VHA
Health Information Systems

Privilege Management Infrastructure

VHA Role Based Access Control (RBAC)

Constraint Catalog

Version 1.34

14 September 2007

Record of Changes

Date	Version	Description	By
12/26/2006	1.0	Initial Draft	Suzanne Gonzales-Webb
12/31/2006	1.1	Quality Assurance Review / Revision	Craig Winter
05/10/2007	1.2	Update	Suzanne Gonzales-Webb
05/11/2007	1.3	Quality Assurance Review / Revision	Craig Winter
06/25/2007	1.31	Update	Suzanne Gonzales-Webb
07/20/2007	1.32	Update post peer review	Suzanne Gonzales-Webb
7/23/2007	1.33	Quality Assurance Review / Revision	Craig Winter
9/14/2007	1.34	Quality Assurance Review / Revision	Craig Winter

Table of Contents

List of Tables

List of Figures

List of Appendices

1 Introduction

Role-Based Access Control or RBAC is a method to control access to resources on an information system. It was developed to overcome the complexities of managing individual user permissions and their assignments.[i] Access control and authorization services ensure that people, computer systems, and software applications can use only those resources (e.g., files, directories, computers, networks) that they are authorized to use and then only for approved purposes. Access controls protect against unauthorized use, disclosure, modification, and destruction of resources and unauthorized issuing of system commands. [HITSP]

The intent of this document is to introduce a process to introduce constraints upon the identified healthcare permissions as presented in the HL7 RBAC Permission Catalog. Future iterations of this constraint document may introduce constraints as applied to structural roles and possibly functional roles. This document presents an overview of constraint types and introduces permission constraint identification using a proposed high level process. The licensed and certified healthcare provider healthcare permissions cited are derived from the HL7 RBAC Healthcare Permission Catalog tables. Future updates may include healthcare permissions that may be assigned to non-licensed healthcare personnel.

Constraints are restrictions (conditions or obligations) that are enforced upon access permissions. In RBAC, a constraint may restrict for example, a user to continue to have an *action* on the data they are accessing. They can include contextual properties such as separation of duties, time-dependency, mutual exclusivity, cardinality, location, etc. For the complex healthcare environments, constraints provide the higher flexibility required in RBAC implementation (see [Neumann Strembeck]).

Examples of contextual constraints could include:

- Head Nurse on a hospital floor at any given time (cardinality of 1, time-dependency),
- Chief of Staff (cardinality of 1),
- Lab Technician vs. Lab Technician Supervisor (separation of duties),
- Provider's access to a remote hospital that is not his/her primary workplace (location), and
- A physician working scheduled clinic hours (time-dependency) vs. physician working in a 24 hour Emergency Room (no time-dependency).

With respect to access control, one has to ask first which parts of these unmanageable quantities of context information are relevant for a specific authorization decision, and how the corresponding information may be elicited and defined on the modeling level. In this document we suggest a process for the specification of context constraints. This process is based as an extension to the scenario-driven role engineering process for RBAC roles presented in Neumann

and Strembeck [2002]. Prior to describing the engineering of context constraints in detail, we give some background information concerning the scenario-driven role engineering process.

In Role-Based Access Control, users (i.e., individuals or authorization services, etc.) are given a set of permissions—the ability to have an action (create, read, write, etc.) on an object (a laboratory order, patient history, etc.). In a healthcare environment arena however, flexibility in RBAC is needed as the duties and functions of identified structural roles[ii] such as a physician, nurse or pharmacist[1] in relation to accessing an information system can vary on the time of day, their location (i.e., clinic, ward) and when given additional temporary duties (i.e., supervisory or administrative). These conditional changes or constraints modify the level of access control an individual user may have. One possibility to deal with this dynamically changing context is to rapidly modify permission assignment relations according to the changes in the [healthcare] environment. This central idea supports constraints on almost all parts of an RBAC model (e.g., permissions, roles, or assignment relations) to achieve a high flexibility.[iii]

Using the Context Constraint Examples above, permission constraints could include:

- Head Nurse permission functions can be accessed only by one Registered Nurse per 12-hour shift on a hospital floor at any given time (cardinality of 1, time-dependency),
- Only one Physician may have access to the Chief of Staff permissions (cardinality of 1),
- A laboratory user can co-sign another Lab Technician's results, but cannot co-sign their own even if logged on as the Lab Technician Supervisor (separation of duties),
- Provider's access to a remote hospital that is not his/her primary workplace (location), and
- A physician working scheduled clinic hours (time-dependency) vs. physician working in a 24 hour Emergency Room (no time-dependency).

With respect to access control, one has to ask first which parts of these unmanageable quantities of context information are relevant for a specific authorization decision, and how the corresponding information may be elicited and defined on the modeling level. In this document we suggest a process for the specification of context constraints. This process is based as an extension to the scenario-driven role engineering process for RBAC roles presented in Neumann and Strembeck [2002]. Prior to describing the engineering of context constraints in detail, we give some background information concerning the scenario-driven role engineering process.

[1] Structural roles are categories of healthcare personnel warranting differing levels of access control. Structural roles allow a user to 'connect' to a resource, but do not grant authorization. Structural roles define what specific healthcare workflow users are allowed to participate in, while functional roles define authorizations granted to an entity to allow access (i.e., to protected health information).

In the scenario-driven role engineering process usage, scenarios of an information system are used to derive permissions and to define tasks. In general, a scenario describes an action and event sequence, for example, to register a new patient in a hospital information system. Thus, each scenario consists of several steps, and a subject performing a scenario must possess all permissions that are needed to complete the different steps of this scenario. In turn, a task consists of one or more scenarios, and tasks are combined to form work profiles. A work profile comprises all tasks that a certain role (functional role) is allowed to perform. In a hospital environment different work profiles for physicians, nurses, and clerks are needed, for instance. In the role engineering process, work profiles are then used together with the permission catalog and the constraint catalog to define a concrete RBAC model. However, the scenario-driven approach presented in Neumann and Strembeck [2002] only provides general guidance for the sub process of defining (exogenous) constraints. This fact and our aim to specify and enforce context constraints in an RBAC environment led us to the definition of the process extension proposed in this section. [Neumann Strembeck]

According to Neumann and Strembeck,[iv] "A *context constraint* is defined as a dynamic RBAC constraint that checks the actual values of one or more contextual attributes for pre-defined conditions. If these conditions are satisfied, the corresponding access request can be permitted. Accordingly, a *conditional permission* is an RBAC permission that is constrained by one or more context constraints." Thus, constraints are restrictions that are enforced upon access permissions. They can include contextual properties such as separation of duties, time-dependency, mutual exclusivity, cardinality, location, etc. Context constraints are used to define conditional permissions.

The conditions to be satisfied can be either positive (e.g., "…location is ward," or negative (e.g., "…the location is not Texas.") In the first instance if the location is "Ward," then access is granted, otherwise access from all other locations is denied. In the second instance, all locations will be granted access except for "Texas."

Table 1 lists definitions of terms used in this document.

Table 1: Definitions[v] [vi]

Term	Definition	Source
Cardinality	*Cardinality* occurs when there is a limit to the certain number of roles (users, people, etc.) who may be holding permission at any one time.	[Neumann-Strembeck]
Conditional Permission	A *conditional permission* is a permission that is associated with one or more context constraints and grants access if each corresponding context constraint evaluates to "true." Therefore, conditional permissions grant an access operation if the actual values of the context attributes captured from the environment fulfill the attached context constraints. The relation between context constraints and permissions is a many-to-many relation. A number of permissions can be associated with the same context constraint if necessary. Similarly, one permission may have many context constraints associated with it.	[Neumann-Strembeck]
Context Constraint	A *context constraint* is a clause containing one or more context conditions. It is satisfied if (if and only if) all its context conditions hold. Context constraints are used to define conditional permissions. [In an RBAC environment], a context constraint is defined through the terms context attribute, context function, and context condition: —A *context attribute* represents a certain property of the environment whose actual value might change dynamically (like time, date, or session-data for example) or which varies for different instances of the same abstract entity (e.g., location, ownership, birthday, or nationality). Context attributes are a means to make (exogenous) context information explicit. —A *context function* is a mechanism to obtain the current value of a specific context attribute (i.e., to explicitly capture context information). For example, a function *date* () could be defined to return the current date. Of course a context function can also receive one or more input parameters. For example, a function *age*(*subject*) may take the subject name out of the _*subject*, *operation*, *object*_ triple to acquire the age of the subject, which initiated the current access request (e.g., the age can be read from some database). —A *context condition* is a predicate (a Boolean function) that consists of an operator and two or more operands. The first operand always represents a certain context attribute, while the other operands may be either context attributes or constant values. All variables must be ground before evaluation. Therefore, each context attribute is replaced with a constant value by using the corresponding context function prior to the evaluation of the respective condition.	[Neumann-Strembeck]

Term	Definition	Source
Location	*Location is a* constraint that creates a location requirement for the role holding the permission. Creation of a location requirement for the role holding the permission	[Neumann-Strembeck]
Mutual Exclusivity	To be mutually exclusive, the minimum user cardinality is 'one' *e.g., constraints like: the roles "accounting clerk" and "controller" must be statically mutual exclusive, or the minimum user cardinality for the "controller" role is "one".* Two mutual exclusive roles must never be assigned to the same subject simultaneously, as specified in static separation of duty (SSD) constraints. In dynamic separation of duty constraints which define that two mutual exclusive roles must never be activated simultaneously within the same user session, or time constraints which restrict role activation to a specific time interval (e.g., from 8 a.m. to 8 p.m.).	[Neumann-Strembeck]
Object	An *object* is an entity that contains or receives information. The *objects* can represent information containers (e.g., files or directories in an operating system, and/or columns, rows, tables, and views within a database management system) or *objects* can represent exhaustible system resources, such as printers, disk space, and CPU cycles. The set of *objects* covered by RBAC includes all of the objects listed in the permission catalog that are assigned to roles.	[ANSI-RBAC]
Operation	An *operation* is a capability provided by a currently executing program (i.e., an executable image) which upon invocation executes some function for the user. Within a file system, *operations* might include read, write, and execute. Within a database management system, *operations* might include insert, delete, append, and update. Basic Permission Name Operations: A = Append C = Create R = Read U = Update D = Delete E = Execute	[ANSI-RBAC]
Permission	*Permission* is an approval to perform an operation on one or more RBAC protected objects.	[ANSI-RBAC]

Term	Definition	Source
Separation of Duties	*Separation of duty* occurs when a single role (user, person, etc.) cannot hold two functionally conflicting permissions at the same time.	[Neumann-Strembeck]
Time Dependency	*Time-dependency* creates a time of day/hour dependence on the role holding the permission.	[Neumann-Strembeck]

2 Context Constraints

A context constraint specifies that certain context attributes must meet certain conditions in order to permit a specific operation. A context constraint is defined (and primarily used) as a dynamic exogenous authorization constraint.

From Neumann and Strembeck, "a context constraint is defined through the terms context attribute, context function, and context condition:

> —A *context attribute* represents a certain property of the environment whose actual value might change dynamically (like time, date, or session-data for example) or which varies for different instances of the same abstract entity (e.g., location, ownership, birthday, or nationality). Thus, context attributes are a means to make (exogenous) context information explicit. On the programming level, each context attribute *CA* represents a variable that is associated with a *domain* which determines the type and range of values this attribute may take (e.g., date, real, integer, string).
>
> —A *context function* is a mechanism to obtain the current value of a specific context attribute (i.e., to explicitly capture context information). For example, a function *date*() could be defined to return the current date. Of course a context function can also receive one or more input parameters. For example, a function *age*(*subject*) may take the subject name out of the *_subject*, *operation*, *object_* triple to acquire the age of the subject, which initiated the current access request (e.g., the age can be read from some database).
>
> —A *context condition* is a predicate (a Boolean function) that consists of an operator and two or more operands. The first operand always represents a certain context attribute, while the other operands may be either context attributes or constant values. All variables must be ground before evaluation. Therefore, each context attribute is replaced with a constant value by using the corresponding context function prior to the evaluation of the respective condition.

A context constraint is a clause containing one or more context conditions. It is satisfied if (if and only if) all its context conditions hold."

Neumann and Strembeck state that constraints in general have multiple dimensions. As such, constraints can be static or dynamic; exogenous or endogenous; and authorization or assignment. These different dimensions are explained in this section.

2.1 Static and Dynamic Constraints

A static constraint refers to constraints that can be evaluated directly at the design time of an RBAC model (i.e., static separation of duties or SSD). Static separation of duty requirements can be enforced in the administration environment. For this reason, assignments that should

never occur can thus be prohibited. It has been pointed out that this feature can be very restrictive to business operations, especially in smaller organizations. However, it may still prove useful in cases where business rules that span an entire organization and stay constant over time must be expressed.

> Example: Two mutually exclusive roles must never be assigned to the same subject simultaneously (i.e., as above wherein the roles of a Physician and a Pharmacist should not be accessed simultaneously by the same user).

A dynamic constraint can only be checked at runtime according to the actual values of specific attributes or with respect to characteristics of the current session (i.e., dynamic separation of duties or time constraints). The objective behind dynamic separation of duty is to allow more flexibility in operations. Consider the case of initiating and authorizing payments. A static policy could require that no individual who can serve as payment initiator could also serve as payment authorizer.

> Example: A Resident can enter an order for a patient but cannot activate an order without a co-signature by their Attending Physician.

2.2 Endogenous and Exogenous Constraints

[Neumann/Strembeck] Endogenous constraints inherently affect the structure and construction of a concrete instance of an RBAC model. It influences the definition of the respective role-hierarchy since it further prohibits that two distinct roles to which these permissions are assigned can have a common senior role. Otherwise a common senior could acquire both (mutual exclusive) permissions and thereby violate the corresponding static separation of duties constraint.

- **Cardinality** occurs when there is a limit of a certain number of people who may be holding the permission at any one time.

  Example: At any given time per floor/unit, only one person may hold the permission of charge nurse.

- **Separation of duties** occurs when the same person cannot hold two related permissions at the same time.

  Example: A Staff RN permission vs. permissions for an RN supervisor.

Exogenous constraints are constraints that apply to attributes that do not belong to the core elements of an RBAC model, but are defined as “side conditions” for certain operations or decisions of an access control service. These include time constraints that restrict role activation to a specific time internal, or allow access operations for a particular resource only on a specific weekday.

- **Time-dependency** creates a time of day/time dependence on the person holding the permission.

Example: A person with banker permission can only e-sign money vouchers during regular business hours 9-5 (or a physician working in a clinic w/'set hours'). An example of no time-dependency is a physician working in the ER.

Example: An RN with Pharmacy privileges outside of regular Hospital Pharmacy Hours. At some sites, an RN may assume pharmacist 'review order' privileges (i.e., accepting an entered physician order to allow it to dispense) so that a patient may receive the medication - only during hours (time) when the pharmacy is closed (pharmacist is not available).

- **Location** creates a location requirement for the person holding the permission.

  Example: A physician or provider access to a remote hospital which is not their primary workplace.

  Example: An emergency room physician who covers another physician's shift/duty call during low patient census at multiple locations. A Physician's Assistant (PA) may be physically present, but the emergency room physician must override the PA order at the remote location in order for the order to process.

2.3 Authorization and Assignment Constraints

Beside the categorization as static/dynamic and endogenous/exogenous, constraints can also be subdivided in authorization constraints and assignment constraints:

—Authorization constraints are constraints that *place additional controls* on access control decisions. Thus, even if a subject is in possession of a permission that grants a certain access request, the access can only be allowed if the corresponding authorization constraints are fulfilled at the same time. For example, such constraints can be applied to implement access control policies based on access histories, as in Chinese Wall policies for instance.

—Assignment constraints are constraints that *control the assignment or activation of permissions and roles (e.g., maximum and minimum cardinalities or separation of duty constraints).* On the source code level, assignment constraints may be implemented through the same means as applied for authorization constraints (e.g., as an authorization constraint on the "assign role" or "activate role" permission). We think, however, that it is sensible to discriminate assignment and authorization constraints on the design level since both types address distinguishable intentions when engineering an RBAC policy. [Neumann Strembeck]

3 Constraint Process

3.1 Assumptions

Readers of this Constraint document should familiarize themselves with the HL7 Role Engineering Process. Each step requires the user to have the permission in order to accomplish the step. The compilation of sequential steps creates a scenario. A subject performing a scenario must possess all permissions that are needed to complete the different steps of this scenario. In turn, a task consists of one or more scenarios, and tasks are combined to form work profiles. A work profile comprises all tasks that a certain type of subject is allowed to perform. In a hospital environment different work profiles for physicians, nurses, and clerks are needed, for instance. In the role engineering process, work profiles are then used together with the permission catalog and the constraint catalog to define a concrete RBAC model. However, the scenario-driven approach presented in Neumann and Strembeck [2002] only provides general guidance for the sub process of defining (exogenous) constraints. This fact and our aim to specify and enforce context constraints in an RBAC environment led us to the definition of the process extension proposed.

Using Figure 3 as an activity diagram for the engineering (sub) process, the role engineering process as a whole and the engineering of context constraints is in essence a requirement for the engineering process. This process is based on goal-oriented requirements engineering techniques as found in van Lamsweerde Goal-Oriented Requirements Engineering: A Guided Tour [Neumann Strembeck].

3.2 High-Level View of the Constraint Process

3.2.1 Model Interrelations

Figure 1 illustrates how the Scenario Model and documents are related to the remaining components of the Constraint Catalog. Developing the Usage Scenarios (i.e., the Scenario Model) is the first step in the process.

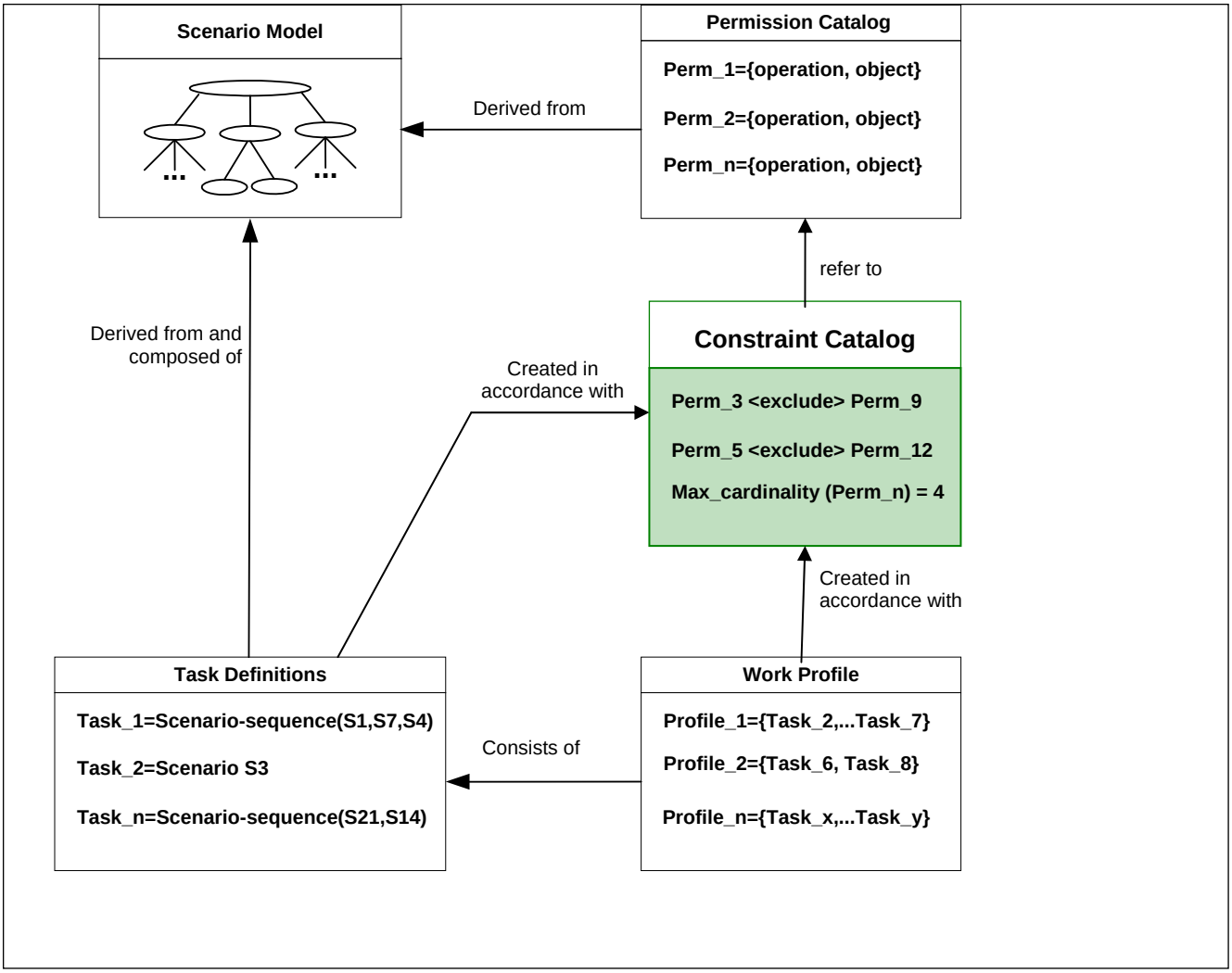

Figure 1: Interrelations of Scenario Model and Documents
(Adapted from [Neumann/Strembeck])

3.3 Process Steps [Neumann/Strembeck]

The following constraints sub-process will be applied to define the following types of constraints:

- User-role constraint – user cannot be assigned to two separate roles (e.g., a user cannot be assigned to both the Prescriber role and the Pharmacist role).
- Role-role constraints – users may have more than one role, but only one role may be active in a session at any given time (e.g., a user cannot activate the Prescriber role and the Pharmacist role within a session).

  Role-permission constraint – wherein a role cannot be assigned a certain permission (e.g., the 'write DNR Order' permission cannot be assigned to the Resident role).
- Permission-permission constraint – wherein permission(s) are in conflict or should not be assigned to the same role (e.g., The 'Order Medication' and 'Dispense Medication' permissions cannot be assigned to the same role).

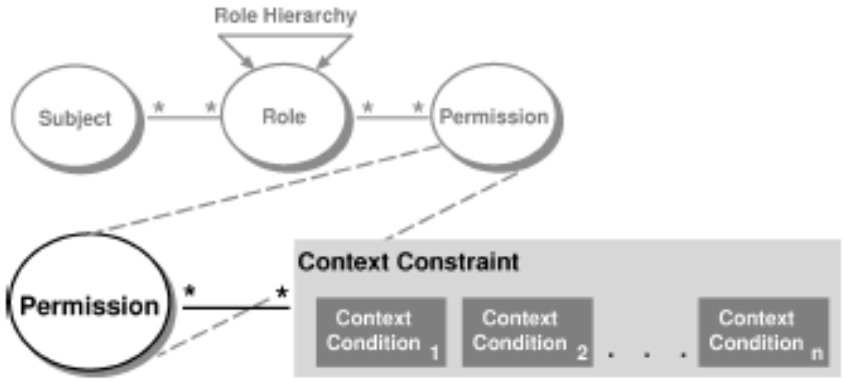

Figure 2: RBAC Permission with Context Constraint

3.3.1 Identification of Permission Constraints [Neumann/Strembeck 4.3]

Preliminary

Scenarios and the scenario model serve as the basis for the scenario-driven role engineering process [Neumann and Strembeck 2002]. The first step of the constraint engineering sub-process shown in Figure 3 is thus to *fetch the current scenario model*.

Process Overview

Review the permissions and identify the applicable constraints. For each of these constraints, create a record in the constraint catalog.

STEP 1 ➡ Review each permission and identify the applicable obstacle or constraint(s).

STEP 2 ➡ For each permission, record the associated constraint(s) if applicable (verify 'constraint' vs. 'business rule' constraint conditions and brief description; include factors which make it differ from a business rule).

STEP 3 ➡ Identify Constraint Type (cardinality, separation of duty, time, location)

STEP 4 ➡ Assign Constraint ID

Table 1 depicts an example of a constraint catalog. Each record in the catalog will consist of a permission ID, basic permission name or {operation, object} pair, and applicable constraint(s). Note: the constraint catalog will be populated after Scenario Model Refinement which is described in a subsequent section.

4 Constraint Table

Listed below are the legends for the healthcare permission table that follows.

ID (xy-nnn) Legend:

x	=	P (permission)
y	=	C (constraint identifier)
nnn	=	Sequential number starting at 001

Unique Permission ID - refers to the identifier assigned to the abstract permission name

Permissions are organized according to the following clinical tasks; [VHA Healthcare Permission Catalog]

- Order Entry (OE),
- Review Documentation (RD),
- Perform Documentation (PD) and
- Scheduling (SC).

Unique Permission-Constraint ID – refers to the identifier assigned to the permission constraint

Constraint Type – refers to the constraint definition as described in Table 1

Appendix A: Constraints Associated with Permissions EXAMPLE

Table 2 lists the definitions of the objects presented in Section 2:

Unique_Permission_Constraint-ID – Unique Constraint identification

Permission_Constraint_Description – brief description of constraint application on the listed permission(s)

Constraint_Type – definition of the constraint as applied to the corresponding permission

Permission_ID, Permission Name – corresponding unique Permission ID and name as described in the VHA Healthcare Permission Catalog to which the constraint is being applied.

Table 2: Permission Constraint Catalog EXAMPLE

Unique_ Permission _Constraint _ID	Permission_Constraint _Description	Constraint_Type	Permission _ID	Permission_Name
PC-002	A Resident may operate in ER as an Attending.	Location	POE-005	New/Renew Outpatient Prescription Order
			POE-006	Change/Discontinue/Refill Outpatient Prescription Order
			POE-016	Change/Discontinue Standing Order(s) PRN
			POE-017	New Verbal and Telephone Order
			POE-018	Change/Discontinue Verbal and Telephone Order
			POE-023	Sign Order(s)
			POE-028	Release Orders
			POE-028	Release Orders

Unique_ Permission _Constraint _ID	Permission_Constraint _Description	Constraint_Type	Permission _ID	Permission_Name
			PPD-001	New Progress Notes
			PPD-002	Edit/Addend/Sign Progress Notes
			PPD-010	Edit/Addend/Sign History and Physical
			PPD-013	Edit/Addend/Sign Consultation Findings
			PPD-030	Edit/Addend/Sign Discharge Summary
			PPD-041	Edit/Addend/Sign Encounter Data
			PPD-045	Edit/Addend Patient Acuity
			PRD-017	Review Progress Notes
PC-003	Surgery Service has rotating designation of Chief Resident (Only one Resident may have title (permission access or role) of Chief Resident in Surgery Service at any given time).	Cardinality	POE-023	Sign Order(s)
			POE-028	Release Orders
			PPD-001	New Progress Notes
			PPD-002	Edit/Addend/Sign Progress Notes
			PPD-010	Edit/Addend/Sign History and Physical
			PPD-015	New Surgical Report
			PPD-016	Edit/Addend/Sign Surgical Report
			PRD-017	Review Progress Notes
PC-004	Only one (1) nurse (RN) may be acting in the role of charge nurse on any given floor or ward.	Cardinality	POE-028	Release Orders

Unique_ Permission _Constraint _ID	Permission_Constraint _Description	Constraint_Type	Permission _ID	Permission Name
PC-005	Only one (1) physician may be acting as Chief of Staff at any given time.	Cardinality	POE-028	Release Orders
PC-006	Only one (1) physician may be acting as Chief of Medical Records at any given time.	Cardinality	POE-028	Release Orders
PC-007	In the event that a Hospital or Clinic Pharmacy does not have 24 hour service. A Charge Nurse may have access to some of the pharmacy override privileges (i.e., verify orders). During regular pharmacy hours, the Charge Nurse would normally not have these permission(s).	Time-Dependency	POE-005	New/Renew Outpatient Prescription Order
			POE-006	Change/Discontinue/Refill Outpatient Prescription Order
			POE-007	New Inpatient Medication Order
			POE-008	Change/Discontinue Inpatient Medication Order
			POE-028	Release Orders

Appendix B

Healthcare Information Technology Standards Panel (HITSP)

Security policy management includes the security rules to be enforced by the system, constraints on the rules and obligations of systems and conveying these to the security mechanism used in enforcing the rules. Security policy provisioning management includes granting privileges and attributes to users and making these known to the various components of the security system. Security policy management includes managening and instantiating security policy within the application security mechanisms. ISO 22600-1 & 2 provide the framework and models for security management used in this package.

Security policy enforcement or access control, deals with ensuring that users attempting to access system functions and data have the requisite privileges (granted and provisioned in privacy and security management). Access control considers all access control information needed to make and enforce an access control decision. ISO 10181-3 provides the framework and models for access control. It also defines the types of access control used in this package.

The following are the requirements derived from the American Health Information Community (AHIC) Use Case for this component:

1. Access Control policies are managed (created, modified, deleted, suspended or restored, and provisioned based on defined rules and attributes)
2. Data access policy is enforced
3. Data access policy bypass permission is established (granted) (Emergency Access/break glass)
4. Data access policy bypass is enforced (Emergency Access/break glass)
5. User data are located by an entity with the ability (privileges) to search across systems
6. Protected data are accessed based on user permission for data access
7. Protected data are modified, updated or corrected by identified users
8. Selective protected data are blocked from users
9. Requests for changes to protected data are made by users to providers/sources of data
10. Obligations can be used to full access permissions and actions

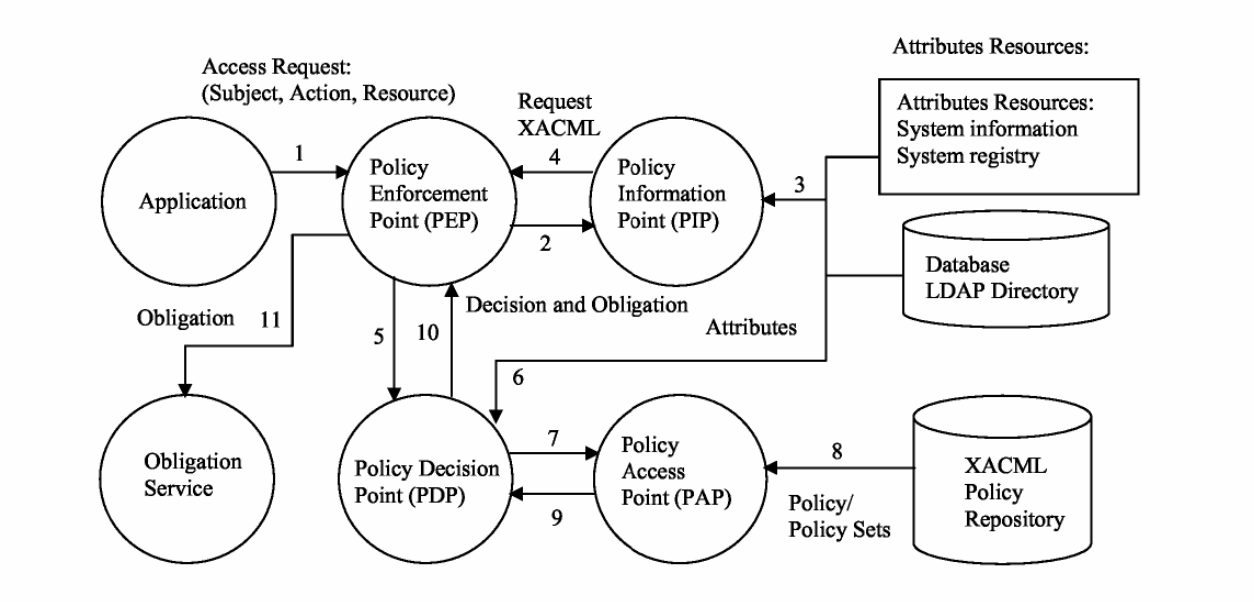

Figure 3: XACML Components

XACML (eXtensible Access Control Markup Language) (is a general-purpose language for specifying access control policies [Pro04]. In Extensible Markup Language (XML) terms, it defines a core schema with a namespace that can be used to express access control and authorization policies for XML objects. Since it is based on XML, it is, as its name suggests, easily extensible. XACML provides features that make it possible to support a broad range of policies [Oas06]; it provides the capability to request a specified action within a system using a standardized syntax, and then receive one of four replies:

- Permit – action allowed
- Deny – action disallowed
  - Indeterminate – error or incorrect/missing value prevents a decision
  - Not Applicable – request cannot be processed.

XACML's standardized architecture (Figure 2) for this decision-making uses two primary components: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). The PEP constructs the request based on the user's attributes, the resource requested, the action specified, and other situation-dependent information through PIP. The PDP receives the constructed request, compares it with the applicable policy and system state through the Policy Access Point (PAP), and then returns one of the replies specified above to the PEP. The PEP then allows or denies access to the resource. The PEP and PDP components may be embedded within a single application or may be distributed across a network.

To make the PEP and PDP work, XACML provides a policy set, which is a container that holds either a policy or other policy sets, plus (possibly) links to other policies. Each individual policy is stated using a set of rules. XACML also includes methods for combining these policies and policy sets, allowing some to override others. This is necessary because the policies may overlap or conflict.

Transferring XACML Policies

Policies may need to be transferred from one entity to another in a Privilege Management Infrastructure. Some of the situations where this is required are:

a) A PDP evaluates a policy that references other policies by name. The other policies must be fetched from a Policy Administration Point (PAP) when required for evaluation.

b) A PDP may need to obtain its "root" policy from the enterprise Policy Administration Point as part of configuration.

c) A resource may be transferred between security domains and the source domain may transfer a policy for protection of the resource that the destination domain is responsible for enforcing.

d) Multiple sites may need to use common policies, even though their PDPs are local for performance reasons. These policies need to be transferred from the central Policy Administration Point to each site's PDP.

While XACML defines a policy language, it's designed to be one component in an overall authorization system. It relies on other components to provide mechanisms for verifying that policy instances were issued by a trusted Policy Administration Point for protecting the integrity and confidentiality of instances of policies, and for protocols used to query for and respond with policy instances. XACML has been integrated with the Organization for the Advancement of Structured Information Standards (OASIS) Security Assertion Markup Language (SAML) Version 2.0 as one way of providing these necessary functions. SAML may be used with XACML to protect Access Control Information attributes as well as policies. The following diagram illustrates the integration of SAML and XACML. [HITSP]

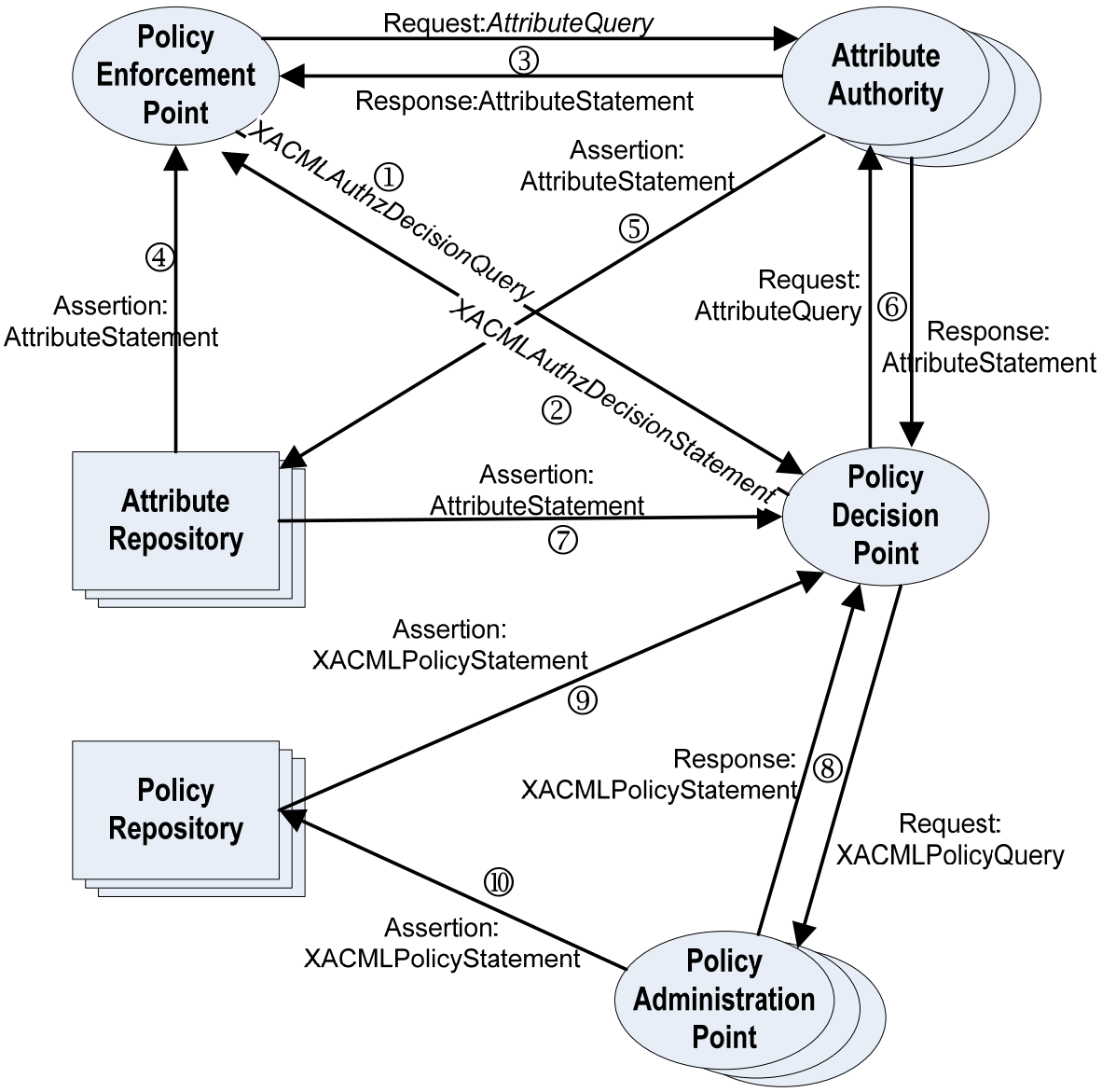

Figure 4: Using SAML 2.0 to transport XACML
(Used by permission of OASIS)

Appendix C: References

[ANSI] American National Standard for Information Technology - Role-Based Access Control, ANSI INCITS 359-2004, 2004

[ANSI] An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments; ACM Transactions on Information and System Security; Neumann and Strembeck

[ASTM] ASTM E 1986-98: Standard Guide for Information Access Privileges to Health Information

[HITSP] Healthcare Information Technology Standards Panel HITSP Manage and Control Data Access Transaction Package, V0.2 May 16, 2007

ISO 10181-3-00: Security Frameworks for Open Systems: Access Control Framework; also available as ITU-T X.812: 1995

ISO TC 215/WG4/N ANSI Health informatics – Privilege management and access control – Part 1: Overview and policy management, 2004-01-08

[Neumann-Strembeck] An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments; Received November 2003; revised March 2004, April 2004 and May 2004; accepted May 2004

OASIS, Extensible Access Control Markup Language (XACML) v2.0, February 2005

OASIS, XACML Profile for Role-based Access Control (RBAC): Committee Draft 01 (normative; 13 February 2004)

VHA/IHS Role-based Access Control Task Force Charter Version 2.0, 17 September 2004

Latest versions of the following documentation can be found on: www.va.gov/RBAC

VHA Role-based Access Control Task Force Project Management Plan (PMP)
VHA Role-based Access Control Task Force Structural Roles
VHA Role-Based Access Control, Functional Roles
VHA Role-Based Access Control, Permission Catalog

---

[i] HL7 RBAC Role Engineering Process v1.1, November 2005
[ii] HL7 Structural Roles v1.0, September 2006
[iii] M Strembeck and G. Neumann, An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments; Received November 2003; revised March 2004, April 2004 and May 2004; accepted May 2004
[iv] M. Strembeck and G. Neumann, An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments; ACM Transactions on Information and System Security, Vol. 7, No. 3, August 2004.
[v] Ibid
[vi] D. Richard Kuhn, Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems. 1997