc: PC): RESULT	Create an assignment from the given object attribute to the policy class. Return a result indicating whether the action succeeded.
C-Assign (e1: PE, e2: PE): RESULT	Create an assignment from the first policy element given to the second one. Return a result indicating whether the action succeeded. A possible alternative for all of the above assignment methods.
C-Assoc (ua: UA, ars: 2_1^{AR}, at: AT): RESULT	Create an association from the given user attribute specified to the other attribute given. Return a result indicating whether the action succeeded.

Method	Description
C-ConjUProhib (u: U, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Create a conjunctive prohibition on the given user for the inclusive and exclusive policy elements denoted by the respective attribute sets. Return a result indicating whether the action succeeded.
C-ConjPProhib (p: P, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Create a conjunctive prohibition on the given process for the inclusive and exclusive policy elements denoted by the respective attribute sets. Return a result indicating whether the action succeeded.
C-ConjUAProhib (ua: UA, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Create a conjunctive prohibition on the given user attribute for the inclusive and exclusive policy elements denoted by the respective attribute sets. Return a result indicating whether the action succeeded.
C-DisjUProhib (u: U, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Create a disjunctive prohibition on the given user for the inclusive and exclusive policy elements denoted by the respective attribute sets. Return a result indicating whether the action succeeded.
C-DisjPProhib (p: P, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Create a disjunctive prohibition on the given process for the inclusive and exclusive policy elements denoted by the respective attribute sets. Return a result indicating whether the action succeeded.
C-DisjUAProhib (ua: UA, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Create a disjunctive prohibition on the given user attribute for the inclusive and exclusive policy elements denoted by the respective attribute sets. Return a result indicating whether the action succeeded.
C-Oblig (u: U, pattern: seq_1 Σ_P, response: seq_1 Σ_R): RESULT	Create an obligation for the given user with the given event pattern and event response sentences. Return a result indicating whether the action succeeded.
D-Assign (e1: PE, e2: PE): RESULT	Delete an assignment between the first policy element given to the second one. Return a result indicating whether the action succeeded.
D-Assoc (ua: UA, ars: 2_1^{AR}, at: AT): RESULT	Delete an association from the given user attribute to the other attribute. Return a result indicating whether the action succeeded.
D-ConjUProhib (u: U, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Delete a conjunctive prohibition on the given user for the access right set and inclusive and exclusive attribute sets. Return a result indicating whether the action succeeded.
D-ConjPProhib (p: P, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Delete a conjunctive prohibition on the given process for the access right set and inclusive and exclusive attribute sets. Return a result indicating whether the action succeeded.
D-ConjUAProhib (ua: UA ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Delete a conjunctive prohibition on the given user attribute for the access right set and inclusive and exclusive attribute sets. Return a result indicating whether the action succeeded.
D-DisjUProhib (u: U, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Delete a disjunctive prohibition on the given user for the access right set and inclusive and exclusive attribute sets. Return a result indicating whether the action succeeded.
D-DisjPProhib (p: P, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Delete a disjunctive prohibition on the given process for the access right set and inclusive and exclusive attribute sets. Return a result indicating whether the action succeeded.
D-DisjUAProhib (ua: UA, ars: 2_1^{AR}, atis: 2^{AT}, ates: 2^{AT}): RESULT	Delete a disjunctive prohibition on the given user attribute for the access right set and inclusive and exclusive attribute sets. Return a result indicating whether the action succeeded.
D-Oblig (p: P, pattern: PATTERNid, response: RESPONSEid): RESULT	Delete the obligation for the given user with the given event pattern and response. Return a result indicating whether the action succeeded.

5 Functional Entity Requirements

5.1 Background

Requests from a user's process to access a resource or policy entity are processed by the PEP under an established session. A user may not have more than one session active at any time.

At the session start, the NGAC framework may assemble information to facilitate processing access attempts initiated for the user by one of its processes. For example, the framework may determine intermediate information, such as the object containers and objects that are accessible to the user, derived from the policy definition and the authorizations held by the user at that moment. If the policy configuration is updated, the intermediate information held could be affected, necessitating its recomputation. Intermediate representations may be maintained in a virtual or actual form at either the PAP or the PIP.

The NGAC functional entities operate mainly in a stateless fashion. The policy persisted at the PIP is the only state information that needs to be maintained by an implementation. Derived relations and other intermediate information computed from the policy may be cached in other components of an NGAC-compliant implementation, but caching is done mainly for data reuse and performance reasons, not to maintain the authorization state of the system. A stateless approach lends itself to better scalability and availability for an implementation.

Requirements for NGAC functional entities are defined in NGAC-FA. Clause 5 discusses additional factors regarding the NGAC framework, which also have a bearing on the implementation of the functional entities, and specifies additional requirements.

5.2 Common requirements

5.2.1 Overview

The functional entities of the NGAC framework need to work closely together to govern access for a computational environment. It should be no surprise that to do so securely and effectively, the entities have many characteristics in common. These characteristics are summarized as follows:

a) exclusivity;
b) discoverability;
c) trustability;
d) secure interactivity;
e) auditability;
f) resiliency; and
g) extensibility.

5.2.2 Exclusivity

The interfaces illustrated previously in Figure 1 are restricted and intended for the exclusive use of the cooperating NGAC functional entities depicted. An NGAC functional entity shall not interact with other NGAC functional entities than those depicted in the figure.

An NGAC functional entity may use the interfaces of non-NGAC entities to accomplish its tasks, provided that a trust relationship is established between them. Entities may include system-level entities that provide essential services within the computational environment, such as location services, audit logging services, and authentication services.

5.2.3 Discoverability

A functional entity shall be able to determine the points of access to a cooperating functional entity on whose interfaces it relies. The means of determining a cooperating entity's points of access are not prescribed by NGAC.

Various means of locating a cooperating functional entity may be suitable depending on the computational environment and design decisions for the implementation. For instance, a functional entity may obtain the network address of a cooperating entity via a discovery service to gain access to the interfaces the cooperating functional entity supports. A discovery service, if used in an NGAC implementation, should be realized as an independent functional entity that is distinct from NGAC functional entities and meets the criteria necessary to be treated as a trustworthy entity.

5.2.4 Trustability

A functional entity shall not use the interfaces of a cooperating functional entity without first having established a trust relationship with it. NGAC does not prescribe the means of establishing a trust relationship.

Different ways to establish trust between functional entities may exist for the computational environment of the implementation, some being better suited than others. Examples of trust relationships range from cooperating entities operating in the same supervisory mode within an operating system kernel, to the interaction between two cooperating entities across a public network using a mutual authentication and cryptographically protected connection protocol. All NGAC functional entities should meet a minimum set of requirements that are stipulated for the computational environment for the establishment of trust relationships.

5.2.5 Secure interactivity

A functional entity shall interact securely with a cooperating functional entity on whose interfaces it relies. If the functional entities are situated in different computational environments (e.g., not collocated within a single device or on the same private, secured network), communications between them should be secured. At a minimum, authentication, integrity and non-repudiation security services should apply to communications between the entities.

5.2.6 Auditability

The behavior of an NGAC functional entity's activities that pertain directly to reaching and enforcing access control decisions should be auditable. Audit information provides a valuable means for detecting and investigating violations of security. An implementation should be able to capture audit information for all security-relevant events in a well-defined format and timely manner, as individually selectable items for the purposes of regulatory compliance, liability mitigation and investigation. All audit information collected should be resistant to tampering such that unauthorized access is readily detected. The manner in which audit monitoring, collection and reporting is accomplished is not specified by this standard.

5.2.7 Resiliency

Where possible, functional entities should return to a secure mode of operation when faced with the occurrence of an unexpected event or other unexpected circumstances. Such situations may result in a disruption of service that affects the availability of the NGAC framework implementation in lieu of allowing the access control policy to be violated.

5.2.8 Extensibility

An implementation of a functional entity may incorporate features and behavior beyond those specified in the NGAC family of standards in order to extend its capabilities, provided that all mandatory features and behaviors are met by the implementation. Any additional features and behaviors shall not interfere with those mandated by the NGAC family of standards. If an NGAC functional entity does not support the extensions provided by the interface of a cooperating NGAC functional entity or vice versa, the interaction shall default to the NGAC mandated features and behavior.

5.3 PEP requirements

A PEP governs access attempts by utilizing a PDP as described in NGAC-FA. Access attempts may take the form of the process invoking the API of the PEP or the PEP intercepting and interpreting the process's invocations of other interfaces within the computational environment. Neither the API supported by a PEP nor the methods of intercepting invocations of other interfaces are prescribed by NGAC.

Each process has a unique identity, is associated with a unique user, and operates within a unique session. A PEP needs to be able to determine the identity of the process attempting access as well as the identity of the process's user. A PEP should not be aware of the details of the policy it enforces.

Policy needs to be enforced ubiquitously and continuously. A PEP should not be able to be bypassed within its scope of operation by processes running on behalf of a user. Information accessed by a user via a PEP also needs to be isolated from other users within a computational environment. The method to ensure that this property is attained is dependent upon the computational environment of the implementation and is not prescribed by NGAC.

Each PEP shall ask exactly one PDP for a decision on an access request. If for some reason that PDP fails to service the request, the PEP may ask another PDP to adjudicate the access request. Only one access request is allowed per authorization adjudication by a PDP. If multiple accesses are needed to perform a complex function, they cannot be adjudicated together as a group, and instead will require several independent access requests to be issued consecutively for adjudication.

A PEP interacts with the computational environment of a RAP (e.g., a filesystem, server, management information base or other service) to access a protected resource and carry out an action. In addition to the minimum security services mentioned in the previous clause, confidentiality services should apply to communications between these entities.

5.4 PDP requirements

PDPs are at the core of the NGAC framework, using the prevailing policy to reach decisions about access requests and event responses. As described in NGAC-FA, a PDP is utilized either by a PEP or by the EPP, respectively, for access requests and event responses. The services provided through the PDP interfaces are highly critical security services and all interactions with a PDP need to be secured accordingly.

To suit the needs of an implementation, the interfaces of a PDP may be enabled or disabled through configuration settings. That is, a PDP may be configured to interact with only certain PEPs, with only the EPP, with all PEPs and not the EPP, or with other such settings.

For a resource access that is granted, a PDP needs to obtain and return to the PEP the locator for the resource referenced by the object identifier conveyed in the access request. The details of the locator and the way in which a locator is obtained and used is not prescribed by NGAC and may be accomplished in

various ways. For example, a mapping of object identifiers to resource locators might be maintained at the PIP and retrieved via the PAP, or maintained by and retrieved from some other trusted entity.

For an administrative access that is granted, whether due to an access request or an event response, locators are not an issue since the identifiers given as operands for the administrative operation are sufficient for a PDP to carry out the access and adjust the policy via the PAP. Before granting access that involves the creation of an association, the PDP shall ensure that the user for whom the association is being created is in possession of the access rights over the policy elements in question. Otherwise, the access shall be denied. The determination of whether such delegation should be permitted is an additional check beyond that performed by the access decision function.

5.5 EPP requirements

The EPP is an optional functional entity that processes event contexts generated by a PEP or a PDP using the facilities of a PDP and the PAP as described in NGAC-FA. The EPP shall process event contexts in the order of receipt.

The EPP uses information conveyed in the event context to match the event context to the event pattern of each defined obligation. The obligation relation has unique characteristics that distinguishes it from other relations. Its main constituents, the event pattern and response, are not identifiers of defined policy entities, but instead identify strings of characters that need to be well formed grammatically for tokenization and semantic transformation during the event context information flow.

The event pattern defines conditions that trigger the execution of the event response when met. The event pattern is a logical expression that conforms to its grammar. To determine whether a triggering condition exists (i.e., the expression evaluates to TRUE), the terms of the expression are recognized and resolved with the information conveyed via the event context as well as with details from the prevailing policy configuration. The event context is used in a similar way to process terms in the event response. The event response describes one or more administrative actions that are to be taken on behalf of the obligation’s defining user. The event response needs to be conformant with its grammar, fully resolved and transformed into administrative actions in order to be carried out.

Obligations are essentially policy-modifying programs that are triggered by a successful access that meets specified conditions. Therefore, their use involves various types of risk, such as specification errors in an event pattern or response, the unexpected triggering of a response by an unforeseen event, and conflicts with administration access requests being performed concurrently.

5.6 PAP requirements

The PAP acts as a managed access point through which the policy information persisted at the PIP is accessed. Operational routines to access data representations of the prevailing policy are typically implemented at the PAP. The main objective of the PAP is to allow PDPs and the EPP access to policy information, while preserving the integrity of the authorization state of the policy and preventing them from interfering with each other’s activities. That is, the PAP controls and coordinates the processing of concurrent directives to access the policy representation.

The PAP may cache policy information persisted at the PIP for various reasons. For example, when the PAP is initiated, it may load policy information from the PIP into its memory in a form conducive to improved performance. As adjustments are made to policy during operation of the NGAC framework, both the PAP-resident policy information and the PIP-persisted policy need to be kept in synch.

A derived relation is one example of policy information that could be maintained at the PAP. Derived relations are the result of evaluating a logical expression over one or more base relations. A derived

relation may be virtual and computed as needed or maintained continuously in memory for access. The former approach requires continual reevaluation of the relation, which can affect performance negatively. However, to keep both representations of policy in synch, the latter may need to entirely reevaluate a derived relation on occasion due to alternations of policy. The implementation strategy and efficiency tradeoffs for key NGAC derived relations, namely privileges and restrictions, can have a significant effect on the performance of the PAP and the access decision function.

Another example of policy information that the PAP could maintain is the set of obligations whose terms have been preprocessed and partially resolved. Recall that the main constituents of an obligation (viz., the event pattern and response) reference character strings or sentences that should be well formed grammatically. Grammar recognition can occur at the time an obligation is defined to verify that the syntax of a sentence supplied as an event pattern or an event response is well formed. The event pattern and response cannot be fully evaluated at the time of definition, however, since some terms used in a sentence may refer to items returned in the event context, which are not available until the time obligations are matched and processed. Therefore, the resolution of undefined terms cannot take place until then. Nevertheless, the event pattern and response, as sentences in a language, can be parsed and converted into an intermediate representation at the time of definition, and maintained at the PAP for later interpretation and final resolution. The benefit is that the bulk of the resolution work can occur at definition time, reducing the effort needed during obligation matching and response processing.

5.7 PIP requirements

The PIP acts as the gateway for the PAP to the NGAC policy. The behavior of the PIP with respect to changes to the authorization state of the policy is formally specified in NGAC-GOADS. Commands issued by the PAP to access the basic elements, containers, and relations that represent policy are actualized at the PIP.

5.8 RAP requirements

A RAP acts as the gateway for PEPs to one or more protected resources under its control. Commands issued by PEPs to access protected resources (e.g., read and write) are actualized at a RAP. A RAP may be used to monitor the status of resources as well as affect them.

6 Other Considerations

6.1 Interoperation of functional entities

The NGAC family of standards provides the architectural, functional and interface definitions necessary to create a full-featured access control system. It is important to note that aspects of the specifications were intentionally left open to allow an implementation to be tailored to the control objectives of the system and the computational environment involved. The NGAC specifications allow functional entities to be implemented independently and function in a consistent manner when deployed together, but only if the details of those open areas are in agreement.

The areas in which agreement needs to be reached include the following technical details:

a) the syntax and semantics for GUIDs;
b) the naming conventions for policy elements;
c) the resource and administrative operations and access rights used to govern accesses;
d) the alphabet and grammar for the language(s) used to express event patterns and responses in obligations;
e) the procedures for authenticating the identity of a user and for establishing trust relationships between functional entities;
f) the particulars of the interfaces used between functional entities, including the concrete encoding of parameters and the options and extensions supported; and
g) the syntax and semantics of resource locators.

6.2 Policy

6.2.1 Representation

NGAC policy is defined in terms of abstractions on the resources and authorizations of a computational environment and the behavior of conceptual functional entities. While certain abstractions are specified for the purposes of standardization, they need to be reified by actual processing entities or settings within the computational environment of an implementation. The adaptation of the abstractions of the security model to the constituents of the implementation environment requires taking into consideration information about the environment when deciding how the abstractions are to be actualized.

6.2.2 Updates

The NGAC security model is specified utilizing a single data store in which the policy is maintained. When the policy is modified, the modifications take immediate effect. While the model captures the required behavior of applied policy updates, it does not address the more general issue of ensuring that the updates are of high caliber. Because updates have a direct impact on the results of the access decision function, it would be useful to provide one or more means to vet intended updates before they are applied to the active policy.

A policy management application can be an important tool in understanding policy abstractions and the impact that policy settings have on controlling behavior. A policy management tool should be able to screen proposed changes before they are applied to verify that they are well formed and do not produce inconstancies in the policy. Graphical renditions are an effective aid in comprehending and administering policy for which the NGAC is well suited.

Establishing separate, distinct spaces for creating and testing policy revisions before they are activated can also prove useful in practice. For example, two policy stores could be maintained: one for the test policy and one other for the active policy. The active policy store contains the policy used to compute runtime access decisions, while the test policy store is used to apply revisions to a copy of the active policy for vetting. Once vetting of a test policy is complete, it can replace the active policy. Some means would also need to be established to transition the test policy to the active policy in a synchronized fashion that avoids disrupting operations.

6.2.3 Performance

The operation of the NGAC relies squarely on the defined policy. Policy information is continually retrieved and updated by functional entities during runtime, making the velocity of those transactions a critical factor in the overall performance of an implementation. A practical method for improving response is to create a high-speed memory cache of all or parts of the prevailing policy persisted at the PIP and use that representation for access decisions and policy analytics. Caching can be done in various ways. For example, changes to policy could be immediately reflected on the intermediate representation of the policy elements and relations in memory and applied to the PIP-persisted policy in a synchronous fashion to maintain consistency. Any updates made directly to the PIP-persisted policy would also need to be applied automatically to the cached, intermediate representation of policy. The intermediate representation could also be serialized at shut down to enable its quick restoration upon start up.

The data structures used to represent cached policy and the algorithms employed to search that intermediate representation have a direct impact on how quickly policy inquiries can be processed. The determination of efficient policy representations and algorithms for enabling rapid evaluation is an area for study. Annex A provides an example of an algorithm and data structure designed to perform policy analysis efficiently, which can be adapted for various purposes.

6.3 Race conditions

The purpose of the event context information flow is to modify aspects of policy based on the occurrence of events and the set of defined obligations in effect when they occur. The modifications may affect access to the resource or policy information referenced by the access request that triggered the event context information flow, as well as to other resources or policy information that are related to the access. Therefore, the potential for race conditions exists in the architecture between administrative and resource access requests and policy changes spawned from an event context information flow, and also among policy changes from concurrent event context information flows. Unexpected and unwanted changes to policy due to concurrent access by multiple functional entities need to be avoided, and the following objectives attained:

a) An event context flow should not affect the processing of the access request that triggered it;
b) An event context flow should be able to affect, where appropriate, subsequent access requests from the process or user referenced in the event context; and
c) An event context flow should be able to affect, where appropriate, access requests from other processes that are adjudicated after the access request of the event context flow.

Race conditions should be addressed in a manner suited to the computational environment of an implementation of the NGAC framework. The manner in which race conditions are addressed is not prescribed by NGAC. Possible methods to mitigate their impact include the following:

a) Using the locking features of the PIP data store to prevent access to specific policy information structures being affected by an event context information flow, until the flow completes;
b) Delaying the return of the results from a successful resource or administration access to the CA until the event context information flow for the access completes; and

c) Enforcing a queue structure within a PEP for delaying access attempts initiated within a session by a CA until the previous access completes.

6.4 Collocated functional entities

The NGAC framework can be adapted to a range of computational environments, from a single, self-contained computer system, to a group of individual network-interconnected systems. An implementation is not required to follow a completely centralized or completely distributed approach. Many types of hybrid configurations are also possible where some of the functional entities within the functional architecture reside together within a single system while the remaining functional entities are located in other systems.

Collocating functional entities can beneficial in certain computational environments. The benefits can include simplified identification and authentication of collocated functional entities and their services, avoidance of network latency between functional entities, and reuse of programmed functionality.

6.4.1 PEP collocation

A PEP may be collocated in various ways. For example, a PEP could be collocated with the application, protected within the kernel space or a trusted layer of the operating environment, to screen access attempts to protected resources via a PDP. Alternatively, a PEP could be collocated with a specific RAP to screen access attempts to protected resources associated with the RAP. The latter is appropriate only if the access attempts of each user pertain solely to the resources of a specific RAP (e.g., only a single RAP exists).

To retrofit existing, non-compliant applications requires a collocated PEP to intercept native access attempts, translate them for compatibility with the NGAC framework and request adjudication from a PDP. The PEP would need to employ the specifications for native access requests to convert them to and from NGAC-conformant requests and responses.

6.4.2 PDP collocation

To reduce the overhead between PEP and PDP interactions, it is possible to closely couple them together (e.g., as an integrated PEP/PDP functional entity). While a PDP may be collocated and closely integrated with a PEP, there are advantages to decoupling one from the other. One benefit of decoupling is that, if needed, the possibility exists to replace the PDP implementation being used with another standard conformant PDP implementation. Simplified maintenance of the PDP implementation is another benefit of decoupling, since a PDP that is independent of the logic and programming of a PEP is neither affected by PEP dependencies (e.g., third-party program libraries) nor by PEP maintenance (e.g., specification changes, patches, and new code deployments).

A PDP operates in a stateless fashion, which makes it suited to have several instances operating concurrently. PDP instances may run on separate threads (e.g., to leverage the availability of multi-core architectures) and on separate hosts (e.g., to increase the capacity of the framework). As with any concurrent operations, care needs to be taken to ensure that interference is minimized and race conditions are avoided.

It may be useful in some situations to establish a central PDP hub by collocating several PDPs together. The PDP hub would behave more like a relay than a PDP. When a request is received by the hub, the details of the request would be used to route the request to a relevant PDP for adjudication. Alternatively, a PDP discovery service could be used to facilitate locating an available PDP to adjudicate access requests.

6.4.3 EPP collocation

The EPP uses the PAP to resolve and match obligations. It also relies entirely on the functionality of the PDP to carry out the processing of responses of matched obligations. Since the functional architecture allows for multiple PDPs, an obvious approach to avoid contention for a PDP is to collocate and tightly couple the EPP with a PDP, designating the PDP for its exclusive use.

6.4.4 PAP collocation

The functional architecture allows only a single PAP and a single PIP to exist within the framework. The services offered by the PAP are at the same conceptual level as other NGAC service interfaces, while that of the PIP are at a more elemental, constituent level relied on by the PAP. These facts compel the collocation of the two functional entities.

6.5 Domain definition and management

NGAC relies on the existence of a principle administrator to establish the initial policy for the framework and manage it. A policy domain is an abstraction of the general configuration and characteristics of the resources within the domain boundaries and the security rules that govern access to these resources. The principle administrator can allocate its responsibilities by defining one or more administrative subdomains and designating administrators for them. The subdomain boundaries may be defined in the context of administrative, business, geographical, and political constraints.

Decentralizing administrative responsibilities through subdomain allocations allows for the coexistence of multiple administrators with measured control over distinct portions of policy, such that an interrelated and consistent policy can be defined and managed in a coordinated manner. The portion of policy allocated to a subdomain may be administered by multiple authorities, but carried out in a coordinated fashion.

Annex A
(Informative)

Policy Computations

A.1 Introduction

It should be of no surprise that the choice of algorithms used to perform various policy computations required of an implementation can greatly affect performance. The computations at the center of the NGAC framework are the determination of the access rights a user has to objects, the adjudication of an access request from a user, and the display of relevant objects for reviews by a user. Fortunately, efficient algorithms exist to perform key NGAC policy computations. This annex describes in detail an efficient algorithm to calculate the access rights a user has to objects representing protected resources. The algorithm can also be easily adapted to make various other key policy determinations.[1]

A.2 Background

A simple example policy is used to illustrate the steps of the basic algorithm. In this example, a savings and loan bank comprised of several branches has the following policy:

a) Tellers can read and write accounts only for the branches to which they are assigned; and
b) Loan officers can read and write loans only for the branches to which they are assigned.

A representation of the policy, populated with several users (viz., u1, u2 and u3) and objects (viz., l11, l12, a11 and a21), is shown in Figure A.1. Assignments are depicted as blue or green arrowed lines emanating from an attribute, user or object to another policy element. Associations are depicted as red dashed lines that span two attributes and are labeled with access rights. Two policy classes are in effect: the branch-constraints policy class containing attributes, users and objects linked via blue assignments, and the position-constraints policy class containing attributes, users and objects linked via green assignments.

Associations between policy elements of the branch-constraints policy class allow employees assigned to one or more branches to access only the products that pertain to those branches (i.e., their respective accounts and loans). Associations between policy elements of the position-constraints policy class allow employees assigned to a position to access only the types of assets that pertain to their position.

The algorithm utilizes the Directed Acyclic Graph (DAG) of the assignment relation, illustrated in Figure A.1, as the basis for processing. Each policy element (i.e., a user, object, attribute, and policy class) is a node of the DAG and each assignment is a directed edge between two nodes. The association relation is also depicted in Figure A.1 by the red dashed lines between nodes.

[1] The algorithm in this annex is based on that described in the following journal article: Peter Mell, James Shook, Richard Harang and Serban Gavrila, *Linear Time Algorithms to Restrict Insider Access using Multi-Policy Access Control Systems*, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, vol. 8, num. 1, March 2017, pp. 4-25, URL: http://isyou.info/jowua/papers/jowua-v8n1-1.pdf.

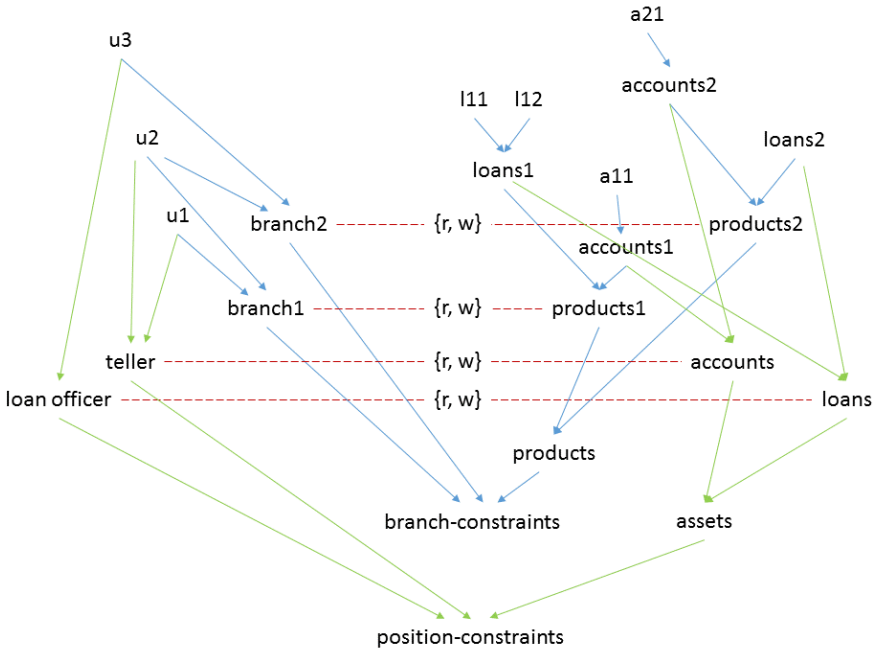

Figure A.1: Simple Bank Policy Representation

The algorithm's computations entail traversing the assignment relation DAG in various directions. The data structures chosen to represent the graph should be well suited for this purpose. The algorithm, for instance, uses an adjacency list representation for the immediate successors of a node, implemented as a dictionary that maps a key (i.e., a node) to a value (i.e., a list of immediate successors) via a hash, which allows quick traversal of directed paths emanating from a node. A second dictionary representing the immediate predecessors of a node is also used to allow quick traversal of directed paths leading to a node.

A.3 Algorithm details

To compute the access rights a user holds over objects, the algorithm traverses the policy graph vertically, in both downward and upward directions. A downward direction in the DAG refers to processing from the tail of a node toward defined policy classes, and an upward direction refers to processing from the head of a node toward defined users or objects. The objective in this example is to determine the set of objects accessible to user u1 and the set of access rights authorized for each of them.

A.3.1 Find the source association nodes

The algorithm begins with the user in question, traversing downward via a breadth first search to identify all reachable user attribute nodes that are source nodes of an association (i.e., the first term of one or more defined associations) along with the set of associations concerned. From this point on, only the source association nodes and their corresponding associations require further consideration.

Figure A.2 illustrates the source association nodes identified in this example for user u1: **teller** and **branch1**. Each source node involves a single read-write association to a destination node.

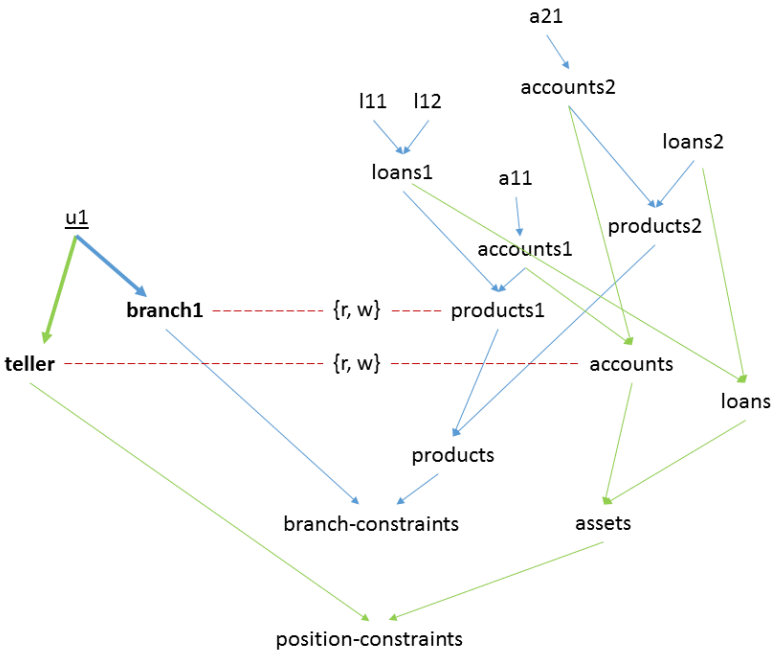

Figure A.2: Source Association Nodes

A.3.2 Find the destination association nodes

For each association from an identified source association node, the respective destination node of the association (i.e., the third term of the association) is then identified. Each identified destination node (viz., an object attribute) is labeled with the set of access rights of the association.[2] The algorithm labels a destination node using a node attribute (i.e., a dictionary with the node as a key), which can be assigned and return various data structures, such as a set, list or dictionary, based on the key provided. In this case a set of access rights is assigned. Since a destination node may be involved in multiple associations, labeling requires forming the union of the set of access rights of each association that references the destination node.

Figure A.3 illustrates the destination association nodes identified in this example, which would be labeled as follows: **products1** – {r, w} and **accounts** – {r, w}.

[2] While all the destination nodes in this example are object attributes, in general, this may not always be the case. Policies can be constructed using associations that involve user attribute destination nodes, which are ignored in this computation since the purpose is to identify accessible objects representing resources. However, this algorithm can be easily altered to address user attributes and users, if their accessibility is of interest.

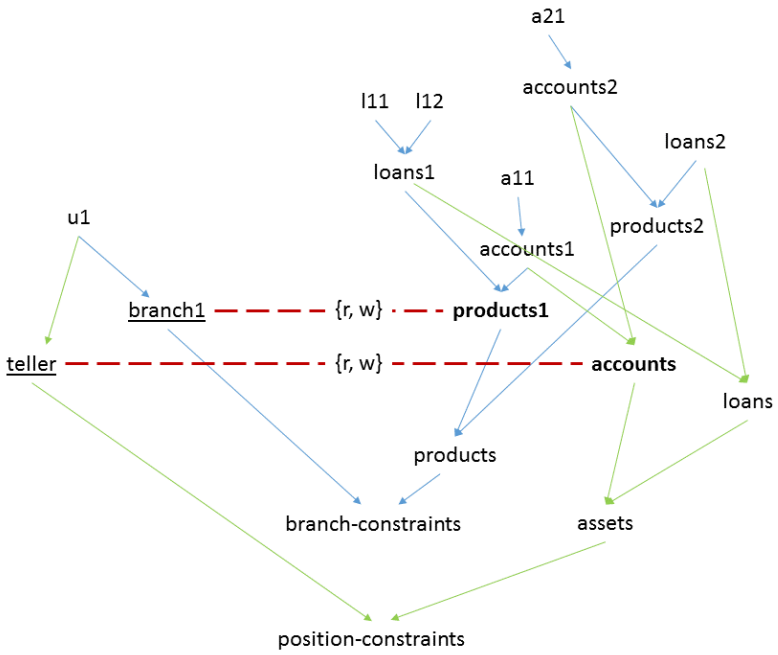

Figure A.3: Destination Association Nodes

A.3.3 Find the objects of interest

The set of objects of interest for the user can now be identified by performing a reverse-edge, breadth first search upward from the set of destination association nodes identified, treating the destination nodes as the first iteration of the search. Any object attribute node that is not on a reverse path from a destination node to an object node can be ignored.

Figure A.4 illustrates the objects of interest identified in this example, which are as follows: **l11**, **l12** (via products1 and loans1), **a11** (via products1 and accounts1 and also accounts and accounts1), **a21** (via accounts and accounts2).

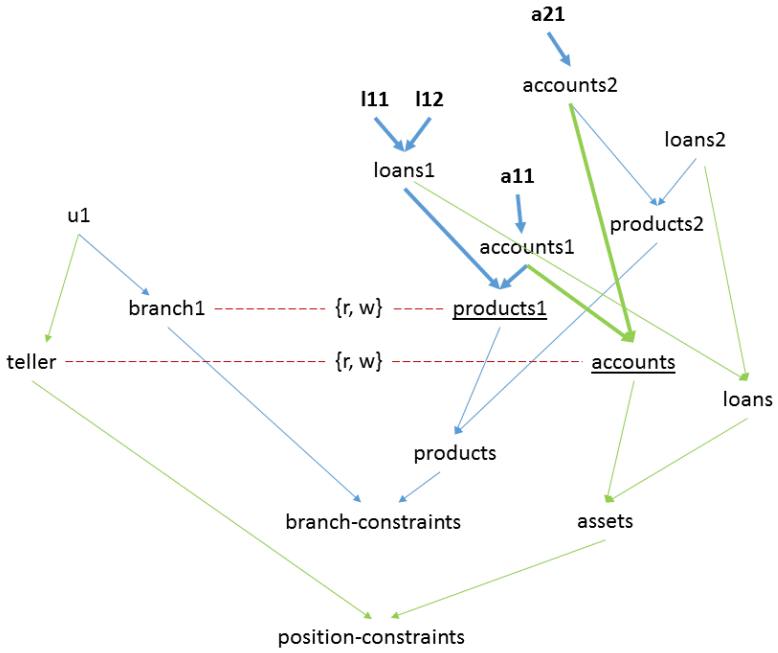

Figure A.4: Objects of Interest

A.3.4 Determine the policy classes that contain an identified object

The computation of a privilege requires knowledge of the policy classes that contain an object of interest. To make this determination, the algorithm performs a topological ordering of the nodes containing each object of interest using a recursive, depth first search. When a policy class node is visited, the algorithm retains the policy class identifier and uses it to cumulatively label each of its ancestor nodes as they are subsequently processed such that each node, including the object of interest, eventually records the set of policy class nodes that contain it. The policy class information recorded at an object attribute node is reused when processing the remaining objects of interest, in lieu of reprocessing and relabeling the node, which is a critical aspect of the algorithm's performance.

Figures A.5 and A.6, illustrate the traversal of the DAG for two of the four objects of interest in this example: l11 and a11. Note that in the traversal for a11, the processing of certain nodes is skipped, which is indicated by dashed arrows.

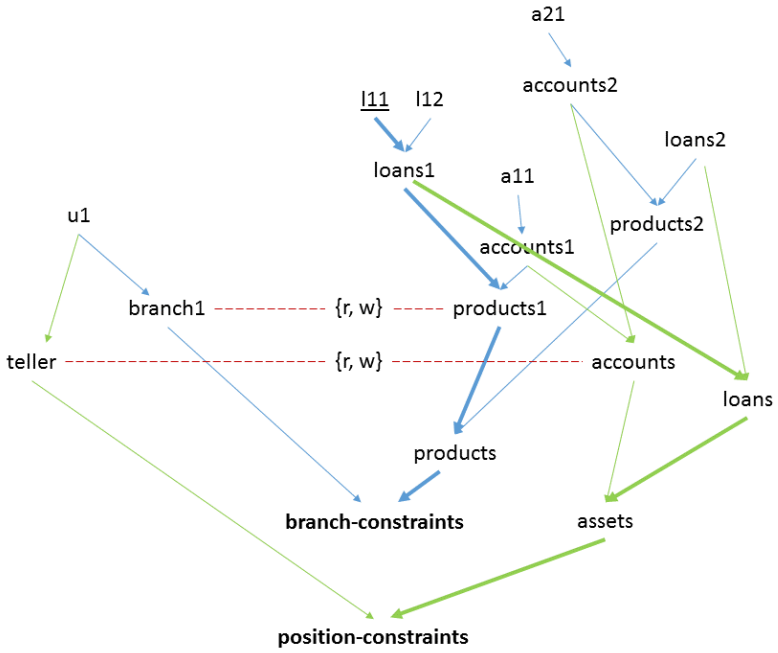

Figure A.5: Depth First Search from Object l11 to Policy Classes

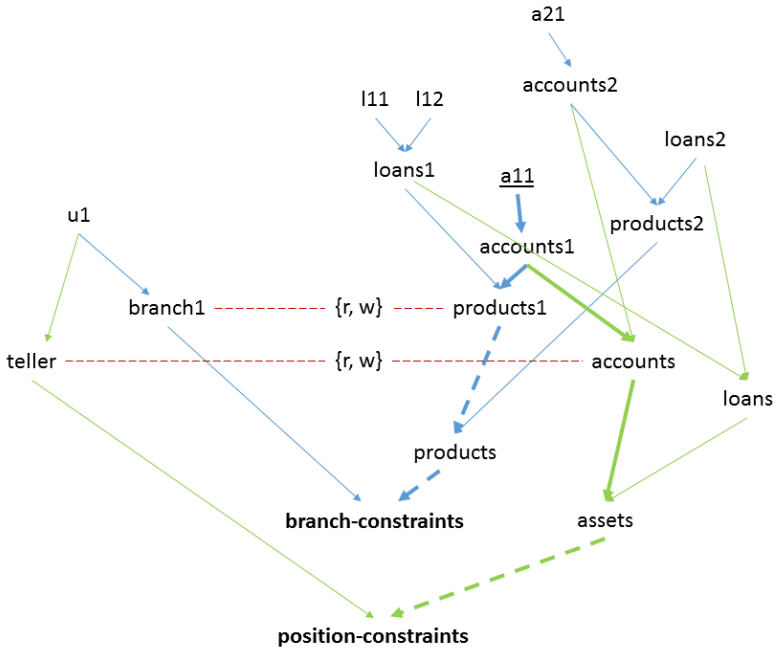

Figure A.6: Depth First Search from Object a11 to Polciy Classes

The results from this stage of processing indicate that all four objects of interest, l11, a11, l12 and a21, are contained by both the branch-constraints (bc) and product-constraints (pc) policy classes. The policy class labeling assignments to nodes of the DAG are summarized below for l11 and a11 as well as l12 and a21. Each node is prefixed with a pair of integers that indicate respectively the step at which processing began and ended for the node. For the second, third and fourth objects of interest on which the depth first

search occurs, some of the nodes encountered are already labeled, allowing the algorithm to avoid reprocessing them.

Steps: Node – Policy Class	Access Rights
01/16: l11 – {bc, pc};	
02/15: loans1 – {bc, pc}	
03/08: products1 – {bc}	{r, w}
04/07: products – {bc}	
05/06: **bc**	
09/14: loans – {pc}	
10/13: assets – {pc}	
11/12: **pc**	
01/10: a11 – {bc, pc};	
02/09: accounts1 – {bc, pc}	
03/04: products1 – {bc}	{r, w}
skip/-: products – {bc}	
skip/-: **bc**	
05/08: accounts – {pc}	{r, w}
06/07: assets – {pc}	
skip/-: **pc**	
01/04: l12 – {bc, pc};	
02/03: loans1 – {bc, pc}	
skip/-: products1 – {bc}	{r, w}
…	
01/10: a21 – {pc, bc};	
02/09: accounts2 – {pc, bc}	
03/08: accounts – {pc}	{r, w}
skip/-: assets – {pc}	
…	
04/07: products2 – {bc}	
05/06: products – {bc}	
skip/-: **bc**	

A.3.5 Determine the access rights that pertain to a containing policy class

When performing the depth first search described above, additional details about processed nodes can be propagated upward to the object of interest. The algorithm takes advantage of this opportunity to label nodes with additional information concerning access rights.

During the depth first search, at the time the identifier of a policy class node is used to label its ancestor nodes, the ancestor node can instead be labeled with a mapping from the policy class identifier to the set of prevailing access rights for the ancestor node, which may be the empty set. The prevailing access rights of a node are null, unless a reachable destination node has been processed and labeled with a mapping from the policy class node to the access right label previously assigned to the destination node.

The algorithm uses a dictionary for the policy class-to-access rights mapping in which the key is a policy class node and the value is a set of access rights. The behavior of the depth first search allows the policy class dictionaries to be propagated upward to the object of interest. When processing a node in which two or more successor nodes are labelled with policy class dictionaries, the node is labelled with the union of those dictionaries, whose keys are the union of the keys from each successor dictionary, and whose values are the union of the values for each identical key from each successor dictionary. The

determination of policy class-to-access rights mapping for each object of interest in this example is summarized below.

Steps: Node – Policy Class	Access Rights	Policy Class to Access Rights Mapping
01/16: l11 – {bc, pc};		bc → {r, w}, pc → {}
02/15: loans1 – {bc, pc}		bc → {r, w}, pc → {}
03/08: products1 – {bc}	{r, w}	bc → {r, w}
04/07: products – {bc}		bc → {}
05/06: **bc**		
09/14: loans – {pc}		pc → {}
10/13: assets – {pc}		pc → {}
11/12: **pc**		
01/10: a11 – {bc, pc};		bc → {r, w}, pc → {r, w}
02/09: accounts1 – {bc, pc}		bc → {r, w}, pc →{r, w}
03/04: products1 – {bc}	{r, w}	bc → {r, w}
skip/-: products – {bc}		bc → {}
skip/-: **bc**		
05/08: accounts – {pc}	{r, w}	pc → {r, w}
06/07: assets – {pc}		pc → {}
skip/-: **pc**		
01/04: l12 – {bc, pc};		bc → {r, w}, pc → {}
02/03: loans1 – {bc, pc}		bc → {r, w}, pc → {}
skip/-: products1 – {bc}	{r, w}	bc → {r, w}
…		
01/10: a21 – {pc, bc};		pc → {r, w}, bc → {}
02/09: accounts2 – { pc, bc}		pc → {r, w}, bc → {}
03/08: accounts – {pc}	{r, w}	pc → {r, w}
skip/-: assets – {pc}		pc → {}
…		
04/07: products2 – {bc}		bc → {}
05/06: products – {bc}		bc → {}
skip/-: **bc**		

A.3.6 Determine the user's access rights for each object of interest

The final step of the algorithm is to determine the access rights authorized for the user using an object's dictionary of policy class to access rights mappings. For each object of interest, compute the intersection of the associated set of access rights of each policy class in its dictionary. The resulting set contains the user's access rights for the object. If no access rights remain, the object cannot be accessed.

The computation for user u1 in the example policy results in r, w access rights for object a11, as illustrated below.

Object of Interest	Policy Class to Access Rights Mapping	Authorized Access Rights
l11	bc → {r, w}, pc → {}	none
a11	bc → {r, w}, pc → {r, w}	{r, w}
l12	bc → {r, w}, pc → {}	none
a21	pc → {r, w}, bc → {}	none

A.4 Algorithm variants

The basic algorithm described above and its search mechanisms can be adjusted to efficiently perform other policy computations. For example, a few simple changes are all that is needed to determine whether a user holds sufficient privileges to perform an operation on an object. First, as soon as the objects of interest are identified, intersect the object in question with the set of objects of interest to form a new set. An empty intersection equates to a deny decision (i.e., no access allowed) and termination of the algorithm. A singleton requires continuation of the algorithm, but only with the one object of interest. Second, at the end of the algorithm, when the user's authorized access rights are computed for the single object of interest, determine whether the access rights are sufficient to perform the operation.

In this example policy, for instance, assume the following access request needs to be adjudicated: user: u1, op: read, object: a11. Intersecting a11 with the four objects of interest computed earlier returns the singleton, {a11}. The adjusted algorithm continues with the recursive, depth first search, topological sort as described earlier, but only for the one object of interest, a11. In this case, the algorithm does not skip certain nodes as described previously, since no other objects of interest that would record information for reuse in the a11 depth first search are being processed. Nevertheless, the algorithm determines the policy class to access rights mapping for object a11, as before, namely bc → {r, w}, pc → {r, w}. Since read and write resource operations map on a one-to-one basis to r and w access rights, the user's read access request can be granted.