A Security Framework for XML Schemas and Documents for Healthcare

Alberto De la Rosa Algarín,
Steven A. Demurjian
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT USA
{ada, steve}@engr.uconn.edu

Solomon Berhe
Department of Biomedical Informatics
Columbia University Medical Center
New York, NY USA
slb7002@dbmi.columbia.edu

Jaime A. Pavlich-Mariscal[1]
Departamento de Ingeniería de Sistemas
Pontificia Universidad Javeriana
Bogotá, Colombia
jpavlich@javeriana.edu.co

***Abstract*— The eXtensible Markup Language (XML) has wide usage in healthcare to facilitate health information exchange via the Continuity of Care Record (CCR) for storing/managing patient data, diagnoses, medical notes, tests, scans, etc. Health IT products like electronic health record (EHR, e.g., GE Centricity) and personal health record (PHR, e.g., MS Health Vault) use CCR for data representation. To manage patient data in CCR, security as governed by HIPAA must be attained when using XML and its technologies (XACML, XSLT, etc.). Our objective is to have an XML document (CCR instance) appear differently to authorized users at different times based on a user's role, constraints, separation of duty, delegation of authority, etc. In this paper, we propose a security framework that targets XML schemas and documents, in general, and CCR schemas and documents, in particular with control capabilities that achieve customizable access to an XML document's elements by applying secure software engineering methodologies and defining new UML XML-focused diagrams for schemas and permissions. This allows us to generate XACML policies, and enforce security at the runtime level on XML instances to insure that correct and required patient data is securely delivered. In a market of rapidly emerging mobile healthcare applications to allow patients to manage their own data (PHRs) and for self-management of chronic diseases, the need for secure access to information and its authorization and transmission to providers (and EHRs) will be critical.**

Keywords- XML schemas; role-based access control; continuity of care record; security policies and enforcement

I. INTRODUCTION

The eXtensible Markup Language (XML) [23] has become the de facto standard format for data exchange across heterogeneous systems, regardless of domain. In the biomedical informatics domain, XML has become the language of choice for most important standards, such as the health language seven (HL7) clinical document architecture (CDA) [14] for health information exchange (HIE) and the Continuity of Care Record (CCR) [28] for information storage of administrative, patient demographics, and clinical data. In health IT (HIT), CDA and CCR come together in various systems such as electronic health records (EHR, e.g., GE's Centricity [34]), practice management systems, personal health records (PHR, e.g., MS Health Vault [35]), etc. HL7 CDA is used to support HIE among hospitals, clinics, physician practices, laboratories, etc., with CCR providing the means to model the data that needs to be exchanged. The importance of both CDA and CCR has increased with the passage of the HITECH Act [33] which provides financial incentives for clinical institutions to implement EHRs and share electronic patient data with other organizations using HIE. CCR documents being exchanged among systems with confidential medical information must adhere to HIPAA [24] and regulations at state and federal levels. As a result, this must be addressed in a broader context, across multiple systems and accessible to multiple users in routine and emergent situations. We must expand security from each individual 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 and chronic diseases and need to securely access information and authorize its exchange with medical providers via EHRs, secure emails, or other means.

The main objective of this paper is to control access to XML documents to share and exchange information, providing a means for the security of an XML schema to be defined that can then be enforced on the individual XML instances for an application. We are seeking document-level access control to allow XML instances to appear differently to authorized users at different 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 as privileges are passed among authorized users, etc. In healthcare, such criteria will be further constrained by access to documents in emergent situations, collaboration of medical personnel in patient centered medical homes (PCMH) [37], delegating authority between providers during off hours (nights and weekends), etc. In all of these situations, the customizability of access to the document will be critical, to provide the ability to limit access to a CCR instance based on role; this may require security on the knowledge used to encode a document's information such as a medical ontology like SNOMED [37]. Towards this end, this paper proposes a

[1] Also affiliated with: Departamento de Ingeniería de Sistemas y Computación, Universidad Católica del Norte, Antofagasta, Chile, jpavlich@ucn.cl

security framework to define security policies that target XML schemas and documents, providing a variety of access control capabilities to achieve customizable access to an XML document's elements at an instance level.

To accomplish this, we leverage our work in secure software engineering [1, 2, 16, 17] using the unified modeling language UML [25], which has had a two-fold focus. First, in [16, 17], we created new UML diagrams for RBAC, discretionary access control (DAC), and users, augmented with a process for secure software engineering using UML; the approach defines a new UML role slice diagram from which aspect-oriented enforcement code is generated. Then, in [1,2], we enhanced the NIST RBAC [12, 13] standard with collaboration of duty and adaptive workflow to define security conditions under which users interact with one another towards a common goal; this work was applied to healthcare with UML diagrams extended appropriately. Our objective in this paper is to leverage both [1, 2] and [16, 17] in order to define a security framework for XML that: represents an XML schema in UML via a new *XML Schema Diagram*; defines security permissions via a new *XML Role Slice Diagram*; generates XACML [26] security policies; and, achieves the enforcement of security at the runtime level on XML instances to insure that filtered, correct, and required patient data is securely delivered. Our proposed framework is targeted for XML schemas and documents, but will be applied to healthcare and CCR.

The remainder of this paper is organized into five sections. Section II presents background information on NIST RBAC, XML, and the CCR standard. Section III presents a brief review of concepts from [16, 17] to establish the context of secure software engineering on UML. Section IV presents the proposed security framework for XML, focusing on new UML XML diagrams and the generation of XACML policies. Section V reviews related work, while Section VI, offers concluding remarks and ongoing work.

II. Background and Motivation

A. *NIST RBAC*

In the NIST RBAC [12, 13], permissions are assigned to roles, which are then assigned to users, shown in Figure 1, where a user can perform any of permissions assigned to the role s/he exhibits. NIST RBAC contains four reference models. $RBAC_0$ allows for policies to be denied at the role level instead of the individual level. To handle role hierarchies, $RBAC_1$ 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. $RBAC_2$ 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). $RBAC_3$ introduces the concept of sessions that represent the lifetime of a particular user, role, permission and their association for a dynamic runtime application.

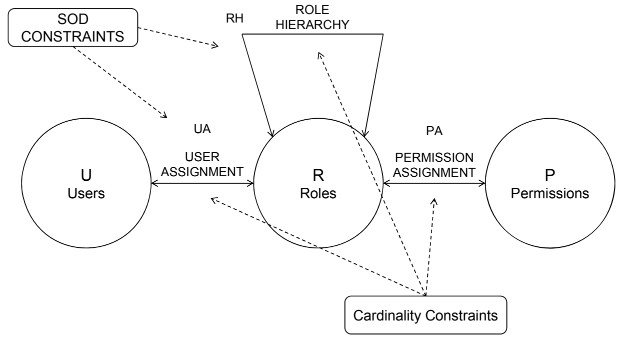


Figure 1: NIST $RBAC_0$, $RBAC_1$, and $RBAC_2$.

B. *The eXtensible Markup Language (XML)*

XML facilitates information exchange across disciplines and offers a flexible means to collect and transmit data between different information systems and platforms as a means of a common, structured language. XML supports information to be hierarchically structured and tagged, and the tags themselves can be exploited to capture and represent the semantics of the information. XML offers the ability to define standards via XML schemas, which serve as both the blueprint and validation agents for instances seeking to comply and be used for information exchange purposes. The main mechanism behind XML schemas is the XML Schema Definition (XSD), following the XML Schema language. As an example, 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, a characteristic shared with classes in UML. With XML schemas, the developer or standard proposing agency can determine constraints, such as the minimum or maximum amount of occurrences for an element (minOccurs, maxOccurs), the data type permitted for its value, and others. The schemas role is to describe and define the domain model, including the type-level characteristics that instances must follow in for validity.

C. *Continuity of Care Record*

Figure 2 shows a sample of the (a) official CCR schema [32] and (b) corresponding CCR document [29] that validates against said schema. CCR documents include sensitive patient information such as demographical information, social security number, insurance policy details and health related information (such as medications, procedures, psychological notes, etc.). The CCR document contains all patient information, but not all such information should be available to all users at all times; this information must be customized or filtered based on a user's role and responsibilities within an organization. For example, a secretary at a private practice performing financial operations might only need to see the patient's demographics and insurance policy details, whereas the primary physician may need to access the entire patient's information but not the SSN. Some information (e.g., pyschiatric notes) are only made available to speficic roles, One important note is that security policy changes must not result in updating the XML schemas and instances. As policies differ across institutions, the security model should offer mechanisms to handle this, which is one of the main objectives of our proposed security framework for XML.

```
<xs:schema xmlns="urn:astm-org:CCR" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ccr="urn:astm-org:CCR" targetNamespace="urn:astm-org:CCR"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="xmldsig-core-schema.xsd" />
<xs:element name="ContinuityOfCareRecord">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CCRDocumentObjectID" type="xs:string" />
        <xs:element name="Language" type="CodedDescriptionType" />
        <xs:element name="Version" type="xs:string" />
        <xs:element name="DateTime" type="DateTimeType" />
        <xs:element name="Patient" maxOccurs="2">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ActorID" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="From">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ActorLink" type="ActorReferenceType"
                                              maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
              ...
```

(a). XML Schema

```
<ContinuityOfCareRecord xmlns='urn:astm-org:CCR'>
    <CCRDocumentObjectID>Doc</CCRDocumentObjectID>
    <Language><Text>English</Text></Language>
    <Version>V1.0</Version>
    <DateTime><ExactDateTime>2008</ExactDateTime></DateTime>
    <Patient><ActorID>Patient</ActorID></Patient>
    <Body>
      <Problems>
        <Problem>
          <DateTime>
            <Type>
              <Text>Start date</Text>
            </Type>
            <ExactDateTime>2007-04-04T07:00:00Z</ExactDateTime>
          </DateTime>
          <DateTime>
            <Type>
              <Text>Stop date</Text>
            </Type>
            <ExactDateTime>2008-07-20T07:00:00Z</ExactDateTime>
          </DateTime>
          <Description>
            <Code>
              <Value>346.80</Value>
              <CodingSystem>ICD9</CodingSystem>
              <Version>2004</Version>
            </Code>
              ...
```

(b). XML Instance

Figure 2: Continuity of Care Record.

III. SECURE SOFTWARE ENGINEERING WITH UML

This section reviews prior work on UML diagrams for secure software engineering [16, 17] to elevate security to a first class citizen handled early in the software development process, and use extend this work to support XML-based security. UML provides multiple diagrams that can be leveraged to visually model domain requirements, but there is a lack of integrating security specifications for RBAC, DAC and MAC in the design phase [16, 17]. For this reason, the UML meta-model was extended with new UML security diagrams in Figure 3: *User Diagram* to grant a user permission to role(s); *Delegation Diagram* to define DAC characteristics during the design phase; *Role Slice Diagram* for the assignment or permissions (methods) to role; *MAC Extension* to define sensitivity levels (unclassified, confidential, secret and top secret) and their assignments to a user's clearance and an object's classification; and the *Secure Subsystem Diagram (SSD)* to identify the portions of the software application APIs that needs to be protected.

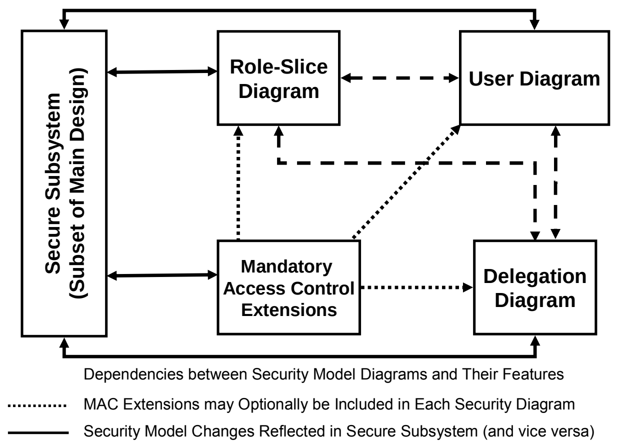

Secure Subsystem (Subset of Main Design)
Role-Slice Diagram
User Diagram
Mandatory Access Control Extensions
Delegation Diagram
Dependencies between Security Model Diagrams and Their Features
MAC Extensions may Optionally be Included in Each Security Diagram
Security Model Changes Reflected in Secure Subsystem (and vice versa)

Figure 3: UML Extensions for Access Control.

For the proposed XML security framework in Section IV, we employ the UML class diagram, the SSD, and the Role Slice Diagram (RSD). For instance, in a health care domain, the Secure Subsystem Diagram would represent the allowed access to portions of the Virtual Chart Application (VCA) that provides a common API across multiple EHRs connected via HIE, but not the entire API. SSD contains the set of classes and methods that are a subset of publically available ones to be protected through access control. The SSD is a specialized UML package, marked with the UML stereotype <<SecureSubsystem>> which contains a subset of the API classes each with a further a subset of the their methods. For example, Figure 4 illustrates the set of read/write operations that can be performed against the EHR (a subset of VCA API). Placing them into the SSD specifies that access to all of their methods must be controlled.

In order to limit access to the methods in Figure 4, the SSD extension to UML is utilized. As mentioned before, $RBAC_1$ supports permission role authorization and a role hierarchy. The work in [16, 17] has each role represented by a specialized UML package marked using the <<RoleSlice>> UML stereotype as given in Figure 5. Each such package is only allowed to contain UML classes. To increase the customizability of the permission role assignment, each method in the RSD contains is specialized: with: <<neg>> stereotype which specifies that this particular method cannot be accessed by the role, effectively turning off the methods from the parent and all it child roles; and, <<pos>> stereotype which specifies that this particular method is allowed to be accessed by the role. This assignment of negative and positive permissions to a particular role attains permission role authorization in RBAC. To further increase the role management, the role inheritance is supported by defining specialized relationships among role slice packages. The <<RoleInheritance>> UML stereotype specifies that the child role slice package inherits all positive methods from the parent role slice package. The example in Figure 5 illustrates Provider, Nurse, and Physician roles. The Provider Role has a set of methods authorized to many stakeholders. A Physician role is authorized to have read and write access to the entire EHR of the patient through the VCA. A Nurse role inherits all of the positive permissions from a base role except those for billing and appointment histories.

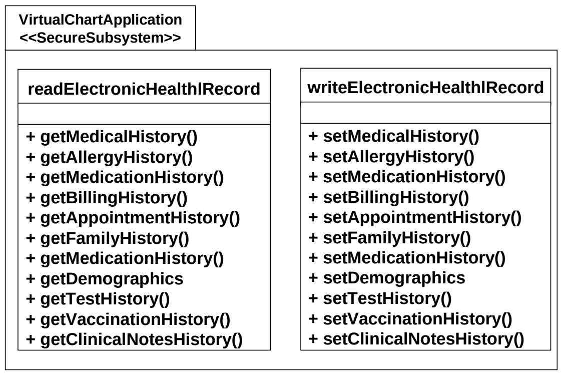


Figure 4: Secure Subsystem Diagram.

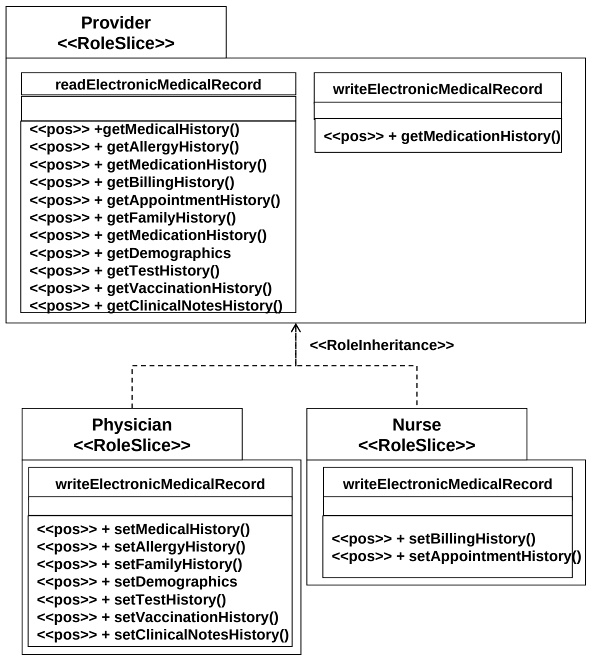

Provider
<<RoleSlice>>
readElectronicMedicalRecord
<<pos>> +getMedicalHistory()
<<pos>> + getAllergyHistory()
<<pos>> + getMedicationHistory()
<<pos>> + getBillingHistory()
<<pos>> + getAppointmentHistory()
<<pos>> + getFamilyHistory()
<<pos>> + getMedicationHistory()
<<pos>> + getDemographics
<<pos>> + getTestHistory()
<<pos>> + getVaccinationHistory()
<<pos>> + getClinicalNotesHistory()
writeElectronicMedicalRecord
<<pos>> + getMedicationHistory()
<<RoleInheritance>>
Physician
<<RoleSlice>>
writeElectronicMedicalRecord
<<pos>> + setMedicalHistory()
<<pos>> + setAllergyHistory()
<<pos>> + setFamilyHistory()
<<pos>> + setDemographics
<<pos>> + setTestHistory()
<<pos>> + setVaccinationHistory()
<<pos>> + setClinicalNotesHistory()
Nurse
<<RoleSlice>>
writeElectronicMedicalRecord
<<pos>> + setBillingHistory()
<<pos>> + setAppointmentHistory()

Figure 5: Role Slice Diagram.

IV. PROPOSED SECURITY FRAMEWORK FOR XML

In this section, we present the proposed security framework for XML schemas. The general approach is to have a set of XML schemas corresponding to an application (upper right in Figure 6), which will be instantiated for the executing application (bottom right of Figure 6). From a security perspective, our intent is to insure that when users attempt to access the instances, that access will be customized and filtered based on their defined user role and associated security privileges (bottom left of Figure 6). In a healthcare setting, a secretary may only have access to patient demographics, a nurse only able to write portions of the data, a physician more broader access. Our approach to secure software engineering (see Section III again) is to insure that these different views of the schemas are applied to the users executing their respective desktop and mobile HIT applications against their authorized instances (patients). To achieve this approach, Section IV.A, presents a new UML class diagram called a *XML Schema Class Diagram (XSCD)* to transition an XML schema into a UML like diagram and notation. This adds a degree of software engineering to the XML design process. We also define a new UML *XML Role Slice Diagram (XRSD)* that extends Figure 5 and allows permissions to be defined against XML schema elements in the XSCD. Then, Section IV.B explores the transition of these XSCDs into a corresponding security policy to automatically generate XACML for enforcement of the XML schema at the instance level; the XCSD in combination with XRSD allows an XACML policy to be defined. This may necessitate the interception of various XMLs tools in order to allow the security check to occur; e.g., using XSLT must only return instances that are allowed and which portions are allowed for a user playing a given role based on defined permissions.

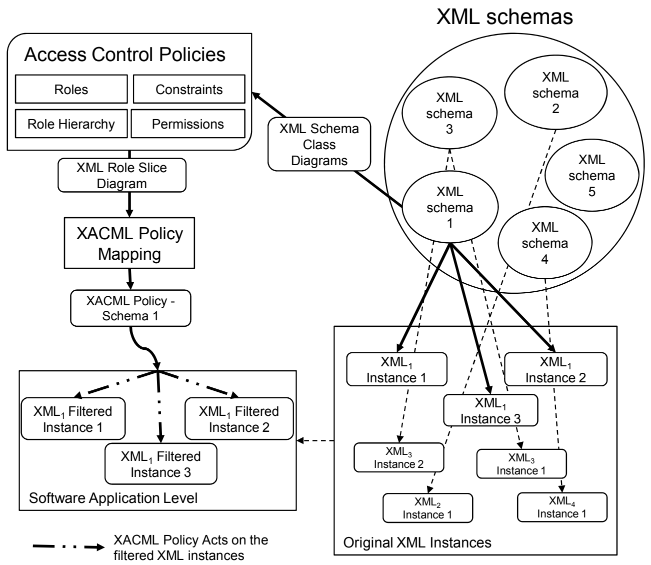
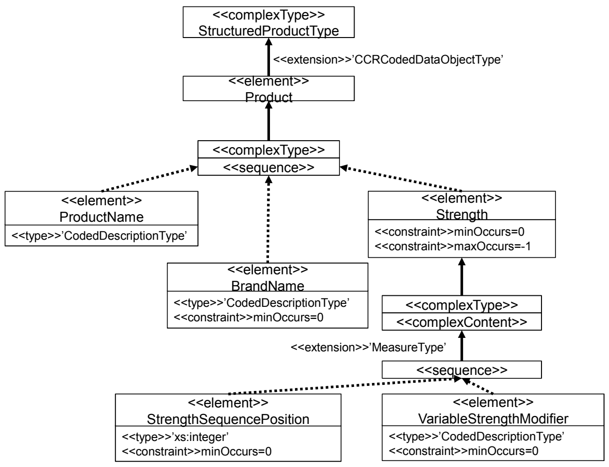


Figure 6: Overview of XML Security Framework.

A. *New UML Schema Class and Role Slice Diagrams*

In this section, we present a new UML *XML Schema Class Diagram (XSCD)* that contains architecture, structure characteristics, and constraints. The set of all XML schemas for a given application are converted into a corresponding set of XSCDs. The intent is to provide a degree of software engineering to the XML design process; rather than just haphazardly building schemas and deploying instances, the creation of an XML schema should be placed into the UML context alongside other diagrams (class, use case, sequence, activity, etc.) that all have the potential to impact the content of a XML diagram for an application. Included in this work is an utilization of the concepts of the RSD from [16, 17] to extend and define a new *XML Role Slice Diagram (XRSD)* which has the ability to add permissions to the various elements of the XSCD, i.e., read, write, no read, no write, or any combination of these, such as read/write, read/no write, etc. The original RSD in Figure 5 focused on <<pos>> and <<neg>> permissions on methods, while XRSDs will emphasize the data aspects that are the focus of an XML schema as modeled in XSCD. In the literature, approaches to translate a defined XML schema into a UML diagram [3, 4, 9, 19, 21] that each provides varying levels of support for model groups, elements, attributes, and identity constraints [4], depending on the approach utilized (e. g., the UML meta model or UML profile, a combination of the two, etc.).

In our approach, the new UML XML *Schema Class Diagram (XSCD)*, shown in Figure 7, is an artifact that holds all of the characteristics of the XML schema, including structure, data type constraints, and value constraints. While the process of the way that the XSCD is built from the original XML schema is out of the scope of this paper, we explain briefly using the CCR schema. Recall that XML schemas are characterized by a hierarchical structure with data type constraints. Another possibility of XML schemas is referencing other XML schemas that provide another layer of structure and constraints. We specify two issues to address when designing the modeling capabilities of XSCD: XML schema references must be supported, and the XML structure and constraints must be maintained.

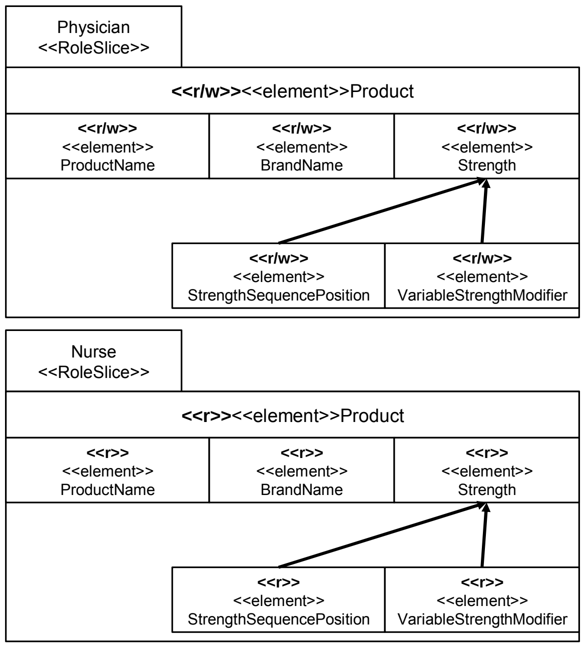


<<complexType>>
StructuredProductType
<<extension>>'CCRCodedDataObjectType'
<<element>>
Product
<<complexType>>
<<sequence>>
<<element>>
ProductName
<<type>>'CodedDescriptionType'
<<element>>
Strength
<<constraint>>minOccurs=0
<<constraint>>maxOccurs=-1
<<element>>
BrandName
<<type>>'CodedDescriptionType'
<<constraint>>minOccurs=0
<<complexType>>
<<complexContent>>
<<extension>>'MeasureType'
<<sequence>>
<<element>>
StrengthSequencePosition
<<type>>'xs:integer'
<<constraint>>minOccurs=0
<<element>>
VariableStrengthModifier
<<type>>'CodedDescriptionType'
<<constraint>>minOccurs=0

Figure 7: Proposed UML Schema Class Diagram.

To handle the hierarchical nature of XML schemas, for XSCD in Figure 7, 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 in Figure 7. 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 7 presents the way that the XSCD for the CCR's schema xs:complexType 'StructuredProductType' would look after the transformation process (note that the figure does not include all children nodes from the CCR due to of space limitations). This XSCD implementation allows for customized access control policies to be generated for the respective concepts of the XML schema.

The next step in the process is to apply security policies to the XSCD (top left of Figure 6) that are consistent with [16, 17] where <<pos>> and <<neg>> permissions were used to limit methods at the API level. Correspondingly, for XSCD, we utilize the RSD (see Figure 5) to define a new *XML Role Slice Diagram, XRSD* (see Figure 8) that is capable of applying access control policies or permissions on the attributes of the XSCD, akin to applying such policies to the private or public data of a class. Figure 8 has two XRSDs, Physician and Nurse, that augment their counterparts in Figure 5 as extended role slices (<<RoleSlice>>). To accomplish this, we also extend the list of stereotypes to represent allowance or denial of access.



Physician
<<RoleSlice>>
<<r/w>><<element>>Product
<<r/w>> <<element>> ProductName
<<r/w>> <<element>> BrandName
<<r/w>> <<element>> Strength
<<r/w>> <<element>> StrengthSequencePosition
<<r/w>> <<element>> VariableStrengthModifier
Nurse
<<RoleSlice>>
<<r>><<element>>Product
<<r>> <<element>> ProductName
<<r>> <<element>> BrandName
<<r>> <<element>> Strength
<<r>> <<element>> StrengthSequencePosition
<<r>> <<element>> VariableStrengthModifier

Figure 8: XML Role Slice Diagrams.

The nature of XML documents warrants the implementation of read, no read, write, and no write permissions. These permissions would have their respective stereotypes, <<r>> (read), <<nr>> (no read), <<w>> (write), and <<nw>> (no write). Figure 8 defines Physician and Nurse XRSDs that indicate the required permissions against the XSCD in Figure 7. Note that the CCR complexType 'StructuredProductType' element Product (see Figure 7) is defining the ability for a role to have read and write permissions (Physician) (represented by a combined stereotype <<r/w>>) and for a role with only read permissions (Nurse) (represented by the stereotype <<r>>). While a Physician role might be able to get all of the information regarding a drug and be able to create new instances following the CCR schema, a Nurse role might only need to read the drug details and not be allowed to create new records. In this case, for the Physician role, all elements have read/write permission; while in the case of the Nurse role, elements only have a read permission (though in practice, a nurse would have limited writes to record vital signs, patient notes, etc.).

B. XACML Polices from XSCD and XRSD

The work of [16, 17] presented a formal enforcement framework that could automatically translate SSDs and RSDs into aspect-oriented enforcement code. We leverage this concept by automatically generating security policies in XACML that use the XSCDs (Figure 7) and their associated XRSDs (Figure 8). To achieve this objective, we utilize the XACML specification 3.0, which offers a mechanism to implement vital parts of our proposed framework. Figure 9 illustrates the overall architecture to generate XACML policies from the XSCDs and XRSDs to be applied on XML schemas and associated instances at runtime. This includes the Policy Retrieval Point which is in charge of housing the security policies) of the XACML architecture that is used for enforcing security policies. The components of this architecture include: the Policy Enforcement Point (PEP), which acts as the bridge between a request and the Policy Decision Point (PDP); PDP which evaluates the request and provides a response according to the policies in place; and, Policy Access Point (PAP) to write and manage policies; which can invoke attributes, values, and subject information from the Policy Information Point (PIP). In order to produce XACML security policies, a mapping of the access control reflected in the XRSDs for the XSCDs is used and applied so that the permissions in the XRSDs are captured by XACML policies. This makes it possible to derive a security policy that could not only act on the software application level, but more importantly at the document-level.

These security policies generated from XSCDs and their XRSDs must be able to control the way that XSLT and other query tools (e.g., XPath [38], XQuery [39], etc.) handle the reading/querying of the XML schema and instances (lower left of Figure 6). Since the expressions from XPath act in the same way as of those for a relational database's queries, the XACML policy can target the expressions, allowing or denying access. We recall that the instances utilized at this point are modified depending on the XACML schemas themselves, filtered depending on the role of the user and usage context. As per the previous example (physician role and nurse role, with their respective permissions), the physician and nurse could perform the XPath expression */StructuredProductType/@Product* to obtain the entire set of Product attributes from StructuredProductType of the respective XML instance. This is possible since their roles permit them to 'see' the structure of the schema as well as permissions of reading obtained from the XRSDs. If another role, such as MedicalOrderly does not have sufficient permissions and tries to execute the previous XPath expression, both a denied access (per XACML access control) and invalid call (per the filtered XML instance) would be the result.

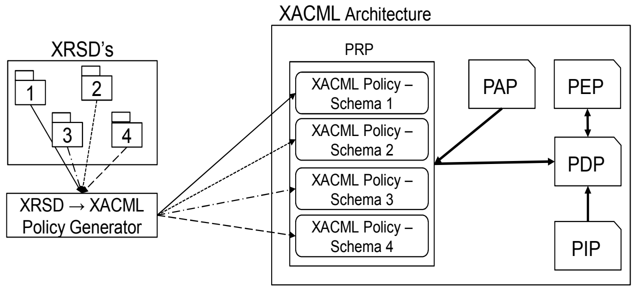


Figure 9: Generated XACML Policies.

Update, insertion and deletion operations are not yet supported by XPath or XQuery (there are proposals as a non-normative extension to XPath [38] and utility updates to XQuery [39]). Thus, writing back to XML instances is handled as a method call from the original software design. In these cases, security can be applied by using our framework by treating such operations as API methods, and applying security as [16, 17]. For example, there would be a write of the entire patient XML instance that would only success if the fields modified were allowable permissions for the user playing a particular role. Figure 10 shows an example XACML policy that illustrates the Physician and Nurse roles, with the MedicalOrderly role, handling reading permissions with a target to the schema elements.

V. Related Work

Access control enforcement in XML has two typical approaches. First, the enforcement can be done as query rewrites, where these are generated depending on the access control policy. Second, the enforcement can be embedded into the XML schema and documents to provide different views based on the policies in place. This section presents related work in both approaches as compared to our own.

The work of [10] 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 also 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, but differ in two ways. First, the work targets XML DTD's, which have been replaced by XML schemas. Second, the security policies are embedded into both the DTD and the instance. When policies experience a change, the cost of updating the XML instances is huge.

Another effort [11] details a model that tries to combine the two discussed methodologies 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. This is similar to our work in that it ultimately targets security in XML document instances via XACML policies, but our work differs by also taking into consideration XML document writing (XPath's design only allow it to perform reading queries to the XML instance).

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy>
    <Description>EMR Read Access Control Policy</Description>
    <Target>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:ruleexample:simpleRule1"
Effect="Permit">
        <Description> Any subject with role "Physician" or "Nurse" can read
<<StructuredProductType>> and subsequent elements.</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match
                    MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
                    <AttributeValue
DataType="CCR:schema:StructuredProductType">StructuredProductType
                    </AttributeValue>
                    <AttributeDesignator MustBePresent="false"
                         Category="urn:oasis:names:tc:xacml:role-category:access:role"
                              AttributeId="urn:oasis:names:tc:xacml:role:role-id"
                         DataType="urn:oasis:names:tc:xacml:data-type:rfc822Name"/>
                     </Match>
                 </AllOf>
             </AnyOf>
         </Target>
     </Rule>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:ruleexample:simpleRule1"
Effect="Deny">
        <Description>Any subject with role "MedicalOrderly" cannot read
<<StructuredProductType>> and subsequent elements.</Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match
                    MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
                    <AttributeValue
DataType="CCR:schema:StructuredProductType">StructuredProductType
                    </AttributeValue>
                    <AttributeDesignator MustBePresent="false"
                         Category="urn:oasis:names:tc:xacml:role-category:access:role"
                              AttributeId="urn:oasis:names:tc:xacml:role:role-id"
                         DataType="urn:oasis:names:tc:xacml:data-type:rfc822Name"/>
                     </Match>
                 </AllOf>
             </AnyOf>
         </Target>
     </Rule>
</Policy>
```

Figure 10: Example XACML Policies for Reading Permissions.

The encryption of different sections of an XML document with different encryption keys is presented in [5]. These keys are then distributed to the specific users based on the access control policies in place. Special focus is given content-based access control, and users are granted or denied access based on their credentials (not roles, as in our approach). This makes it difficult to handle policies such as role-delegation, time and value constraints, unless they are handled at the application level.

Efforts by [6, 7] present Author-X, a Java-based system for DAC in XML documents, and provides customizable protection to the documents with positive and negative authorizations. Author-X employs a policy base DTD document that prunes an XML instance based on the security policies (similar to our approach), but focuses on discretionary access control (different to our approach of RBAC and its extensions and its lack of XML schemas).

Another example of embedding access control policies into the XML DTD and instances is proposed by [8, 22] via a usage control model allows for a more custom control than the more commonly used access control models. By embedding security into documents, changes to security have broad impact on instances. When security policies change, the cost of re-securing all created instances is directly proportional to the amount of instances.

Work by [18] presents a distributed access control model for collaborative environments where XML documents are used. The proposed framework utilizes a cryptographic methodology, employing a key management scheme to enforce security policies (much different to 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, unlike our approach.

VI. Conclusion and Ongoing Work

XML is a dominant player in data and information exchange, playing a pivotal role in health care via HL7 and CCR standards. Given the diverse set of stakeholders in healthcare, there is a need to provide customized and controllable access to XML instances on confidential patient information to allow different users different views of authorized instances at different times. To address this problem, we have proposed a security framework for XML schemas and documents with a number of unique features and characteristics. To underline the proposed framework, we make use of our research in secure software engineering with UML [1, 2, 16, 17] that introduces new UML diagrams for RBAC, DAC, MAC, and collaboration (see Section III). Using this work as a basis, we include the XML schema definition as part of the secure software engineering process and have proposed a UML XML Schema Class Diagram, XSCD to capture data/information exchange requirements, and a UML XML Role Slice Diagram, XSRD, to capture permissions against the XSCD (see Section IV.A). With these in hand, we demonstrate the ability to generate XACML security policies for XSCD and its security XRSD, providing the ability for different instances to appear different to authorized users at different times (see Section IV.B), which is achieved by intercepting XML tools.

By tackling the information security problem with a software engineering approach, the proposed framework creates a relation between secure software and secure data. We present a duality from our previous research (securing API methods) to the research presented here (securing elements in XML schemas). We take this generalization to present that the data utilized in a software system can be secured in a similar manner as the system itself, and it can be done at the design phase of the software engineering process.

Our ongoing work includes the formalization of the UML Schema Class Diagram, so that schemas such as those that reference other schemas are also handled. This includes extending the framework to handle access control constraints such as role delegation and collaboration of duty extensions to RBAC [1, 2]. We have also begun prototyping efforts for the proposed framework in two directions: we are modifying mobile apps for medication management to filter CCR instances from MS Health Vault based on role; and, we are exploring extensions to handle the non-normative additions to XPath and XQuery that support XML instance modification methods (update, insertion, deletion), that verifies for security assurance and policy consistency. Larger scoped worked includes the ability of having an integrated UML and XML secure software engineering process, and the extension of our proposed framework to other knowledge encoding structures, such as ontologies (more specifically, those represented in OWL [31]), since CCR data in EHRs is highly tied to medical ontologies.

REFERENCES

[1] S. Berhe, et al., "Secure, obligated and coordinated collaboration in health care for the patient-centered medical home," in *AMIA Annual Symposium Proceedings*, vol. 2010. American Medical Informatics Association, 2010, p. 36.

[2] S. Berhe, et al., "Leveraging uml for security engineering and enforcement in a collaboration on duty and adaptive workflow model that extends nist rbac," *Data and Applications Security and Privacy XXV*, pp. 293–300, 2011.

[3] M. Bernauer, et al., "Representing xml schema in uml - an uml profile for xml schema," *Citeseer, Tech. Rep*., 2003.

[4] M. Bernauer, et al., "Representing xml schema in uml – a comparison of approaches," *Web Engineering*, pp. 767–769, 2004.

[5] E. Bertino et al., "Secure and selective dissemination of xml documents," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 3, pp. 290–331, 2002.

[6] E. Bertino, et al., "Protection and administration of xml data sources," *Data & Knowledge Engineering*, vol. 43, no. 3, pp. 237–260, 2002.

[7] E. Bertino, et al., "Access control for xml documents and data," *Information Security Technical Report*, vol. 9, no. 3, pp. 19–34, 2004.

[8] J. Cao, et al., "Towards secure xml document with usage control," *Web Technologies Research and Development-APWeb 2005*, pp. 296–307, 2005.

[9] D. Carlson, "Uml profile for xml schema," 2008.

[10] E. Damiani, et al., "Design and implementation of an access control processor for xml documents," *Computer Networks*, vol. 33, no. 1, pp. 59–75, 2000.

[11] E. Damiani, et al., "A general approach to securely querying xml," *Computer Standards & Interfaces*, vol. 30, no. 6, pp. 379–389, 2008.

[12] D. Ferraiolo, et al.. CISM, "Role-based access control (rbac)," in *Proc. of 15th NIST-NSA National Computer Security Conference*, 1992.

[13] D. Ferraiolo, et al., "Proposed nist standard for role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[14] R. Dolin, et al., "Hl7 clinical document architecture, release 2," *Journal of the American Medical Informatics Association*, vol. 13, no. 1, pp. 30–39, 2006.

[15] D. Kibbe, et al., "The continuity of care record." *American family physician*, vol. 70, no. 7, pp.1220–1222, 2004.

[16] J. Pavlich-Mariscal, et al., "A framework of composable access control definition, enforcement and assurance," in *Chilean Computer Science Society*, 2008. SCCC'08. International Conference of the. IEEE, 2008, pp. 13–22.

[17] J. Pavlich-Mariscal, et al., "A framework for security assurance of access control enforcement code," *Computers & Security*, vol. 29, no. 7, pp. 770–784, 2010.

[18] M. Rahaman, et al., "Distributed access control for xml document centric collaborations," in *Enterprise Distributed Object Computing Conference*, 2008. EDOC'08. 12th International IEEE. IEEE, 2008, pp. 267–276.

[19] N. Routledge, et al., "Uml and xml schema," in *Australian Computer Science Communications*, vol. 24, no. 2. Australian Computer Society, Inc., 2002, pp.157–166.

[20] R. Sandhu et al., "Access control: principle and practice," *Communications Magazine*, IEEE, vol. 32, no. 9, pp. 40–48, 1994.

[21] D. Skogan, "Uml as a schema language for xml based data interchange," in *Proceedings of the 2nd International Conference on The Unified Modeling Language (UML'99)*. Citeseer, 1999.

[22] L. Sun et al., "Dtd level authorization in xml documents with usage control," *International Journal of Computer Science and Network Security*, vol. 6, no. 11, pp. 244–250, 2006.

[23] [Online]. Available: http://www.w3.org/XML/

[24] [Online]. Available: http://www.hhs.gov/ocr/privacy/

[25] [Online]. Available: http://www.uml.org/

[26] [Online]. Available: http://docs.oasis-open.org/xacml/

[27] [Online]. Available: http://www.w3.org/XML/Schema

[28] [Online]. Available: www.astm.org/Standards/E2369.htm

[29] [Online]. Available: http://goo.gl/G5wTR

[30] [Online]. Available: http://www.omg.org/spec/XMI/2.4.1/

[31] [Online]. Available: http://www.w3.org/TR/owl-features/

[32] [Online]. Available: http://goo.gl/FmGSp

[33] [Online]. Available: http://healthit.hhs.gov/programs

[34] [Online]. Available: http://centricitypractice.gehealthcare.com/

[35] [Online]. Available: http://www.microsoft.com/en-us/healthvault/

[36] [Online]. Available: http://www.ncqa.org/tabid/631/default.aspx

[37] [Online]. Available: http://www.ihtsdo.org/snomed-ct/

[38] [Online]. Available: http://www.w3.org/TR/2002/WD-xpath20-20020430/

[39] [Online]. Available: http://www.w3.org/TR/xquery-update-10/