An XML Security Framework that Integrates NIST RBAC, MAC and DAC Policies

Alberto De la Rosa Algarin
Ph.D. Proposal
Major Advisor: Dr. Steven A. Demurjian
Associate Advisors: Dr. Jinbo Bi, Dr. Swapna Gokhale
Dr. Xiaoyan Wang

Abstract
Today’s applications are often constructed by bringing together functionality from multiple systems that utilize varied technologies (e.g., APIs, web services, cloud computing, data mining) and alternative standards (e.g., XML, RDF, OWL, JSON, etc.). Most such applications, constructed as a meta-system (systems of systems), achieve information exchange (IE) via the eXtensible Markup Language (XML), the de facto standard format used in many domains such as library repositories, collaborative software development, health informatics, etc. In such a situation, the security challenge is to integrate the local security (existing systems) and their associated access control models into a global solution for the application being constructed and deployed. In this proposal, we present an XML security framework that operates across various contexts (e.g., information exchange, databases, web services, etc.) while simultaneously integrating and enforcing local and global security policies for role-based access control (RBAC), mandatory access control (MAC), and discretionary access control (DAC). Our solution approach proposes an XML security framework that extends the Unified Modeling Language (UML) with new diagrams to model RBAC, MAC, and DAC for XML, and in turn is then able to generate enforcement policies in eXtensible Access Control Markup Language (XACML) that target XML schemas and instances for any application. The framework is also capable of generating a global enforcement security policy in XACML for the new application by leveraging both the UML-XML diagrams and associated mapping algorithms to facilitate the secure interactions from applications that include information exchange, databases, etc. As a result, the XML security framework requires the inclusion of a secure information engineering process that provides the guidance and structure to accomplish information usage and exchange for a complex application. To demonstrate the feasibility of the XML security framework, we utilize applications from the health care domain to provide an actual scenario of the use of medical health information technologies systems by patients and providers on different platforms (mobile and PC). Using this as a basis, we are able to construct an XML security framework that is able to operate across a wide range of information technology systems, regardless of domain, that utilize XML or other equivalent document structures for functionality, where information must be examined, modified, and exchanged.
1. Introduction

Today’s world is dominated by systems with a wide range of technological approaches (e.g. APIs, web services, cloud computing, data mining, etc.) that strive to effectively utilize, share, and exchange information as an application is constructed as a meta-system (systems of systems), with the new application interfacing with multiple technologies, comprised of many interacting components. In such an environment, one major challenge is to ensure that local security policies (of constituent systems) and access control mechanisms are satisfied not only when the application accesses a single system, but also when considered from a higher-level perspective. That is, an application’s security is the combination of the security that must be attained by each constituent system. This raises a number of critical questions. What happens when systems have different access control models (e.g., role-based, mandatory, discretionary, etc.)? What happens if systems have different implementations of the same access control model? What happens when security privileges of individual systems are in conflict with one another? How do we then reconcile these conflicting local security policies to support an application that needs access to multiple systems? Is it possible to define a global encompassing security definition and enforcement process that provides a level of guarantee to the local security policies from both access control model and enforcement perspectives? As today’s applications continue to become more and more complex, interacting with many other systems using varied technological paradigms and different access control models, there will be a need to provide some degree of assurance that security for the application (global) satisfies the sum of the parts (local security of constituent systems). Information usage, sharing and exchange have increased exponentially, in domains such as biomedical, health informatics, library repositories, collaborative software development, etc. These domains present security challenges that have not been yet addressed, neither in a limited scope (local security and associated access control models), nor across a broader scope of multiple formats and meta-systems (global security).

The usage, sharing, and information exchange of information has increased across domains such as library repositories, collaborative environments, health care, etc., accompanied by the development of generic document structures that promote and simplify interconnection between systems and applications. As a process, information exchange (IE) involves more than one system with one or more applications, with the purpose to either provide a consolidated view of data stored in different repositories, or the transmission of data from one repository to another so that consistency of information is preserved. With the introduction of more complex systems and applications (e.g. meta-systems, or systems of systems), the eXtensible Markup Language (XML) has been predominately utilized for packaging data before transmission. XML provides an extensible structure for documents that is machine readable and easy to process, combined with the ability to design and develop schemas for document instance validation. The resulting creation of XML standards for representations and their use by meta-systems dictate the need for a new layer of functionality: the ability to secure and enforce access control of the information from local data sources in a global manner. As shown in Figure 1, a meta-systems are built as a combination of different health information (HIT) systems (right hand side of figure), where each one can have their own local security policies; the example is focused on a health care with an electronic health record (EHR) PatientOS utilized by a hospital, clinic or medical provider, a personal health record (PHR) Microsoft Health Vault for patients to manage their own information, and a patient portal (PP) that provides a means for appointments, referrals, and interactions with a medical provider. When local policies of systems utilized by a meta-system collide and differ in security capabilities, access control models, and/or permissions, there is a need to integrate and evaluate which capability/model/policy takes precedence and maintains a level of security necessary for the application at a global level (left hand side of Figure 1). Our main objective in this proposal is an XML security framework for information usage, sharing and exchange that is able to provide a means to integrate local security policies and their associated access control models within a global context to allow access by application’s to be more effectively controlled.

1.1. Motivation for Access Control Models

In support of the proposed XML security framework, we seek to allow the customization and filtering of XML documents (instances) by controlling the ability of authorized users to access privileges based on role, security classification level, or via a delegation of authority. Towards this end, from an access control perspective, we intend to support the security definition on the XML schemas utilized for information sharing and exchange using: National Institute for Standards and Technology’s\(^2\) (NIST) standard role-based access control (RBAC) (Ferraiolo et al. 2001, Liebrand et al. 2002) to allow privilege definition based on what a user needs to accomplish his/her tasks; mandatory access control (MAC)\(^3\) that will tag aspects of the data with classification levels and user clearance levels to access information; and discretionary access control (DAC)\(^4\) to allow authority to be delegated from user to user in dynamic situations that require someone to acquire permissions that they would not normally possess. Why are the three major access control models important? The answer is that they are targeting across the spectrum of the possible domains, from e-commerce to health care, banking, infrastructure and national defense. The end result will be the ability to deliver XML instances to users (or applications) that adhere to the exact combination of access control that is required. As shown in the right hand side of Figure 1, the proposed XML security framework is comprised of multiple access control mechanisms, the algorithms and infrastructure for permission definition that are used for policy generation, when given the local security policies and the health information technology (HIT) application(s) that provide the schemas to be enforce. The bottom of the figure shows the generated policies that are filtered for delivery to the end user. For a health care application, the HIT Data sources are EHRs, PHRs, PPs, etc. (as in Figure 1), while the HIT application can be for patients who want to monitor their chronic conditions, for clinical researchers that are seeking to access de-identified data from a clinical research data warehouse, and for providers that are seeking to improve care through patient interactions and clinical decision support.

For the purposes of this proposal, we focus on the attainment of multiple access control models (RBAC, MAC, and DAC), which in health care would support the definition of security policies on an XML document (patient data) at the schema level can then be used to specify (allow or deny) different permissions on certain portions of the schema’s instances. This will allow the same instance to appear

differently to specific users (patients and medical providers) acting in a chosen role at different times, with the ability to severely limit access to data (using MAC classifications to protect mental health data that is more restrictive under HIPAA) while simultaneously allowing authority to be delegated (so that a provider in an emergent situation can get the authority to access a system s/he has not previously been asked to use). To accomplish this, we leverage a secure information engineering process that promotes the consideration of security at the early stage of the software development and continued throughout the process. In health care, the usage of an XML schema for an HIT application requires a security framework for that XML schema that allows the design, implementation, and deployment of an enforceable security policy to allow access to XML instances to be precisely controlled by role, security level, or delegation authority. The definition of security at the XML schema level separates the security from the XML instances, which avoids the overhead required when updating security policies that would otherwise be embedded in XML instances. Figure 1 demonstrated that data will be obtained from different systems that have their own local security policy, and in some cases, there may be the need to translate the data from a local source into an XML format in order to define the local security policy and allow the information to be shared, as shown by the XML-C yellow diamond in Figure 1.

1.2. Motivation from the Healthcare Domain

Security for health care goes well beyond the needs of compliance of the Health Insurance Portability and Accountability Act (HIPAA), which provides a set of security guidelines in the usage, transmission, and sharing of protected health information (PHI); protecting personally identifiable information (PII), including names, addresses, accounts, credit card numbers, etc.; encryption of PHI and PII data and its secure transition (e.g. SSL); extensive usage of standards such as the Health Level Seven’s (HL7) clinical document architecture (CDA) (Dolin, Giannone & Schadow 2007) and the Continuity of Care Record (CCR) for storage of administrative, patient demographics, and clinical data; and the leveraging of XML for a wide range of health care standards (CDA, CCR, LOINC, SNOMED, UMLS). Instead, to attain security for digital health care, we will need all of these underlying technologies and standards that must be coupled with a strong understanding of the way that health care data is utilized by a wide range of stakeholders, including patients, providers, researchers, etc. In HIT, CDA and CCR come together in various systems such as EHRs for organizations to manage patient data, PHRs like Microsoft HealthVault for patients to track their health history; and, PPs where a patient can make appointments, order refills, request referrals, etc. As CDA and CCR XML documents are circulated among various systems and made available to particular users with specific needs, we must expand security from each individual local system to a focus that is more expansive in controlling a CCR document and its content, particularly for HIE, and in the rapidly emerging mobile healthcare domain, where patients manage personal health information for chronic diseases and need to securely access information and authorize its exchange with medical providers via mobile applications, EHRs, secure emails, or other means. This will require document-level access control of XML schemas to allow XML instances to appear differently to authorized users at specific times based on criteria that include, but are not limited to, a user’s role, time and value constraints on data usage, collaboration for sharing data, delegation of authority, etc. A first view of the overall architecture for the XML security framework was given in Figure 1.

The emerging trends in health care are all strongly tied to information usage, sharing, and exchange, across a wide range of medical initiatives related to patient care. First, a Patient Centered Medical Home (PCMH) that is targeted toward coordinating chronic conditions and optimizing care plays by interacting with multiple stakeholders (e.g., specialists, therapists, visiting nurses); in this situation, there may be a need for the lead provider to access information in other EHRs, PHRs, PPs, etc., in a timely manner.

---

6 Continuity of Care Record, http://www.astm.org/Standards/E2369.htm
Second, accountable care organizations (ACOs)\(^9\) that bring together larger groups of providers, clinics, and hospitals in an effort to give coordinated care to Medicare patients, which is particularly for chronic conditions, attempts to eliminate duplicate test and procedures. Third, secondary use (SU)\(^10\) of clinical data that allows both providers and researchers to analyze specific diseases and their treatments across a large patient base via a CRDW, seeking events such adverse drug reactions, infection monitoring, disease monitoring in a larger population. Fourth, personalized medicine (PM)\(^11\) which is targeting to treat individual based on their unique medical profiles which might include specific types of diseases and focus on the use of a patient’s genomic information. In support of all of these initiatives, secure information engineering is vital to insure that the correct data is available at the specific time by the appropriate stakeholder. Note that what is shared in an HIE implementation is determined by the institute (practice, clinic, hospital, etc.) that owns the data; it doesn’t mean all data is shared and the data to be shared is often offloaded into another server particularly intended for that purpose so that there is no impact on the real-time usage of an EHR in a care setting.

Figure 1 characterizes the involved HIT systems, XML medical standards, and end-user patient applications, to provide an infrastructure for PCMH, ACO, SU, and PM. The intent is to examine this infrastructure from three viewpoints: the reconciliation of security (both local and global) within the environment to insure that the required clinical data reaches provides involved in PCMH and PM; the availability of de-identified patient data for providers and clinical researchers, in support of ACO and SU, that can be used for data analysis, mining, and clinical decision support in order to learn about what works and what doesn’t in terms of treatments of various illnesses and diseases; the facilitation of the first two perspectives via the use of the eXtensible Markup Language and all of the associated standard, and terminologies. To demonstrate document-level security aspects of the work in this proposal, we use a health care and CCR case study of the Personal Health Assistant (PHA) application, which consists of two related mobile health applications for health information management, one that supports a patient/healthcare provider scenario where the patient can authorize a subset of the PHI (CCR instance) stored in MSHV to different providers at different time, and a second that allows providers to select and view the authorized PHI on a patient-by-patient basis. The proposed XML Security Framework is aimed towards a comprehensive approach to security at global and local levels operating within an environment that is driven to use, share, and exchange, and having to deal with the distributed nature of repositories, the fragmentation of patient data across these systems, and differences on sharing and security policies.

1.3. A High-Level View of our Approach

In this proposal, the initial high-level view of our XML security framework (De la Rosa Algarín, A. et al. 2012) for information usage, sharing, and exchange from Figure 1 can be refined as given in Figure 2. Security (via access control models) is defined at the schema level and realized at the instance level through the creation and generation of eXtensible Access Control Markup Language (XACML)\(^12\) policies (as shown by the XML Security Extensions to UML box), and demonstrated via a case study of health care composed of a set of HIT applications. Figure 2 provides the ability for a designer to pick and choose to create local XACML security policies (XML schema\(_{EHRa}\), XML schema\(_{EHRb}\), XML schema\(_{PHRa}\), and policies for meta-systems (XML schema\(_{Meta-Systema}\), and XML schema\(_{Meta-Systemb}\)) for applications (in HIT, these would be EHRs, PHRs, PPs, etc.). The resulting XML security framework achieves granular security by taking the generated XACML policies and enforcing them on the targeted XML schemas and instances via a combination of schema modeling, policy generation, and policy integration, as shown in the second horizontal box in Figure 2. These policies are supported in access control models (RBAC, MAC and DAC), as shown in the third horizontal box in Figure 2, and provide a

\(^{10}\) SU, http://www.ncvhs.hhs.gov/050726p5.pdf
\(^{11}\) PM, http://www.personalizedmedicinecoalition.org/
\(^{12}\) OASIS XACML, https://www.oasis-open.org/committees/xacml/
custom or filtered view based on role, user clearance against data classification, and delegated authority. Our approach provides separation of security concerns to tackle the challenge of changing security policies that can apply to multiple of XML instances. In support of this framework, we leverage our prior work on secure software engineering using UML (Pavlich-Mariscal, Demurjian & Michel 2008), which proposed the creation of new UML-like diagrams for the NIST RBAC, MAC and DAC models that were consistent with an object-oriented design paradigm. In our proposed work, we adopt these ideas but apply them in order to create new UML diagrams that can capture XML schemas, roles, security levels, and delegation rules that allow privileges on an application’s XML schemas (see the fourth horizontal box in Figure 2) to be allowed and/or denied to different users different times. All of these applications require that the XML that is delivered to a user be restricted by role (a filtering of the content of the instance), by security level (classification of data vs. clearance of user), and by delegation of authority, in order to insure that only the authorized information is provided to the user; these are referred to as security enforced (SecE) instances that have gone through potentially multiple levels of security until the instances has reached a state where it can be safely presented to a user. The end result, shown in the bottom horizontal box in Figure 2, are generated security policy models for the meta-systems.

By achieving this security framework for XML schemas and instances, leveraging three major components (UML, XACML and access control models: RBAC, MAC and DAC), we present a model-driven approach to document-level security and policy integration. Our intent is to bring together the processes of software engineering and knowledge engineering overlaid with a security emphasis, resulting in a broader definition of secure information engineering, a term that we mean to denote a comprehensive process that leverages both software and knowledge engineering in order to emphasize on their information to be secured. In turn, designers can follow a secure information engineering process model to achieve both the modeling and security definition of an application. In the perspective of IE, the document-level security framework for XML produces local security policies that act on the system being

![Figure 2: High-Level View of Proposed XML Security Framework.](image-url)
engineered and developed. Utilizing the extensions to UML with respect to NIST RBAC, MAC and DAC, we can then integrate these local policies into an all-encompassing meta-policy that can be utilized by a newly developed meta-system (system of systems) as shown in Figure 1.

1.4. Research Objectives and Expected Contributions

From the research perspective, the problem of secure data sharing and exchange is achieved by realizing the following expected contributions.

A. **UML Extensions for XML Security**: The expected contribution of this aspect of our work is to represent an XML schema as an UML-like diagram which is augmented with security features that capture the core aspects of the three main access control models: RBAC, MAC, and DAC. To date, we have developed two new UML XML diagrams: the XML Schema Class Diagram (XSCD) to represent an XML schema as an UML-like diagram; and, the XML Role-Slice Diagram (XRSD) to represent RBAC security augmented version. Representing the XML schemas with these diagrams permits us to tackle document security from an information engineering perspective. This provides several benefits, including a more consistent process towards secure information engineering, and facilitating the secure policy generation process by utilizing modeling artifacts that contain all the pertinent information for the security policy. We are currently exploring the other necessary UML extensions for supporting MAC and DAC.

B. **Security Policy Generation**: The intent of this objective is to leverage XML and UML for an automatic XACML Policy Creation. This is achieved via a mapping from new UML XML diagrams for RBAC (defined), DAC and MAC (to be defined) and the XACML Policy structure, which can be used as part of an enforcement mechanism to assure security. Towards this goal, we have a mapping relation between the XRSD elements and an XACML instance’s elements (Policy, Subject, Action, Resources), as well as an algorithm for the automatic creation of XML schema targeting RBAC policies. The intent is to extend this work to include MAC and DAC.

C. **Security Policy Integration (in progress)**: The expected contribution of this aspect of our work is to provide UML extensions for locally acting security policies (which can enforce RBAC, MAC or DAC security requirements) for policy integration. Towards this goal, we have created two new UML constructs: the Policy Slice Diagram (PSD), which acts as a generalized version of the XRSD (in the case of RBAC); and the Security Policy Schema Set, which represents a locally acting security policy as a UML class diagram. We have also introduced a new concept, the Master Role Index (MRI), which is an XML schema that validates instances that contain all relevant role data for policy integration. We have also developed an integration process that is performed from a software engineering perspective, yielding conflict free RBAC security policies. The intent is to extend this work to include MAC and DAC enforcement.

D. **Secure Information Engineering (in progress)**: This contribution provides a secure information engineering process to meta-system creation by focusing on the perspective of the data to be secured. As shown in Figure 2, this secure information engineering process involves the global security model and policy integration by modeling the XML schemas to be secured with the respective access control models (Contribution A), the generation of locally acting security policies (Contribution B), and the integration of the constituent local policies (Contribution C) into an enforcement meta-policy for a newly developed meta-system. The end result of this contribution is a formal engineering process that a software designer or developer can follow in order to ensure security and information assurance.

This dissertation will explore the potential impact of these contributions via the health care domain which provides an excellent mechanism to justify the need for the XML security framework in HIT and HIE.
1.5. Research Progress to Date

In support of the proposed XML security framework, a number of articles have been published (or accepted):


or are under submission:


1.6. Overview of the Proposal

The remainder of this proposal has 6 sections. Section 2 provides the Unified Modeling Language (UML), access control models (NIST RBAC, MAC and DAC), the eXtensible Markup Language (XML), the eXtensible Access Control Markup Language (XACML) and our security enforcement prototype. In Section 3, we present the portion of the XML security framework that involves extensions to UML with XML diagrams to model RBAC via the XML Schema Class Diagram, which serves to model the XML schema as a UML diagram; the XML Role-Slice Diagram, which serves as a NIST RBAC augmented representation; and, and, the Master Role Index developed as an XML schema with the intent to provide structure and quantify the available roles. These UML representations of security requirements and access control policies are defined at a granular level from the perspective of the targeted XML schema. In Section 4, we detail our security policy integration model and approach which is represented by the box of Generated Security Policy Models in Figure 2, and includes UML extension to model security: the Policy Slice Diagram which serves as a representation to a set of security rules with respect to a role; and, the Security Policy Schema Set which is the representation of a local system’s security policy. Next, Section 5 utilizes the recurring example of the health care domain to present the approach for Local policy integration into a Global policy by using name equivalence, structural equivalence and conflict resolution. Section 5, also presents a set of mapping relations that produce an enforcement meta-policy in XACML. We conclude this proposal by discussing the research timeline of the remaining work for the dissertation in Section 6, and reviewing the expected research contributions in Section 7.
2. **Background**

In this section, we present background information regarding the healthcare requirements and challenges with regards to the XML security framework. First, in Section 2.1, we review key healthcare requirements and challenges. Section 2.2 examines the Unified Modeling Language (UML) and its meta-model. Section 2.3 reviews the three access control models, RBAC, MAC, and DAC. Section 2.4 examines the eXtensible Markup Language (XML). Section 2.5 reviews the eXtensible Access Control Markup Language (XACML). Finally, Section 2.6 briefly examines the PHA application that is used in examples throughout the proposal.

2.1. **Healthcare Requirements and Challenges**

The distributed nature of the HIT systems, repositories, and usage present challenges that, though not unique to the domain, represent major obstacles to overcome in order to achieve a proper sense of security that protects and maintains protected health information (PHI) and personally identifiable information (PII). This distributed nature also pushes systems and application developers to build new products as meta-systems (systems of systems) that rely on the information found across heterogeneous repositories (such as EHRs, PHRs, etc.). Thus, it is not only sufficient to provide robust security at the document-level from the data’s perspective, but it is also necessary to consider the way that the different security definitions from individual systems (constituents to the information exchange process) govern the locally stored data (from the repository’s perspective). Thus, to achieve proper security, the global security targeting a newly developed meta-system must comply with the sum of its parts (the local systems).

We recognize that in order to do this, and considering the different document formats, resources, security policies, users and roles; we must provide document-level security at the schema definition and instances, integrate security policies to form a meta-policy that resolves all conflicts, and a bidirectional mechanism of document format conversion that can take in any structure (e.g., JSON, RDF, etc.) and produce a unique and correct XML serialization (including a schema that validates the instances). Towards this end, we will utilize UML, to model XML schemas, the security requirements, and eventual policy generation; XML as the format of choice to secure; and, XACML policies as the resulting instances for enforcement.

2.2. **Unified Modeling Language (UML)**

The Unified Modeling Language (UML) is a general-purpose modeling language for object-oriented systems (Fowler 2004). Currently managed by the Object Management Group (OMG)\(^\text{13}\), UML can be used throughout the software development cycle by combining data, business and object modeling. UML diagrams can exhibit two views of a system’s model: the *structural view*, which represents the objects, attributes, operations and relationships in the system; and, the *behavioral view*, which represents the collaboration among objects and changes to the internal states of objects. Towards this purpose, UML provides different kinds of diagrams. *Structure diagrams*, which include the class diagram, component diagram, composite structure diagram, deployment diagram, object diagram, package diagram, and profile diagram, focus on the representation of components of the system; *behavior diagrams*, which include the activity diagram, UML state machine diagram, and the use case diagram, focus on the series of events that must happen in the system; and, *interaction diagrams*, which include the communication diagram, interaction overview diagram, sequence diagram, and timing diagram, focus on the data- and control-flow between the components of the system being modeled.

\(^\text{13}\) Object Management Group (OMG), http://www.omg.org/
The UML language can be extended via the use of the meta-model architecture developed by OMG. This meta-model architecture, called the Meta-Object Facility, consists of four layers. As shown in Figure 3, the M3 layer consists of the meta-meta model. M2-models are built using the M3 language. In turn, M2-models describe the elements of the M1-layer, while the M1-models describe the elements of the M0-layer (the runtime instance of the modeled system). Due to the inclusion of UML into ISO as a standard for software systems, several tools (and development environments) exist to aid in UML modeling, including: ArgoUML\textsuperscript{14}, StarUML\textsuperscript{15}, Eclipse\textsuperscript{16}, Visual Studio\textsuperscript{17}, NetBeans\textsuperscript{18}, and others.

![Figure 3: UML Meta-Object Facility Architecture consisting of the M3, M2, M1 and M0 Layers.](image)

### 2.3. Access Control Models (RBAC, MAC and DAC)

In the NIST RBAC model, permissions are assigned to roles, which are then assigned to users, shown in Figure 4 where a user can perform any of permissions assigned to the role s/he exhibits. NIST RBAC contains four reference models, as shown in Figure 4. RBAC\textsubscript{0} allows for policies to be denied at the role level instead of the individual level. To handle role hierarchies, RBAC\textsubscript{1} allows for parent roles to pass down common privileges to children roles so that permissions high in the hierarchy can be inherited by the roles below, and specific permissions are associated with roles that act as leafs in the hierarchy. RBAC\textsubscript{2} provides definition of constraints, such as separation of duty (SoD) and cardinality.

As an example, consider the scenario of a group of health care professionals reading sensitive patient data. The reading of such data is definitively allowed under certain conditions. SoD ensures that the authorization role that grants permissions exists as a different entity to the other roles. This ensures that roles are not allowed themselves to view sensitive data they would otherwise have no authorization to. Mutual exclusion ensures that two or more specific roles may not be assigned to any particular user, enforced by restrictions put in place by the cardinality constraint (the number of users/permissions getting assigned to a particular role). RBAC\textsubscript{3} introduces the concept of sessions that represent the lifetime of a particular user, role, permission and their association for a dynamic runtime application.

---

\textsuperscript{14} ArgoUML, http://argouml.tigris.org/
\textsuperscript{15} StarUML, http://staruml.sourceforge.net/en/
\textsuperscript{16} Eclipse, http://www.eclipse.org/modeling/mdt/?project=uml2
\textsuperscript{18} NetBeans, http://netbeans.org/projects/uml/
In the MAC model (Bell, La Padula 1976), security policies are defined and set by the security administrator. This means that users are not able to change their security attributes (unlike the DAC model). In this model, access to objects (e.g., segments of an XML document, tables in a database, etc.) by subjects (e.g., users, processes in a system, etc.) is granted based on the security definitions on the targeted object (exhibited via tags) and the credentials granted to the user. There have been several implementations of the MAC model, including FreeBSD (as part of the TrustedBSD project), Windows (with Windows Vista), Trusted Solaris and the NSA SELinux research project.

In the DAC model, access to objects is permitted or denied based on the subject’s identity or the group they are part of. As a difference with MAC, in DAC users are capable of passing permissions to other users (hence the term discretionary access control). There are two broad categories of implementations of the DAC model. The first, with owner, define that every object contains an owner that controls the permissions that target it. This means that the object owner controls which users can access the object, and which permissions are allowed. The second, with capabilities, permit users to delegate (transfer) their permissions (access) that target certain objects with other users.

2.4. eXtensible Markup Language (XML)

The eXtensible Markup Language (XML) facilitates the exchange of information between disciplines, offering a flexible means to collect and transmit data between information systems and platforms. XML aims to provide a common, structured language that is independent of the system that utilizes it. XML supports information to be hierarchically structure and tagged, with the tags offering the capabilities to represent semantics of the information. By allowing the definition of XML schemas, the language offers the ability to define standards that server as both the blueprint and validation agents for instances seeking to comply and be used for information exchange purposes. Towards this end, the main mechanism behind XML schemas is the XML Schema Definition (XSD), which follows the XML Schema language.

As an example (Figure 5), an XML schema can be composed of multiple xs:complexType, xs:simpleType, xs:sequence, xs:element, etc., and these can be combined and nested in any way to form a more encompassing xs:complexType. This is a characteristic, which is shared with UML, permits developers or standard proposing agencies to determine constraints (e.g. minimum of maximum amount of occurrences via minOccurs and maxOccurs), the allowed data type for the value of an element, and others. The schema's role is to describe and define the domain model from the perspective of the data utilized, including the type-level characteristics that instances must follow in order to be valid.
2.5. eXtensible Access Control Markup Language (XACML)

In the same way that XML aims to provide a common, structured language for information exchange among heterogeneous systems, the OASIS eXtensible Access Control Markup Language (XACML) aims to define a common language and processing model from the perspective of access control policies. This would then permit a level of security interoperability among the heterogeneous systems. The XACML schema provides various elements and a general structure for the design and development of access control (security policies). These elements (as shown in Figure 6) include the PolicySet, the Policy and the Rule. An XACML PolicySet is utilized to make the authorization decision via a set of rules in order to allow for access control decisions. A PolicySet can contain multiple Policy structures, and each Policy contains the access control Rules. As a result, the Policy structure acts as the smallest entity that can be presented to the security system for evaluation. The collection of Policy structures is contained in a PolicySet, combined via an algorithm specified by the PolicySet’s PolicyCombiningAlgId attribute that targets the particular XML schema. The XACML specification defines four standard combining algorithms: Deny-overrides, Permit-overrides, First-applicable, and Only-one-applicable.
2.6. Personal Health Assistant and Enforcement Prototype

Over the past few years, we have been developing health care applications for medication management and medication reconciliation. Personal Health Assistant (PHA) is a test-bed, not publicly available, mobile application (for Android\(^{19}\) and iOS\(^{20}\)) for medication management that allows: patients to view and update their personal health record stored in their Microsoft HealthVault (MSHV) account and authorize medical providers to access certain portion of PHI; and for providers to obtain the permitted information from their respective patients that they have been authorized to view. The patient version of PHA (lower left of Figure 7) allows users to perform a set of actions regarding their health information. Users can view and edit their medication list, allergies, observations of daily living (ODLs/Wellness Diary) and set security policies for read/write permissions on their tied providers by role. Security settings can be set at a fine granular level, and each provider gets view/update authorizations to the different information components available in PHA. Using this information, XACML policies are generated and stored in the patient’s MSHV account. The provider version of PHA (lower right of Figure 7) allows the users (health professionals or medical providers) to view and edit the medical information of their patients as long as there are permitted to do so as dictated by the security settings created by the user (patient). By logging in with their personalized account, a list of patient tied to the provider is displayed. Upon selecting a patient, the information associated with that patient can be viewed and updated.

The personal health record, Microsoft HealthVault (MSHV), acts as the PHA’s data source (top of Figure 7) where a user can save demographic and health information, including medications, allergies, procedures, conditions, etc. MSHV stores this information in a proprietary format that can be exported via as XML structures that can be turned into a CCR schema compliant XML instance. To recreate the typical XACML enforcing architecture, our MSHV Middle-Layer Server (center of Figure 7) acts as the contained solution of policy access, information, decision, and enforcement points. The XACML policies created and stored in the MSHV account of each respective user (acting as the policy retrieval point) limits access to MSHV to through the MSHV Middle-Layer Server, which handles the requests (where data is sent as JSON) of PHA for both the patient and provider versions (middle and bottom of Figure 7). To store the relations (mappings) between the authorized list of providers and their respective patients (used in both versions of the application), our Middle-Layer Server uses MySQL\(^{21}\) with a RESTful API.

---

\(^{19}\) Android, http://www.android.com/
\(^{21}\) MySQL, http://www.mysql.com/
done in PHP\textsuperscript{22} with the Slim Framework\textsuperscript{23}. With this implementation, the server acts as a generic, common point of access for different applications by utilizing web services mapped to MSHV’s API.

Figure 7: Microsoft HealthVault – Middle-Layer – PHA General Architecture.

JSON is utilized for the communication of PHA and the Middle-Layer Server (middle and bottom of Figure 7), allowing us to insure a uniform communication with any application (not only PHA) that can be created for users. The communication between the patient version and the Middle-Layer Server (middle and lower left of Figure 7) is done with unmodified JSON objects, while the communication between the provider version and the Middle-Layer Server (middle and lower right of Figure 7) is combination of unmodified (for the initial request of patients) and filtered (for the resulting data allowed by the policies enforced) JSON. Requests done by the patient PHA are translated to and from MSHV objects (upper left of Figure 7), since the patient is the owner of the data. Requests done by the provider application determine the format of the data to be utilized (upper right of Figure 7). If a provider is

\begin{footnotesize}
\begin{itemize}
\item \textsuperscript{22} PHP, http://php.net/
\item \textsuperscript{23} Slim Framework, http://www.slimframework.com/
\end{itemize}
\end{footnotesize}
requesting information in the patient’s CCR document, then data from MSHV is exported as a CCR schema compliant XML document with policy enforcement performed, whereas any input from the provider to MSHV is first received as a JSON payload, converted to an XML document based on the CCR schema, enforced with policies, and once authorized, translated to MSHV objects for write back.

3. XML Security Framework

UML provides multiple diagrams to visually model applications, but there is a lack of diagrams for the explicit inclusion of security. Our prior work defined new UML security diagrams for supporting RBAC (Pavlich-Mariscal, Demurjian & Michel 2008) via the UML meta-model. Using this as a basis, we have extended this work to define two new UML artifacts (De la Rosa Algarin, A. et al. 2012, De la Rosa Algarin, A. et al. 2013b, De la Rosa Algarin, A. et al. 2013a): the XML Schema Class Diagram (XSCD) that contains architecture, structure characteristics, and constraints of an XML schema; and the XML Role Slice Diagram (XRSD) which has the ability to add permissions to the various elements of the XSCD, i.e. `read/write`, `read/nowrite`, `noread/write`, `noread/nowrite`; both new diagrams are reviewed in Section 3.1. The set of all XML schemas for a given application are converted into a corresponding set of XSCDs. As a result, we provide secure software engineering to the XML design process where the creation of an XML schema is placed into the UML context alongside other diagrams (class, use case, sequence, activity, etc.) that all have the potential to impact the content of an XML diagram for an application. In the literature, approaches to translate a defined XML schema into a UML diagram exist (Bernauer, Kappel & Kramler 2003, Bernauer, Kappel & Kramler 2004, Carlson 2008, Combi, Oliboni 2006, Conrad, Scheffner & Christoph Freytag 2000, Routledge, Bird & Goodchild 2002), each providing varying levels of support for model groups, elements, attributes, and identity constraints (Bernauer, Kappel & Kramler 2004), depending on the approach utilized (e.g., the UML meta-model, a UML profile, a combination of the two, etc.).

In our approach, the new UML XML Schema Class Diagram (XSCD), as shown with a CCR schema’s segment in Figure 8, is an artifact that holds all of the characteristics of the XML schema, including structure, and data type and value constraints. To handle the hierarchical nature of XML schemas in the modeling with the XSCD, we represent each `xs:complexType` in the schema as a UML class with their respective UML stereotype. If an `xs:element` is a descendant of another schema concept, then this relation is represented as an equivalent class – subclass relation in the class diagram. This holds true for `xs:sequence`, `xs:simpleType`, etc. XML schema extensions (`xs:extension`) are represented as associations between classes. We represent data-type cardinality requirements (`minOccurs`, `maxOccurs`) and other XML constraints with a generic «constraint» stereotype assigned to the attribute. The `xs:element` type is respectively represented with a «type» stereotype. Figure 8 presents the way that the
XSCD for the CCR’s schema `xs:complexType ‘StructuredProductType’` (see Figure 8) would look after the transformation process (note that the figure does not include all of the children nodes from the CCR due to space limitations). This XSCD implementation allows for customized access control policies to be generated for the respective concepts of the XML schema.

Next, we present a new UML XML Role Slice Diagram (or XRSD, Figure 9) that is capable of applying access control policies or permissions on the attributes of the XSCD based on role, thereby achieving fine-grained control through the application of security policies and definitions to the XSCD. Permissions on XML documents are read, no read, write, and no write, represented in the XRSD as the respective stereotypes of all the possible combinations, «read/write», «read/nowrite», «noread/write», and «noread/nowrite». As an example, Figure 9 defines Physician and Nurse XRSDs with permissions against the XSCD in Figure 8. Note that in Figure 9, the CCR complexType ‘StructuredProductType’ element Product allows a role to have read and write permissions (Physician) or only read permissions (Nurse) depending on their responsibilities. A Physician role can get all of the information regarding a drug and be able to create new instances following the CCR schema, while, a Nurse role may be limited to read the drug details and cannot create new records. In addition to the permission stereotypes, we also augment the XRSD with the «RoleDescription» and «RoleRequirements» stereotypes, which are used to capture in prose the description of each role and its requirements in the application in terms of permissions.

![Figure 8: XSCD of the CCR schema Segment StructuredProductType](image1)

![Figure 9: XRSD of the CCR Product Element in a Sample Health Care Scenario.](image2)
Finally, we introduce the concept of a Master Role Index (MRI), which is an XML schema to track roles that have the potential to be shared. The Master Role Index is akin to the Master Patient Index that is used for HIE, where the index is able to keep track of patients that have different identifiers in different systems, by creating one master identifier. MRI is intended to give each role a unique identifier, a name, a description of the role, and the type of information that the role can read and/or write. At its schema definition, the MRI consists of an `xs:complexType` with a sequence of elements: `RoleID`, `RoleName`, `RoleDescription`, and `RoleRequirements`. These elements serve as the attributes evaluated when name equivalences between roles of heterogeneous policies are performed for integration. The MRI XML schema and an associated instance are illustrated in Figures 10 and 11, respectively.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="MRI" xmlns=""
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="MRI" msdata:IsDataSet="true"
    msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Role">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="RoleID" type="xs:string" minOccurs="0" />
              <xs:element name="RoleName" type="xs:string" minOccurs="0" />
              <xs:element name="RoleDescription" type="xs:string" minOccurs="0" />
              <xs:element name="RoleRequirements" type="xs:string" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 10: Master Role Index XML schema.

3.2. XACML Mapping with XRSD for Policy Generation

To map the XRSD into an XACML policy, we utilize the policies’ language structure and processing model. As discussed in Section 2.5, XACML policies consist of a `PolicySet`, a `Policy`, and a `Rule`. An XACML `PolicySet` is utilized to make the authorization decision via a set of rules in order to allow for access control decisions. A `PolicySet` can contain multiple `Policy` structures, and each `Policy` contains the access control rules. As a result, the `Policy` structure acts as the smallest entity that can be presented to the security system for evaluation. Based on our understanding of XACML and its usage, we are taking an approach that each XRSD must be mapped into a XACML `Policy` structure with its own set of rules that represent the appropriate enforcement for roles against a schema. Note that multiple XACML `Policy` structures may be generated, resulting in a `PolicySet` for a specific set of XML schemas that comprise a given application.

The collection of `Policy` structures, one for each XRSD, is contained in a `PolicySet`, combined via an algorithm specified by the `PolicySet`'s `PolicyCombiningAlgId` attribute that targets the particular XML schema. The XACML specification defines four standard combining algorithms: `Deny-overrides`, `Permit-overrides`, `First-applicable`, and `Only-one-applicable`. For our intent with XML instance security, and the way that we map the XRSD into an XACML Policy, the combining algorithm of choice is Deny-overrides. With this algorithm, if a single Rule or Policy element is evaluated to Deny, the evaluation result of the rest of the Rule elements under the policy is also Deny. While this might be the case when focusing on access control for XML instances in the document-level, as in our approach, other higher-level systems (e.g. software applications that utilize the XML instance, etc.) can very well deploy security policies with different combining algorithms.

```xml
<?xml version='1.0' encoding='ISO-8859-1' ?>
<MRI>
  <Role>
    <RoleID>1234</RoleID>
    <RoleName>Nurse</RoleName>
    <RoleDescription>Direct involvement with patient care on a daily basis.</RoleDescription>
    <RoleRequirements>
      All clinical information for the patients that they are responsible for (referred to subsequently as clinical info.). Can write/modify a portion of clinical info to record the results/patient progress.
    </RoleRequirements>
  </Role>
</MRI>
```

Figure 11: Master Role Index XML Instance.
In Figure 12, we present the entire mapped XACML policy for the Physician XRSD in Figure 9 that is utilizing data as defined in the subset of CCR schema and instance given in the XSCD in Figure 8. To create an XACML Policy structure per each XRSD, we present the mapping equivalences and rules:

**Policy and Rule descriptors and structure** (green highlight in Figure 12):
- Policy’s PolicyId attribute value is the XRSD’s Role value concatenated to AccessControlPolicy (e.g., the Physician role in Figure 12)
- Rule’s RuleId attribute value is the XRSD’s Role value concatenated to the XRSD’s higher order element (e.g. in Figure 9 it would be Product as defined in the XSCD in Figure 8), also concatenated to “ProductRule”.
- Rule’s Description value is the XRSD’s Role concatenated to “Access Control Policy Rule”.
- There are two XACML Rules under a higher level Target element, one for allowed and one for denied permissions.
- XACML Policy and Rules target and match the role (Subject, e.g., Physician in Figure 12 and 9), the schema elements (Resources, e.g., ProductName, BrandName and Strength in Figure 12, 9 and 8, respectively), and the permissions (Actions, e.g., read and write in Figure 12 and 8).

**Rule Target’s Subject** (yellow highlight in Figure 12):
- Only one XACML Subject and SubjectMatch per Rule.
- SubjectMatch’s MatchId uses the function “string-equal” to evaluate the user’s role as modeled in the XRSD.
- AttributeValue of the Subject is a string, and the value is the XRSD’s Role (e.g., Physician in Figure 11 and 9).
- SubjectAttributeDesignator’s AttributeId is the role attribute.
- While more than one Rule per Policy might exist, the Subject is equal in both cases. This means that the role to be considered for policy evaluation is the same for operations that are allowed or denied.

**Rule Target’s Resources** (blue highlight in Figure 12):
- One XACML Resource per permitted XRSD element.
- Each Resource’s ResourceMatch has a MatchId that determines the usage of the function “string-equal”.
- Resource’s AttributeValue’s value is the XRSD’s element names from the XSCD (e.g., ProductName, BrandName and Strength in Figure 12, 9 and 8), referencing the original schema.
- Resource’s ResourceAttributeDesignator is an AttributeId that determines the target-namespace and datatype of the element.

**Rule Target’s Actions** (grey highlight in Figure 12):
- One XACML Action per operation permitted exists (e.g. read and write in Figure 12 and 9).
- ActionMatch’s MatchId uses the function “string-equal”.
- ActionAttributeDesignator’s AttributeId value is action-write or action-read.
- ActionMatch’s Attributevalue is the permission, read or write, depending on the stereotypes of the XRSD (e.g., read and write in Figure 12 and 9).

Collectively, our approach presents three types of mapping: a role mapping that takes a specific role (e.g. Physician) to a Subject; an element mapping that takes an attribute (e.g. ProductName, Brand, Strength) to a Resource; and a permission mapping that establishes permissions for the element (read and/or write) as Actions. These mapping equivalences and rules permit each XACML Policy to capture the information modelled on the XRSD, while simultaneously limiting the amount of policies needed to
only one per role. While each policy will have two high level Target elements, each with its own rules (for those permissions that are allowed, the Effect of the Rule will be Permit, while those that are denied will have an Effect of Deny), a special case is given to those roles where the permissions are all positive (a «read/write» stereotype in the XRSD) or all negative (a «noread/nowrite» stereotype in the XRSD). In these cases, only one higher-level Target element with one Rule is necessary, and the positivity or negativity of the stereotype determines the Effect of the rule (if «read/write», then Permit, else if «noread/nowrite», then Deny).

The process of mapping the XRSD to an XACML Policy can be automated, as shown by Algorithm 1. The XRSD and schema to be secured serve as the parameters, while the XACML schema is utilized as template for the resulting instances. The first step of the algorithm is determining whether or not all of the permission stereotypes in the XRSD are all positive or all negative (either «read/write» or «noread/nowrite», respectively). If they are, then we know that only one Target and Rule is needed to completely generate an equivalent Policy, and the algorithm proceeds down the right side branch. In this case, the algorithm proceeds through a series of steps. First, the template of the XACML Policy is created (based on the XACML schema) with one high-level target and rule. Depending on the permission stereotypes from the XRSD, the Policy Rule is set with an effect of Permit («read/write») or Deny («noread/nowrite»). Then a three-fold mapping is performed between: the XRSD role and Rule’s Subject (yellow highlight); the XRSD elements and the Rule’s Resources (blue highlight); and, the XRSD permission stereotypes and Rule’s Actions (grey highlight); this finalizes the XACML Policy (Figure 12). Alternatively, if not all of the permission stereotypes in the XRSD are all positive or all negative, then the XACML Policy will require two high-level targets and rules, and the algorithm would proceed down the left side branch. In this case, the algorithm proceeds in a series of alternative steps. The first step is also creating the template XACML Policy, but with two high level Targets and two Rules (one with the Effect of Permit, the other with the Effect of Deny). The fulfilment of these two rules then depends on the permission stereotypes on each element. For those who have a positive permission (either read or write), the elements are mapped as resources of the respective rule; and the permissions are mapped as actions. Note that while two rules exist in this case, the subject will be the same on both (the XRSD role Physician). After this mapping process is complete for each rule, the XACML Policy is finalized.

From an enforcement perspective, in support of either mapping, the process is relatively straightforward. If a user has a role that has a no read permission (like the Nurse role in Figure 8), the policy enforcement point (or equivalent structure in the non-normative XACML architecture) filters the secured XML schema and the instance requested based on the permitted and allowed elements. The generality of these policies could be applied to XSLT (Clark 1999) or other query tools (e.g., XPath and XQuery), provided they have equivalent enforcement points as described above, in order to provide filtered results depending on the role queries; an alternative to the more traditional XML security approach of query rewrites. Our XRSD generated policies must be able to target the software methods at the system’s level; we need to intervene in the case of write actions. In these cases, validation of instance consistency to the original XML schema is left as a task of the system and the application programmer. In support of this, we are exploring the design of our own software to intercept and deal with insertions and updates as part of our overall XML security framework.
Figure 12: Mapped XACML Policy from XRSD.
3.3. Example: Enforcing XACML Security Policies on CCR Instances

In this section, we present the enforcing of the generated XACML policies on XML instances based on a subset of the CCR schema. To demonstrate this, we utilize our in-house developed Personal Health Assistant (PHA) mobile application for health information management from Section 2.6 with its MSVH backend. Recall there are two versions: one for patients to manage their medications stored in MSHV and authorize usage to medical providers; and, the other for providers to view and/or update information on their authorized patients. Finally, we discuss the workflow utilized in the middle-layer server to enforce permissions (read and write) on the CCR instances.

In this section, we describe the way that the XACML policy is enforced when handling reading and writing requests on XML instances whose schema has been secured. From Section 2.6, we assume that a patient has used PHA to authorize a provider with view and update capabilities on their data stored in MSHV. This section starts from this point in order to detail the process and steps that are taken when a user of the provider version of the PHA attempts to access data on a patient, providing an explanation of the way that the CCR XML is securely controlled in its access by a medical provider. The enforcement of security in reading and writing requests is handled by the Middle-Layer Server (center portion of Figure 7) of our prototype. We discuss the enforcement by separating the presentation into the process for a read of and a write of the CCR instance.

Algorithm 1: Mapping from XRSD to XACML Policy.

The process of securing the CCR instance for reading is shown as a set of interconnecting steps in the flowchart of Figure 13, and begins with a request from the provider's PHA. When an initial request is
made, the server retrieves the list of patients tied to the provider pertaining information. When a patient is selected, the server retrieves two XML documents: the complete CCR instance and the XACML policy that targets the schema. When these two XML documents are retrieved, the server enforces security on the CCR instance by filtering and removing elements from the instance as directed by the XACML policy generated from the user preferences. In terms of the XACML given in Figure 12, this would correspond to **read**. Since the PHA application handles information in JSON format, the last step of securing the CCR instance is converting the filtered XML into an equivalent JSON object (as shown in the JSON calls in the right side of Figure 7). This equivalent JSON object is then utilized by the PHA to present the patient data to the provider who is able to view and update as necessary. There may also be a data conversion step, as shown in Figure 1, to take the format from the HIT system and convert to XML in order to allow the security policies to be defined and enforced in the same common format.

As an example scenario, consider a user with a role of Nurse requesting information on a patient's personal health record. In this scenario, the permission of read for a Nurse role has been allowed for medications and allergies, but not for procedures. The permission of write has been disallowed for all of the data elements. When a nurse utilizes the provider’s PHA, s/he selects the patient named Jane Doe. The MSHV Middle-Layer Server retrieves the Jane Doe CCR along with the XACML policy, and enforces security by filtering the CCR XML as directed by the XACML policy. The filtered CCR XML is then converted into a JSON object so that the PHA application can present the information. The process of securing the CCR XML for writing is shown as a set of interconnecting steps in the flowchart of Figure 14, and begins with a request from the provider’s PHA. When a provider wants to update a patient’s record (e.g., medication), the request is sent to the Middle-Layer Server tied to the update data as a JSON object (see middle and lower right side of Figure 7) which verifies the target on which the rules of the requester’s XACML Policy act upon, and evaluates the requester’s role against the policy in order to determine if the write is allowed. This was shown in Figure 12 with the highlighted **write**. If the user requesting an update operation has a role with a permission that allows it to occur, the CCR instance is updated with the sent data, and validated with the CCR schema before the write-back to MSHV. If validation against the schema is successful, then the write-back occurs, and the update performed by the provider is saved in the patient’s MSHV record. If the requester has a role that is not allowed to perform writing operations on the desired element, the Middle-Layer Server drops the request.

---

**Figure 13: Enforcing Reading Permissions.**

- **Initial Request: Patient Health Information**
  - Retrieval of XACML policies
  - Does XACML exist?
    - NO
      - Drop Request: Deny access
    - YES
  - XML – Conversion to JSON
  - Respond Request: JSON

- **Policy Enforcement and Instance Filtering**
  - Match User Role with Policy Rules’ Subject Role
  - Verify Actions and Targeted Resources per Rule
  - Filter XML instance with Guidance from Policy
  - Export filtered XML instance

**Figure 14: Enforcing Writing Permissions.**

- **Initial Request: Information Update**
  - JSON Data Payload
  - Evaluation of target and policy writing rules
  - Is role allowed?
    - NO
      - Drop Request: Deny access
    - YES
  - Write-back to XML instance
  - Validation of updated XML with schema
  - Validation passed?
    - YES
      - Commit saved data
    - NO
  - Drop Request: Invalid
  - Respond Request: Success
3.4. Related Work on XML Security

The work of (Damiani et al. 2000) presents an access control system that embeds the definition and enforcement of the security policies in the structure of the XML documents in order to provide customizable security. The security details can be embedded in the XML DTD, providing a level of generalization for documents that share the same DTD. This is similar to our work in that security policies act in both a descriptive level of the XML instances and target the XML instances. However there are two major differences. First, their work targets XML DTD’s, which have been replaced by XML schemas (our approach). Second, their security policies are embedded into both the DTD and the instance (akin to embedding into the CCR schema and its instance). As a result, when policies experience a change, the cost of updating the XML instances is huge; we emphasize a separation of concerns so that changes in security do not impact schemas and instances. Another effort (Damiani et al. 2008) details a model that combines the embedding of policies and rewriting of access queries to provide security to XML datasets. The XML schema is extended with three security attributes: access, condition, and dirty. Any changes done to the security policy must be updated in the XML schema, and therefore on any XML instance constructed from the schema. While this work is similar to our work in that it targets security in XML document instances via XACML policies, it differs by requiring changes to instance when the policy is modified and does not consider XML document writing (XPath’s design only allow it to perform reading queries to the XML instance) as we do.

The XML Security Working Group (SWG) works on three different security aspects: XML signatures, XML encryption, and XML Security Maintenance. Generally, current approaches only allow for signing complete XML resources, but the XML signature specification contains the methods necessary to only sign portions of an XML instance (Ardagna et al. 2007). XML encryption provides the same granularity as the XML signature, allowing for the encryption of portions (down to the element or element content) of the XML instance, which provides a similar level of granularity to our approach. When XML encryption is utilized, the elements secured are replaced with an EncryptedData element with EncryptionMethod, KeyInfo, CipherData and CipherValue children elements.

The encryption of different sections of an XML document with different encryption keys is presented in (Bertino et al. 2002, Bertino, Ferrari 2002). These keys are then distributed to the specific users based on the access control policies in place. Special focus is given to content-based access control, and users are granted or denied access based on their credentials, which contrasts to roles as in our approach. This makes it difficult to handle policies such as role-delegation, separation of due, and other time and value constraints, unless they are handled at the application level. Finally, (Rahaman, Roudier & Schaad 2008) presents a distributed access control model for collaborative environments where XML documents are used. Their proposed framework utilizes a cryptographic methodology, employing a key management scheme to enforce security policies (much different from our secure software engineering approach). The framework also supports delegation of access control decisions via the use of a lazy rekeying protocol. Ultimately, this approach only handles the reading of XML instances, and does not handle the writing permissions.

Enterprise resource planning (ERP) consists of an integrated packaged software that serves as a single solution for database and communication (Chandrakumar, Parthasarathy 2012), and introduces several features and security of ERP Systems by utilizing XML by focusing on the XML Signature specification and its implementation in .NET. This security framework differs from our work with regards to the security approach taken and the possible access control models supported. XML Signature is only done at the document level, and no level of granularity is supported (element or element content level) as with our approach. Secondly, the primary purpose of the framework is to provide a means of detecting whether a change has been made to the XML information since the last signature was committed, and no sort of access control is handled; while this is an important feature, it doesn’t address the need to control content, but focuses on when the content was changed by whom.
The work in (Ammari, Lu 2010) presents an architecture capable of handling the receiving of XML messages from heterogeneous systems. The architecture focuses on the classification of messages as sensitive and non-sensitive (utilized to determine which sections of the message to be encrypted), and the securing of XML messages via the use of encryption methods applied on flagged XML segments (each segment is encrypted differently based on the data sensitivity and importance level previously defined). While this approach provides direct method for enforcing access control on XML messages, it differs from ours in that the encryption is done in the XML instance (as part of the XML Encryption specification), whereas our approach (though it does not utilize encryption) achieves access control while at the same time extracts the policy and rules from the instance itself.

4. Security Policy Model

Information exchange between heterogeneous systems is hindered by numerous obstacles: the variance in data resources used by the systems; the users and roles that utilize the systems; the permissions the users can perform; the workflow dynamics and constraints that act on the information; the ability to delegate authority in emergent situations; etc. To demonstrate, consider Figure 15, with health information exchange (HIE) among two electronic health records (EHRs) that maintain patient data, a personal health record (PHR) for patients to track health data under their control, and, a patient portal (PP) used by a physician/clinic to interact electronically with patients about medication, referral, and appointment requests. In the figure, hospital A (EHR_A) might utilize CDA (created from XML) as its document format for patient care. This hospital also has its own role-based access control (RBAC) policies, in which nurse and physician might share permissions over certain information components, and may have a DAC policy for a role that can be delegated to an “Emergency Access MD” which could be used to delegate privileges to a physician at hospital B who needs to see data stored at hospital A for a patient that just arrived at hospital B. If hospital A were to share patient information with hospital B (EHR_b), there may be differences in: data format (e.g., B uses CCR or another proprietary XML format); permissions of the logically equivalent roles (physician and nurse); and/or the data provided by hospital A may not satisfy what B needs. Even if HIE is supported (see Figure 15), the type of information and permissions may differ for each contributor. Our focus in this section is on the governing policies that dictate which resource can be accessed by whom, when, where and how. Towards this end, we define two new UML diagrams: the Policy Slice Diagram (PSD) in Section 4.1, which models the granular permissions for roles; and, the Security Policy Schema Set (SPSS) in Section 4.2, which models the entire security policy as a set of PSDs (which correspond to XRSDs). We assume each local system (e.g., EHR_A) has its own secure policies, with complementary and conflicting requirements, that can be an obstacle for attaining information exchange, particularly when constructing an application that is interested from accessing from all sources in Figure 15. To complete the discussion, Section 4.3 explores related work in policy and security modeling.

4.1. Policy Slice Diagram (PSD)

Each PSD, as shown in Figure 16a is a UML diagram, that describes a segment of the local security policy of an application (say, EHR_A) from the perspective of security rules (SR) that are triples of the form: SR = <Role (r), Permission (p), Entities (ents)>, where an entity can be an object, an XML attribute, an XML xs:complexType, an XML XQuery, an XML schema, etc.; this is left very general to allow for all of the potential types of access that are provide by the different systems (see Figure 15). Each PSD contains a set of one or more security rules. A sample SR from Figure 16b is <Nurse, «read», ProductName> which indicates that Nurse can read the attribute ProductName. Figure 15b represents a total of 8 SRs: Read and Nowrite in combination with ProductName, Brand, and Strength (6 SRs) and Read and Write with Brand (2 more SRs). In practice, the set of all of Policy Slice Diagrams (PSDs),
define all of the security rules for a local application (say, EHR\textsubscript{A}) that are to be made available (published), which in turn is represented by an SPSS; these of course may be less than what is available in the local system, e.g., patient names and address, de-identified data on patient and diagnoses, aggregate data, etc. Security rules are also defined using an XML schema, and their definition augments and complements XRSD so that when coupled with the PSD is then able to assist in the appropriate generation of a XACML enforcement policy which can then be used to filter XML instances into Secure Enforced (SecE) instances that guarantee that the correct information is precisely delivered with a degree of assurance. The XML schema, as given in Figure 17, and an associated instance for SRs, in Figure 18, continue the ongoing example.

![Policy Integration Model](image)

**Figure 15:** A Health Information Exchange Model.
4.2. Security Policy Schema Set (SPSS)

The union of all PSDs for a source (e.g., EHR_A) defines a local security policy (that is, the entire policy that acts on a system, application, repository, piece of software, etc.), which is a Security Policy Schema Set (SPSS) – what is published. SPSS, as shown in Figure 19, is the local security policy (LSP) of the
application. At the top of Figure 18, UML classes represent: Roles with RoleID, RoleName, RoleDescription and RoleRequirements as string type members (defined in the MRI XML schema definition); Permissions with read and write as Boolean members; and, Entities with two generic object members, one extended with the «atomic» stereotype and the other with a «nonatomic» stereotype. Since entities can exhibit semantic differences (e.g., an element in an XML schema can have multiple node children, or they might be a leaf in the XML schema tree), we categorize by utilizing two stereotypes, «atomic» and «nonatomic». The Permissions class has a dependency relation between the Roles and Entities class to grant a permission and verify the role and the entity the permission targets, while all three classes have an instance level relationship of composition with the parent LSP SPSS, which is shown at the bottom of Figure 19, where SPSS is a union of the various PSDs.

![Diagram of UML SPSS with PSDs Realizing the LSP.](image)

Note that the union of PSDs that yields the LSP security is not performed as a traditional mathematical set union. In order for security policies to maintain a consistent degree of semantics, policy combination algorithms exist to determine the result of a policy evaluation when one or more rules (or policy in the case of policy sets in XACML) are inconsistent. For example, the XACML specification offers combination algorithms (shown in detail in Appendix A) such as: Deny-overrides in which a policy is denied if at least one of the rules is denied; Permit-overrides in which a policy is permitted if at least one of the rules is permitted; First-applicable in which the result of the first rule’s evaluation is treated as the result of all evaluations; and, Only-one-applicable in which the combined result is the corresponding result to the acting rule. To maintain an equivalent level of semantic knowledge in the SPSS, the union of its PSDs must be performed in a similar manner. Towards this end, we present four corresponding policy slice combination algorithm stereotypes («deny-overrides», «permit-overrides», «first-applicable», «only-one-applicable»), assigned to the SPSS, which specifies which combination policy is enforced by LSP. In Figure 19, one of these four stereotypes would take the place of the «PSCombAlg». To place this into an appropriate context, recall Figure 15 and assume that each local application has defined a set of PSDs for its SPSS, where each role is in the MRI with its own XML instance. A Security Rules Schema repository has also been instantiated for each application.
4.3. Related Work in Policy and Security Modeling

For security and policy modeling, one approach based on RBAC and UML is SecureUML (Lodderstedt, Basin & Doser 2002) that combines a graphical notation for RBAC with constraints, and policies can be expressed using RBAC permissions, with complex security requirements done with a combination of authorization constraints. This work is similar to ours in the aspect of a graphical approach to security requirements modeling, but our approach differs in the fact that we model policies for eventual integration and enforcement. A later effort (Mouelhi et al. 2008) presents a model driven security approach for designers to set security requirements along with the system models to automatically generate an access control infrastructure. The approach combines UML with a security modeling language defining a set of modeling transformations; the former produces infrastructures for JavaBeans, and the latter can generate secure infrastructures for web applications. This work is similar to ours with regards of using UML as the modeling language for security, but we generalize away from policy and security instances by focusing on a higher conceptual level of security modeling.

(Basin, Doser & Lodderstedt 2006) utilizes the model driven architecture paradigm to achieve security for e-government scenarios with inter-collaboration/communication. This is achieved by describing security requirements at a high-level (models), with relevant “security artifacts” being automatically generated for target architectures, removing the otherwise present learning curve in specifying security requirements by domain experts with no technical know-how. This is dramatically different from our work, which is focused on information exchange and its security assurance. Similar to our presented work, (Breu et al. 2005) uses aspect oriented programming and a generic security meta-model for the automatic generation of platform specific XACML. In contrast our approach’s objective is to model security policies for the eventual integration for a distributed scope.

An earlier work (Alam, Breu & Breu 2004) shows the way that a system designer can build a web-services interface model tied to security requirements using object constraint language and RBAC, which then automatically generate XACML policies. This approach is similar to our work with regards to modeling and automatically generating XACML policies, but differs in the language utilized to model the security requirements. Another difference is that we aim to model security requirements and existing security policies alike, with the eventual purpose of integration. Lastly, the work of (Nakamura et al. 2005) presents a tooling framework used to generate security configurations for web services. Their approach utilizes a model driven architecture, which yields automatically generated security configurations from the security intentions added to the model. These security configurations transform UML constructs and create a security environment model in the form of configuration files.

5. Policy Integration Model and Process

Integration of security policies has been typically done by similarity finding (Mazzoleni et al. 2008, Mazzoleni et al. 2006) or specialized algebra (Rao et al. 2009) that utilizes two instance policies, usually validated against the XACML schema, to produce an integrated result. Using our introduction of the Policy Slice Diagram (PSD) in Section 4.1 and the Security Policy Schema Set (SPSS) in Section 4.2, in this section, we are able to present an approach towards heterogeneous policy integration at both the model and process levels. This is intended to demonstrate the way multiple HIT system (each with their own XML schema) can be brought together using the new UML XML diagrams (XCSD and XRSD), the Master Role Index, and the PSD and SPSS, in order to define a global unifying security solution. Towards this purpose, in Section 5.1, we detail the general steps of the policy integration process by describing the way to integrate local SPSSs into a Global SPSS, providing the linkage of the UML XML extensions in Section 3 with thee PSD and SPSS in Section 4. Next, in Section 5.2, we discuss the process that is undertaken to ensure that concepts that cut across multiple local SPSSs are addressed through a process of name equivalence, structural equivalence, and conflict resolution. Then, in Section 5.3, we
examine the process to generate an enforcement meta-policy in XACML from the integrated Global-SPSS. We finish this section with related work in policy integration in Section 5.4.

5.1. Integrating Local SPSS into a Global SPSS

In this section, we present a process and the steps that are required to characterize our XML security framework from the earliest stages (identification of HIT systems) through the usage of security enforced (SecE) XML instances provided to users, which involves the integration of Local SPSSs into a Global SPSS for a new application, by using the MRI and Security Rules (SRs) for each Local SPSS, as given in Figure 20. Recall that we are proposing a publisher/subscriber model; each local system (e.g., EHR\textsubscript{A}) is free to propose multiple SPSSs that contain different PSDs in order to tailor the information they want to publish for different stakeholder (e.g., patient data for a clinician at another facility in an emergent situation, de-identified data for a clinical researcher, symptom results for a population research seeing evidence of epidemics, etc.). As a development process, we propose three steps:

1. Local Systems have subsets of their SPSS (or some specialized SPSS) for information usage, sharing, and exchange published and available for designers/developers of new applications who wish to subscribe to information, functionality or other entities governed by local SPSSs.
2. A new application is developed that will utilize resources from these local systems, but must first check the Local SPSSs for exposed (published) PSDs. If there are exposed SPSSs for public usage, the new application can subscribe to those published SPSSs that are relevant.
3. The new Global SPSS for the new application will result from the integration of the exposed (published) local SPSS that have been subscribed to, as well as the new security requirements of the new application.

In Figure 20, the entire process at an instance level is represented; in effect, we are expanding the Policy Integration Model box from Figure 15 to show the complete picture starting from the data sources and ending up with a GPSS that is enforced to yield XML instances that are SecE. From the perspective of a meta-system currently on development, existing data sources (top-right of Figure 20) can utilize any XML schema (as in EHR\textsubscript{A}, EHR\textsubscript{B}, PHR\textsubscript{A} and PP) or another document structure standard that can be readily converted to an XML equivalent (as in PHR\textsubscript{B} with a JSON structure). By representing these schemas as UML diagrams with the XML Schema Class Diagram (see Section 3.1 again), they can be augmented with any access control model requirements (top-left of Figure 20), yielding security-related UML constructs (such as the XML Role-Slice Diagram for RBAC security requirements). These XRSDs, grouped by the schema of each respective data source, can be represented as Policy Slice Diagrams (PSDs), which act as a generalization of security rules without regards of access control model. In turn, the set of PSDs for a respective schema (data source) make up the original security policy, which we represent with the Security Policy Schema Set (SPSS, left of Figure 20). With these PSDs, we employ our integration model, utilizing the Master Role Index (MRI), name equivalence, structural equivalence and conflict resolution to produce a conflict-free global security policy schema set (bottom of the policy integration box of Figure 20). Utilizing a similar methodology of XACML policy generation as the one shown in Section 3.2, we can produce an enforcement meta-policy that can effectively targets all the validated instances of the original schemas from the data sources. This enforcement produces security-enforced (SecE) instances (bottom-left of Figure 20) at the software application level.

5.2. Equivalence and Conflict Resolution

In order to reconcile naming conventions of roles and their permissions, and clearly understand the way that information can be shared and the scope of roles and their permissions against that information (entities), the publisher/subscriber paradigm provides an ideal way for the data source owner to decide what to present to whom. We assume that each Local SPSS has its own set of SRs captured as XML
instances (\(<\text{role, permission, entity}>\) triples), and that the local application may have multiple SPSSs. We further assume that when given a set of Local SPSSs, is it possible in an automated fashion to attempt to match across all of the involved systems to determine which PSDs (or portions of PSDs) have some level of equivalence.

In \textit{name equivalence} the intent to be able to take two different PSDs and see if they are equivalent. Recall that the MRI schema is a shared resource that is instantiated with an initial set of roles (in our case, for a health care domain, perhaps the roles Nurse, Physician, and Office Admin). As each system (see Figure 15) establishes its own local SPSS (publishes a PSD or portion thereof), it is free to use existing instances (roles) in MRI, or define new roles, thereby expanding this shared role repository. Remember, \textit{EHR}_A or \textit{EHR}_B may only wish to expose some of its PSDs, and for each PSD, may want to limit even further. For name equivalence, from Figure 15, if \textit{EHR}_A has a PSD with RoleID “1234” (Nurse) with RoleDescription and RoleRequirements, and \textit{EHR}_A has a PSD with Role “1234” (Nurse) with the same
RoleDescription and RoleRequirements values, then these are clearly equivalent by name. But this only means that they have the potential to also be structural equivalent to some degree. The reason is that the security rules are unique for each system; EHR\textsubscript{A} has its own set of SRs, and the ones for the Nurse role might be different from the SRs for the Nurse role in EHR\textsubscript{B}.

Name equivalence verification between roles of heterogeneous local systems can be achieved by utilizing the MRI, where two roles are name equivalent iff their respective RoleName, Role Description, and RoleRequirements, are the same. Formally, we define:

**Definition 1:** Local System \textsubscript{A} and Local System \textsubscript{B} with L-SPSS\textsubscript{A} and L-SPSS\textsubscript{B}, PSD\textsubscript{A}Role and PSD\textsubscript{B}Role, with PSD\textsubscript{A}Role.RoleName = (PSD\textsubscript{B}Role.RoleName, are name equivalent iff PSD\textsubscript{A}.Role.RoleDescription = PSD\textsubscript{B}.Role.RoleDescription) and PSD\textsubscript{A}.Role.RoleRequirements = PSD\textsubscript{B}.Role.RoleRequirements.

Unlike name equivalence, structural equivalence seeks to examine the conditions under which two policy slices are the same by looking at the way they are constructed piece by piece. In order to see if they policy slices are the same; an algorithm for structural equivalence needs to be executed. We ask ourselves, does Nurse, as given in Figure 16b PSD for EHR\textsubscript{A}, have the exact same list of 8 SRs for the Nurse PSD for EHR\textsubscript{B}? The hierarchies do not have to be identical, but the security rules that are constructed from each PSD must be equivalent, namely: EHR\textsubscript{A}.PSD\textsubscript{NURSE}.SRSet ≡ EHR\textsubscript{B}.PSD\textsubscript{NURSE}.SRSet. The level of structural equivalence can vary, if for instance EHR\textsubscript{B} that are not present in EHR\textsubscript{A}.

These equivalences (and containment) at the structural level are crucial to allow sharing among systems in a consistent manner. Towards an information exchange process, a Global SPSS can be constructed for a new application by subscription to the available Local SPSSs. Unlike integrating homogeneous policies (policies utilized on the same application, with roles, permissions, rules, etc., that target a limited, unchanged set of resources), our process aims to integrate heterogeneous policies (which contain equivalent and nonequivalent role and permissions) that act on different resources utilized and provided by different local systems. To successfully create the Global SPSS for a new system, the Local SPSSs need to be integrated via name and structural equivalence and other techniques that are able to analyze PSDs and their SRs across a set of published Local-SPSSs requested by the Global SPSS.

Finally, conflict resolution involves the process of examining the equivalences and understanding the way that they relate across multiple local SPSSs as attained within the Global SPSS. Finding name equivalences results in two sets of roles. An equivalent set (EqualRoles), which satisfy the previous definition; and an unmatched set of roles (DistinctRoles), which do not match with any other roles. The PSDs that correspond to the roles in the set DistinctRoles can automatically become part of the new Global SPSS (if the Local SPSS is subscribed to), as there is no risk of conflict between other roles and their respective security rules. Note that two roles that are name equivalent may still have differing sets of SRs in their respective PSDs. As a result, structural equivalence must also be quantified.

Structural equivalence is a process that assumes name equivalence is met, and then proceeds to examine the actual SRs for each of the involved PSDs of Local SPSSs. To verify structural equivalence it is necessary to examine the security rules set (SRSet) of each involved PSD. To do so, we note that there are four combinations that can exist as a result of the comparison between two SRSets, as given in Figure 21: equality, where the SRSet of two systems are contained within one another; containment, where the SRSet of one PSD fully contains the SRSet of another PSD (proper containment, hence not equal); intersection, where the SRSet of two PSDs have at least on SR rule in common (intersection is no empty); and disjoint, where the SRSet of two systems with equivalent roles have no security rules in common.
(intersection is the empty set). These relations can be the result of either security rules overlapping with one another, or different entities being available as resources on one system, but not the rest.

<table>
<thead>
<tr>
<th>Equality</th>
<th>Intersection</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SRSet_A = SRSet_B )</td>
<td>( SRSet_A \cap SRSet_B )</td>
</tr>
<tr>
<td>Containment</td>
<td>Disjoint</td>
</tr>
<tr>
<td>( SRSet_A \subset SRSet_B )</td>
<td>( SRSet_A \cap SRSet_B = \emptyset )</td>
</tr>
</tbody>
</table>

Figure 21: Possible Relations of SR Sets of Local SPSSs.

Successfully integrating these SRSets of PSDs while maintaining a proper level of security that complies with the local security requirements of constituent systems depends on the resulting relation between SRSets. Equality between SRSets is trivial; since no conflict can arise, both security rule sets can be consolidated and made part of the Global SPSS. Handling SRSets that have a containment relation (proper subset) results in a subset of security rules being added to the Global SPSS (all of the security rules of the contained smaller set), leaving the complement of the contained set in a state of conflict. Similarly, the intersection between two SRSets, has two cases: when null, the SRSets are disjoint, and both SRSets are in a state of conflict and there are two options, either include them both or omit them both from the Global SPSS; when non-null, only the intersection is included, resulting in the SRs that not in the intersection to be excluded.

Conflict resolution can be achieved by either utilizing a fully automated approach, which makes hard assumptions in order to provide the highest level of access control, or a human-guided approach, where human intervention is necessary in order to select which security rules will become part of the Global SPSS due to the intersection and disjoint cases. For an automated approach, we note that there exist different options capable of maintaining a proper sense of security:

1. **Safe and lazy approach**: The Global SPSS will utilize the most restrictive security rules, with respect of the entity involved, ensuring that no destructive operations (write) or a breach in access (reading) can be performed; SRs are excluded if not in common.
2. **Hierarchical approach**: A hierarchical order is given to the constituent local systems (and their SPPSs) in order to decide which SR(s) takes precedence over other SR(s). This order of priority results in SRs for a local system that can span a spectrum in terms of the information to be published, i.e., the granularity of data would range from a detailed patient record (lower level of a hierarchy) to an aggregation of patient records for a population study (higher in the hierarchy).
3. **Data Owner over Requester**: The Global SPSS utilizes the SRs specified by the data owner (e.g., a repository, and only if available), disregarding other conflicting SRs.

While these automated approaches maintain a robust level of security, we note that they also present limitations. In the case of the safe and lazy approach, there could be automatic denials of access to SRSets that could have otherwise been authorized in the scope of the local system via human intervention. The data owner approach presents the possibility of granting access to an otherwise secured request (for example, an EHR can allow read permissions to nurses, but the new developed system might want to limit the amount of reads to a subset of the information). The hierarchical approach offers the best alternative to conflict resolution, but the implementation depends highly on the order that the
information published by Local SPSSs would need to be ranked by some process (or human). While there are natural hierarchies of information and its granularity in health care, this may not be generalized to all domains.

Clearly, the safest approach to conflict resolution is one where human intervention is employed to create an appropriate Global SPSS for the new application. This is not an unreasonable requirement, given our secure information engineering approach approach to policy integration, since the human designer of the new application will be in the loop to examine the published PSDs for each Local SPSS in order to decide which of them are relevant for the new information exchange/sharing application. This would require a final decision be made on all of the conflicting security rules. Potentially, if there is no name and structural equivalence between SRSets, the designer will have to decide for each of the security rules that will make part of the system.

5.3. Generating an Enforcement Meta-Policy from the Global SPSS

The software designer that is creating the new application will use the capabilities described in Sections 5.1 and 5.2 and the potential integration of Local SPPSSs in order to arrive at a Global SPSS that contains all the relevant SRs for the new application, with respect to the roles, packaged into a respective Global PSDs. Using the approach, we decompose the Local Security Policies by modeling them as SPSS (higher level), PSD (intermediate level), and SRSets (lower level); we build the Global SPSS following a bottom-up approach (from SRSets to PSDs to Global SPSS). Since we assume that all of the SRS conflicts have been resolved, whether by the use of a human guided approach or an automated approach that satisfies the new system’s security requirements, the process of building the Global SPSS involves three steps. In the first step, Global Policy Slice Diagrams (G-PSDs) are built utilizing the SRSets with respect to roles. This is done with the aid of the MRI, following Definition 1, ensuring that name equivalence is correct and that we are packaging the properly equivalent SRSets. In the second step, a translation between the security rules represented in the PSDs (recall the triple SR = <Role (r), Permission (p), Entities (ents)>) and UML class members, is performed. To achieve this, we group the SRs by Role and Entities, and propose the following equivalence:

- The combination of possible permissions (read, write) and Entities from the SR will result in a class method with a stereotype that determines whether the method is permitted («allow») or denied («deny») for execution. For example, <Nurse, Read, Medications> and <Nurse, NoWrite, Medications> will result in two class methods, «allow»readMedications() and «deny»writeMedications().

Once this step is completed, the third and final step is structuring the UML artifact with respect to roles. This involves several rules and steps:

- Roles that will be part of the Global SPSS are represented as UML subclasses of the Global SPSS class. The role classes are named after their respective roles.
- Each role class will have descriptive string members, RoleDescription and RoleRequirements, per the MRI.
- Each role class will have the methods, as given by the second step of the integration. This means that two methods per targeted entity of each role will be given, as dictated by the SRs.

The result of this process can be seen in Figure 22. Note that the Nurse and Physician, roles are represented as subclasses of the parent Global SPSS class. The Nurse subclass has two string members, which are instantiated to the MRI values of the RoleDescription and RoleRequirements. This subclass also has 8 methods (one for each SR), which are divided into 4 entities and their permissions are given as attached UML stereotypes. The Physician role is structured similarly. We note that the structure of the Global SPSS is not similar to the SPSS structure of Figure 19, used to represent Local SPSSs. The justification for this lies in the end use of each structure. In the case of the Local SPSS (Figure 19), it
tries to model an already existing security policy in order to obtain the policy slices (and in turn, security rules) without the loss of structure or information. The Global SPSS, on the other hand, is the result of integration that will be utilized as a blueprint for the generation of an actual enforcement policy. Due to this, we chose to structure the Global SPSS in a role centric way, granting us the benefit of avoiding any complexity in the mapping process for the enforcement policy generation.

Generating an enforcement XACML Meta-Policy from the resulting Global SPSS follows a similar process as the one utilized in our previous research (De la Rosa Algarín, A. et al. 2013b), where XACML policies are generated from modeled XML Role-Slice Diagrams (XRSDs). The benefit of utilizing the Global SPSS as the blueprint for an instance XACML policy generation is that we can avoid performing any equivalence testing (since it was done indirectly as part of the name and structural equivalence), thus providing us with an already organized diagram that only requires direct translation to code. The rules to create an instance of the enforcement XACML meta-policy are as:

- The Global SPSS class is an XACML PolicySet structure that acts as the container for the respective policies.
- The subclasses of the Global SPSS class (Role classes) are individual XACML Policy structures with the policy combination algorithm specified by the system’s security requirement (deny-overrides, permit-overrides, etc.).
- The sub-subclasses of the Global SPSS class (the Global PSDs) are XACML Rule structures, with the effect of Permit, part of their parent policy.
- Each XACML Rule is tied to a Global PSD, which contains a Subject, Resource and Action attributes. Subject is mapped to the PSDs role (R), with the RoleDescription and RoleRequirements as sub-elements. Resources and Actions are extracted of the methods of each role class of the Global SPSS.
- In the case of methods having different permission stereotypes (the Nurse Global PSD in Figure 16), two rules will be created for the policy, one with the allowed permissions and entities and the other with the denied permissions and entities (analogous to our approach in (De la Rosa Algarín, A. et al. 2013b)).

![Global SPSS of Health Care Example.](image)

**Global SPSS**
- Roles : {RoleName, RoleDescription, RoleRequirements}
- Permissions : {read, write}
- Entities : Object Set

«GlobalPSDs» Nurse

- RoleDescription : String = “Direct involvement with patient care on a daily basis.”
- RoleRequirements : String = “All clinical information for the patients that they are responsible for (referred to subsequently as clinical info.). Can write/modify a portion of clinical info to record the results/patient progress.”

- «allow» +readStrength()
- «allow» +readProductName()
- «allow» +readBrandName()
- «allow» +readPersonal()
- «deny» +writeStrength()
- «deny» +writeProductName()
- «deny» +writeBrandName()
- «allow» +writePersonal()

«GlobalPSDs» Physician

- RoleDescription : String = “Handle the medical needs (diagnosis, treatment, etc.) for patients.”
- RoleRequirements : String = “All clinical info. on patients and who they can contact (nurses, technicians, therapists, etc.) regarding their patients. Also, insurance related data and other office-based data to maintain their private practice. Overlap of write privileges with Staff_RN, but can also modify portions of clinical info. that issue orders.”

- «allow» +readStrength()
- «allow» +readProductName()
- «allow» +readBrandName()
- «allow» +readPersonal()
- «allow» +writeStrength()
- «allow» +writeProductName()
- «allow» +writeBrandName()
- «allow» +writePersonal()
5.4. Related Work in Policy Integration

The work of (Mazzoleni et al. 2008) presents a policy integration methodology to find similarity of policies on distinct levels (rule effects, targets, roles), and integrating over a set of defined rules, also considering policy decision conflicts. This is similar to our work, but differs in that the integration is done at the instance level (XACML) and is performed over homogeneous policies. In a similar way, (Lin et al. 2007), has an approach that also focuses on the instance level, creating a dependency on the OASIS XACML specification and policy structure to achieve proper results.

Another approach (Rao et al. 2009) proposes an algebraic solution to the problem to integrate complex security policies at a granular detail with an algebra that consists of five operations (three binary and two unary). In turn, this algebra is utilized as part of a framework to achieve the generation of an instance policy automatically. This work is similar to ours in the sense that it seeks to integrate policies at their security definition layer, but it differs in other areas. For example, we utilize a model-driven approach to integration instead of an algebra-based method. Another difference is that the algebra is aimed to instance XACML policies, while we seek to abstract away to the conceptual model (effectively removing the need of an existing instance policy).

The work of (Hale et al. 1999) recognizes the challenge of security policy coordination between heterogeneous and autonomous systems, and present an architecture that utilizes mediators and a ticket-based authorization model to handle policy discrepancies across constituent systems. Their method is similar to ours in the aspect of heterogeneous policy management of constituent systems in an information exchange process, but it differs in the fact that we seek to integrate policies for use in a newly developed meta-system. Instead of focusing on the distributed environment (where each constituent system is treated as autonomous and a primary actor), we focus on the case where a new system is being developed that utilizes resources from each of the existing systems.

In (Shafiq et al. 2005) an approach based on integer programming is used for RBAC defined policy integration. In this work, policies from multiple domains can be integrated into a global access control policy. Due to the nature of multiple domains and integration, conflicts are resolved by utilizing an integer programming approach. This work is similar to ours with regards to heterogeneous policy integration, though we do not focus on the integration of cross-domain policies. A notable difference is that, whereas their work focuses on RBAC policies, we seek to integrate RBAC, MAC and DAC policies into a global security policy.

Lastly, (Yau, Chen 2008) presents an approach to policy integration that includes a similarity-based policy finding algorithm with the special distinction of handling collaborative workflows. This work is perhaps the most similar to ours since they seek to create a complete solution to integration and conflict resolution. The work differs in the method of integration and conflict resolution, which we opt to provide a software engineering based approach that depends on human intervention (since its part of the software development process), whereas this work tries to provide an approach that can be automated with user-defined control variables.

6. Remaining Research and Completion Time

The presented work on this proposal leverages UML and extensions done to it to achieve an XML security framework with the purpose of automatically generating enforcement policies and integrating heterogeneous policies for the generation of an enforcement meta-policy. The work presented herein focused on the attainment of RBAC defined security policies, and in turn our focus on remaining work focuses on these major areas:
• **MAC and DAC Support for UML-XML Extensions**: In this continuing work, we will define UML-XML security diagrams that support the MAC and DAC models. The result of this will allow our document-level security framework for XML schemas and instances to properly secure XML structured data across systems and applications that utilize a model other than RBAC to define security. What about a User Authorization that maps Users to Role? Won’t we need that to generated the filtering for each user playing one of their authorized roles.

• **Integration Process of RBAC, MAC and DAC Defined Security Policies**: Our existing work tackles the integration of RBAC defined security policies with the use of the PSD, SPSS and the MRI concepts. In this continuing work, we seek to support the integration of security policies defined by the different access control models (RBAC, MAC, DAC), effectively providing a global approach to policy integration regardless of the security definitions utilized.

The timeline for the remaining and future work is as follows (Figure 23). The UML extensions for MAC and DAC support will be the main focus of the next 4 months. Shortly afterwards, the following 4 months will be dedicated for the formal definition of the integration process between RBAC, MAC and DAC policies. As soon as this aspect of our work is complete, we will formalize the secure information engineering process model for secure meta-system creation. We expect this component of our research to take around 4 months as well. The final step would be to complete the prototyping efforts, which we aim to complete in 5 months, and will serve as a proof of the research done. Dissertation writing will be done in parallel to the remaining work, giving a suggested timeline of 16 months to completion starting today.

<table>
<thead>
<tr>
<th>Task / Month</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML-XML for MAC and DAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RBAC, MAC and DAC Policy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure Information</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Engineering Process Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efforts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure Information Engineering Process Model</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping Efforts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dissertation Writing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 23: Remaining Research and Tentative Completion Time.**

7. **Conclusion and Expected Contributions**

The work presented has proposed an XML security framework that has addressed the issue of providing security in meta-systems that utilize security policies defined after the different access control models (RBAC, MAC and DAC). In these cases, it is not enough to utilize the security requirements of the newly developed system; special attention must be given to the security definitions and requirements of the constituent systems. Towards this objective, we focus on XML, due to its widespread adoption and its status as the de facto standard of information exchange, to be the data format in need to be secured. In Section 3 we detailed our security framework for XML schemas and instances focusing on the attainment of RBAC, which utilizes extensions to UML (the XRSD and XSCD) to provide a secure information engineering approach towards information security. The XRSD in turn is utilized for the generation of XML schema targeting security policies, written in XACML. This approach we have chosen benefits systems developers and software engineers by separating security concerns from the data, an otherwise
unachievable goal with traditional methods of XML security such as embedded security. Continuing our efforts towards secure information exchange in these meta-systems, Section 4 and Section 5 presented our model and process for heterogeneous security policy integration from the perspective of the development of a new meta-system. By extending UML once again, Section 4 presented the new policy oriented diagrams (the PSD and the SPSS), which enable us to abstract away from the need of an instance security policy that other approaches depend on (such as XACML). In turn, our integration process, as presented in Section 5, yields a Global-SPSS that can be utilized as the blueprint to generate an enforcement instance policy written in XACML.

In summary, the work presented in this proposal contributes to the following research areas and disciplines:

- **UML Security Extensions for UML (completed for RBAC, in progress for MAC and DAC):** Focusing on the NIST RBAC model first, we provided two extensions to UML in the form of diagrams, the XSCD and the XRSD, which permits developers to design the security aspect of information from a software engineering perspective.

- **XML targeting XACML Policy Generation (completed for RBAC, in progress for MAC and DAC):** By utilizing the XRSD as a blueprint, developers can engage in an automatic process that results in the generation of enforcement XACML policies, which in turn targets the XML schema the XSCD and XRSD were modeled, and its instances.

- **Security Policy Integration (completed for RBAC, in progress for MAC and DAC):** Extending UML with two policy oriented diagrams, the PSD and the SPSS, developers can integrate policies from heterogeneous systems in order to obtain a conflict free policy that can be utilized for a developed meta-system.

- **Secure Information Engineering (in progress):** Utilizing our XML and policy extensions to UML, we can develop a well-defined software development process for integrated meta-systems and applications that contain functionality dependent on information exchange processes. With this new secure information engineering paradigm, developers of meta-systems can follow a development cycle that provides security and information assurance in terms of compliance with constituent systems and their security requirements.

The result of attaining this security framework will provide a means to integrate security policies, and their associated access control models, within a global context. In turn, this will allow access by applications built as meta-systems to be more effectively controlled. This will greatly benefit information usage and sharing in an information exchange process.
References


Appendix A – XACML Rule and Policy Combination Algorithms

Permit- Overrides:

Decision permitOverridesCombiningAlgorithm(Decision[] decisions) {
    Boolean atLeastOneErrorD = false;
    Boolean atLeastOneErrorP = false;
    Boolean atLeastOneErrorDP = false;
    Boolean atLeastOneDeny = false;
    for( i=0 ; i < lengthOf(decisions) ; i++ )
    {
        Decision decision = decisions[i];
        if (decision == Deny)
        {
            atLeastOneDeny = true;
            continue;
        }
        if (decision == Permit)
        {
            return Permit;
        }
        if (decision == NotApplicable)
        {
            continue;
        }
        if (decision == Indeterminate{D})
        {
            atLeastOneErrorD = true;
            continue;
        }
        if (decision == Indeterminate{P})
        {
            atLeastOneErrorP = true;
            continue;
        }
        if (decision == Indeterminate{DP})
        {
            atLeastOneErrorDP = true;
            continue;
        }
        if (atLeastOneErrorDP)
        {
            return Indeterminate{DP};
        }
        if (atLeastOneErrorP && (atLeastOneErrorD || atLeastOneDeny))
        {
            return Indeterminate{DP};
        }
        if (atLeastOneErrorP)
        {
            return Indeterminate{P};
        }
        if (atLeastOneDeny)
        {
            return Deny;
        }
        if (atLeastOneErrorD)
        {
            return Indeterminate{D};
        }
    }
    return NotApplicable;
}

Deny- Overrides:

Decision denyOverridesCombiningAlgorithm(Decision[] decisions) {

Boolean atLeastOneErrorD = false;
Boolean atLeastOneErrorP = false;
Boolean atLeastOneErrorDP = false;
Boolean atLeastOnePermit = false;

for( i=0 ; i < lengthOf(decisions) ; i++ )
{
    Decision decision = decisions[i];
    if (decision == Deny)
    {
        return Deny;
    }
    if (decision == Permit)
    {
        atLeastOnePermit = true;
        continue;
    }
    if (decision == NotApplicable)
    {
        continue;
    }
    if (decision == Indeterminate{D})
    {
        atLeastOneErrorD = true;
        continue;
    }
    if (decision == Indeterminate{P})
    {
        atLeastOneErrorP = true;
        continue;
    }
    if (decision == Indeterminate{DP})
    {
        atLeastOneErrorDP = true;
        continue;
    }
    if (atLeastOneErrorDP)
    {
        return Indeterminate{DP};
    }
    if (atLeastOneErrorD && (atLeastOneErrorP || atLeastOnePermit))
    {
        return Indeterminate{DP};
    }
    if (atLeastOneErrorD)
    {
        return Indeterminate{D};
    }
    if (atLeastOnePermit)
    {
        return Permit;
    }
    if (atLeastOneErrorP)
    {
        return Indeterminate{P};
    }
    return NotApplicable;
}

First-applicable:

Decision firstApplicableEffectCombiningAlgorithm(RuleOrPolicies[] rulesOrPolicies)
{
    for( i = 0 ; i < lengthOf(rulesOrPolicies) ; i++ )
    {
        Decision decision = evaluate(rulesOrPolicies[i]);
        if (decision == Deny)
        {
            return Deny;
        }
if (decision == Permit) {
    return Permit;
}
if (decision == NotApplicable) {
    continue;
}
if (decision == Indeterminate) {
    return Indeterminate;
}
return NotApplicable;

Only-one-applicable:

Decision onlyOneApplicableCombiningAlgorithm(RuleOrPolicies[] rulesOrPolicies) {
    Boolean atLeastOne = false;
    RuleOrPolicy selectedRuleOrPolicy = null;
    ApplicableResult appResult;
    for (i = 0; i < lengthOf(rulesOrPolicies); i++) {
        appResult = isApplicable(rulesOrPolicies[i]);
        if (appResult == Indeterminate) {
            return Indeterminate;
        }
        if (appResult == Applicable) {
            if (atLeastOne) {
                return Indeterminate;
            } else {
                atLeastOne = true;
                selectedRuleOrPolicy = rulesOrPolicies[i];
            }
        }
        if (appResult == NotApplicable) {
            continue;
        }
    }
    if (atLeastOne) {
        return evaluate(selectedRuleOrPolicy);
    } else {
        return NotApplicable;
    }
}