6-19-2014

An RBAC, LBAC and DAC Security Framework for Tree-Structured Documents

Alberto De la Rosa Algarin
University of Connecticut - Storrs, alberto.delarosa.algarin@engr.uconn.edu

An RBAC, LBAC and DAC Security Framework for Tree-Structured Documents

Alberto De la Rosa Algarín, Ph.D.
University of Connecticut, 2014

Information modeling is focused on representing, using, and exchanging information in large-scale applications that include law-enforcement, healthcare, e-commerce, and others. Information modeling, as achieved by varied data formats, creates new security challenges. First, there is a need to integrate the security requirements of existing information applications that use and exchange information in via tree-structured documents. Second, there exists a need to consolidate this security in support of a newly developed information system. Third, we ask if is it possible to develop an approach for security for information applications that is able to reconcile the security policies across potential constituent component systems in an information exchange scenario. In this dissertation we present a security framework aimed towards an approach to modeling the security of information at global and local levels. This framework leverages the three major access control models: RBAC, LBAC, and DAC to achieve security assurance throughout varied scenarios. First, we introduce a security model that creates the base for the rest of the framework. This security model considers the access control requirements as realized in tree-structured documents with schemas. Second, we extend UML model and metamodel layers with new diagrams that provide a graphical notation for the security model. Third, we present a mapping process between the UML diagrams and the XACML that yields security policies ready to be deployed. Last, towards the overall purpose of the dissertation, we advance the information security problem to a software engineering perspective, elevating information security to a first-class citizen of the

software design and development process, resulting in *secure information engineering*. By tackling the problem from a perspective of tree-structured documents, any data format that is represented by such a structure (e.g. XML, specialized JSON structures, RDF, OWL, etc.) can be secured. This effectively allows us to provide separation of concerns with respect to information security by defining security requirements in one software process phase and generating enforcement policies in another phase. These enforcement policies are not embedded in the system, on the contraire, they are agents evaluated and enforced in the overall security architecture of the application.

An RBAC, LBAC and DAC Security Framework for Tree-Structured Documents

Alberto De la Rosa Algarín

B.S., Computer Science | Mathematics, University of Puerto Rico, Puerto Rico, 2010

M.S., Computer Science and Engineering, University of Connecticut, USA, 2013

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2014

APPROVAL PAGE

Doctor of Philosophy Dissertation

An RBAC, LBAC and DAC Security Framework for Tree-Structured Documents

Presented by

Alberto De la Rosa Algarín

B.S., Computer Science | Mathematics, University of Puerto Rico, Puerto Rico, 2010

M.S. Computer Science and Engineering, University of Connecticut, USA, 2013

Major Advisor: _____

Dr. Steven A. Demurjian

Associate Advisor: _____

Dr. Jinbo Bi

Associate Advisor: _____

Dr. Swapna Gokhale

Associate Advisor: _____

Dr. Xiaoyan Wang

University of Connecticut
2014

ACKNOWLEDGEMENTS

First, I want to give my heartfelt thanks to what I consider is the best professor and advisor a graduate student can hope to have, Dr. Steven A. Demurjian. Your guidance in research have made me the researcher I am today and the one I will be in the future. I appreciate your continuous motivation by giving me new responsibilities, your encouragement to take on new opportunities and your risk taking by allowing me to branch out and seek new, unproven possibilities without much supervision. Your wisdom in the biomedical and software engineering domains has opened my mind to all different areas I didn't consider, and I know without a doubt that I would have not achieved as much as I have without your major guidance. I can only hope to have lived up to your expectations and standards as a graduate student, and to become a researcher of your caliber in the future.

I would also like to thank the members of my doctoral committee, Prof. Jinbo Bi, Prof. Xiaoyan Wang and Prof. Swapna Gokhale. Prof. Bi, you taught me my first graduate level class at UConn. I remember walking into your classroom, a student straight out of undergraduate level-courses, without much knowledge. Your ability to explain the computational and informatics problems of the healthcare domain made it easy for me to come up with crazy ideas. It was your help with one of those ideas that allowed me to write a proposal for the NSF GRFP, a learning experience on its. Prof. Wang, you gave me the opportunity to write my first research paper aimed for publication. Your knowledge made me appreciate natural language processing (both its incredible uses and its complexity), which ended up being the research topic for my first class with Prof. Demurjian. Your continuous support and encouragement to pursue

publishing the paper made me think more about the area of research, which resulted in me realizing my interest in security (the focus of that original paper). I know it must have not been easy to help me in that initial endeavor, considering my limited knowledge on the existing research, so I greatly appreciate your patience. Last, but not least, Prof. Gokhale. You offered me the first opportunity to branch out of biomedical informatics, security and everything I had considered was my focus point, during your social networks class. The research paper presentations made me understand the power and exceedingly value of information (and I admit made me a little bit paranoid with respect to social networks), something that I always have in mind as motivation for my research. It made me even more excited to know that the effort we had dedicated to the research resulted in two posters and a conference publication.

Next are my dear friends, those who I met when my graduate career started and those whom I met along the way. Without you, graduate school would have been a different experience. Specially, I want to thank Michael Zuba. Never in my mind would I have thought that the kid I met in a random dorm meeting at the beginning of the fall 2010 semester would become my best friend and brother. ‘Bro’, you know you saved my life my first and second years of graduate school by taking me grocery shopping when I needed to. The times we decided to not worry about work and steam off playing videogames were awesome. It comes straight from the heart when I say I am glad to have met you and Tierney (so is Elisa), you guys both rock and are amazing human beings. I wish you both nothing but the best, I hope you both become a successful and important people so I can tell everyone I know someone famous. Next up, Armando. You and I have known each other since undergraduate computer science. I have to admit, it made me

happy to know that you wanted to pursue a Ph.D. in the field, knowing the amazing potential and talent you have. You have made lunchtime the most hilarious time of the day, something to look forward to with our semi-philosophical, semi-comedic discussions in lab and Husky Pizza. I am proud to call you my friend. I hope you get nothing but the best in life, and swift success in your goals. My other dear friends, those who are in UConn (Gino Sanzi, Yaira Rivera, Solomon Berhe) and those who are not (Timoteus Ziminski, Rishi Saripalle, Antonio Cusano, Ryan McGivern, Brendan Heckman, Joseph Gilbert), you either taught me things I knew nothing about, or made me laugh with the random conversations we had in lab (or both!). I know you are all achieving your dreams, so I can only wish you smooth sailing onward.

The most important group of people in my life, my family, deserves a big acknowledgement. First, Yahaira. Sis, your achievements and desire to learn more have been one of the biggest pushers to my pursuit of knowledge. Your selflessness and constant search of better things taught me to never settle and always strive for better. I could have not asked for a better role model. I guess we both turned out how we are thanks to mom and dad. On that note, mom (mami) and dad (papi). Mami y papi, I cannot even express in words the love I have for you both. Your daily phone calls ever since I stepped on Connecticut were beacons of tranquility on what otherwise would have been hectic days. Even those days where the phone calls reduced to obvious questions like ‘how cold is it?’, to which 90% of the time the answer was always ‘very, very cold’, made me feel like I was back home with you. Your worry and constant concern would test anyone’s patience, but I know you are like that because of your love. Even with a 60 to 90-degree difference, 1,500 miles of distance and 1-hour difference (sometimes), you

both made me feel like I was in Puerto Rico every day. I know it was not easy seeing me part every time I went home and my ‘vacation’ was over (it surely was not easy for me), even when you went over the same thing with Yahaira years before, but I do appreciate and enjoy to the fullest the days I could spend with you. I can only hope to have made you proud, as you make me proud of having you as parents. I love you all.

Last, but definitively not least, is the most special person in my life. My soul mate, best friend, partner in adventure and mischief, Elisa. I do not lie when I say that without you, graduate school would have been exponentially harder. I am extremely thankful for you; you are the most amazing person I know. Your resilience motivates me; your love and humility push me to be a better person. I love you so much, your dreams are my dreams, your goals are my goals and your success is my mine of happiness. You are my source of inspiration, always making me think outside the box. The adventure we called graduate school, which we both started 4 years ago, has been amazing. It seems only fitting that, since I gift you a book of our yearly adventures, this dissertation will be the one book that caps off the period of time we were both working towards our dreams. I know this one does not have any pictures of us in cool or funny places, but as I write this, I cannot help but think and remember all the fun we had the last four years while trying to get away from the stress of school. This book represents all that. I have said this before, but I will say it again since I have some space left in the page: you are my beacon of light, my partner in adventure, my best friend, my soul mate. Elisa, I love you with all my heart. I cannot wait to see what is next in our lives.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1
Introduction

Information modeling is focused on representing, using, and exchanging information in large-scale applications that include: law-enforcement repositories that use the model minimum uniform crash criteria data model standard (Ogle, Alluri, & Sarasua, 2011); healthcare collaborative and non-collaborative scenarios that use the Health Level 7's clinical document architecture (Alschuler, Mair, Boyer, & Dolin, 2002; Dolin et al., 2006) or the continuity of care record (Ferranti, Musser, Kawamoto, & Hammond, 2006; Kibbe, 2005; Kibbe, Phillips, & Green, 2004) to store all types of data associated with patients; and, cXML (Merkow, 1999) for e-commerce communication between procurement applications. These information systems may support a wide range of data formats such as: the eXtensible Markup Language (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 1998) that models documents as schemas and instances; the resource definition framework (Klyne, Carroll, & McBride, 2004) that brings semantics to the web; the web ontology language (McGuinness & Van Harmelen, 2004) that is used for applications that need to process semantic information instead of just its presentation; and, the JavaScript Object Notation (Crockford, 2006; Lanthaler & Gütl, 2012), which uses a programming like notation to model data interchange; etc. These information applications data often have a tree structure of index and entity nodes that allows for information to be modeled via schemas (that define structure) which can be used as blueprints for the creation of new documents (instances) and their validation (enforcement). In such settings, where sensitive data is utilized by users for time-critical applications, security that is achieved via access control is a paramount concern. These applications all present unique

challenges to the objective of providing a high-degree of protection to information. This is illustrated in Figure 1.1, where a role-based access control (Ferraiolo, Sandhu, Gavrila, Kuhn, & Chandramouli, 2001; Liebrand, Ellis, Phillips, & Ting, 2002) approach is represented as a user attempting to access the instances of an application. This is accomplished by utilizing the role of the user to determine the required instances of the application that are authorized by role to the user (top half of Figure 1.1) which is balanced with the enforcement of the access of the relevant instances via security schemas for roles, users, and constraints, that are able to filter the application instances before they are delivered to the user (bottom half of Figure 1.1). In the scenario as given in Figure 1.1, the ability to provide granular access control in support of information modeling for structured documents is based on security policies defined in an local manner (e.g., institutions) and guidance defined and enforced at a more global scope (e.g., legal entities and active pieces of legislation), has proven to be difficult to achieve.

From a system perspective, today's information applications are designed and developed with the purpose of leveraging multiple technologies. Offering multiple application programming interfaces (APIs), cloud computing capabilities, web services such as the representational state transfer (Masse, 2011) and simple object access protocol (Parastatidis, Webber, Woodman, Kuo, & Greenfield, 2005), data mining capabilities, etc., have in turn increased the number of data formats (XML, RDF, JSON, OWL, etc.) used to both represent (model) and exchange information. In turn, the usage and information exchange of data has increased across information applications such as library repositories, collaborative environments, healthcare, etc., accompanied by the development of standardized document structures that promote and simplify

interconnection between systems and applications. As a process, information exchange 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.

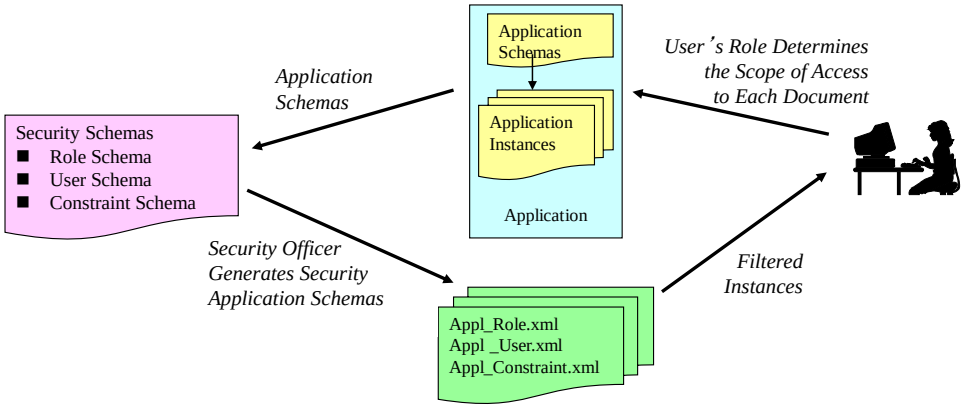

Figure 1.1: Document-based Secure Information Access.

Information modeling as achieved by varied data formats creates new security challenges. First, there is a need to integrate the security requirements of existing information applications that use and exchange information in via tree-structured documents. Second, there exists a need to consolidate this security in support of a newly developed information system. Third, we ask if is it possible to develop an approach for security for information applications that is able to reconcile the security policies across the constituent component systems in order to control the instances that are utilized and shared. These three challenges in turn drive us to the main research questions that involve the ability to bring together multiple systems in support of a larger overriding information application. How do we provide a solution at an information modeling level that operates across various contexts? Is there be an approach that is capable of integrating the local security of the individual interacting information systems into some higher-level global security mechanism? Is it possible to integrate different access control models (e.g.,

lattice-based access control (LBAC) (Sandhu, 1993), role-based access control (RBAC) (Ferraiolo et al., 2001), and discretionary access control (DAC) (Dittrich, Hartig, & Pfefferle, 1989; Downs, Rub, Kung, & Jordan, 1985; Sandhu & Samarati, 1994)) into a single framework that is capable to providing a diverse and flexible approach to information security? Finally, how can we enforce security across multiple interoperating systems? The constituent systems, services, and technologies might have a varied field of security requirements realized through different security approaches. The main challenge for a newly developed information application that employs tree-structured document formats is to utilize to all of the resources that are available in the constituent information systems allowing their secure access to be achieved.

The work presented in this dissertation provides a framework that allows for tree-structure documents for a new information application to be modeled, utilized, and exchanged in conjunction with the constituent information systems (and their documents and services). The proposed work supports the definition of security at a model level (schemas) that is then enforced on the executing information application to customize the information (instances) that is to be utilized and exchanged. Specifically, the proposed work seeks to control the information security of an application by providing the ability to define access control permissions across LBAC, RBAC, and DAC for the information schemas in an application and their respective instances. For LBAC, the ability to define sensitivity levels (clearance for users and classifications for data) for tree structured documents (at the schema level) that provide a robust access control mechanism to control what can be seen from each authorized instance. For RBAC, the ability to define on with specific permissions on the tree structured documents (at the schema level) that

are then filtered to show custom instances that is delivered to a user is a needed requirement. Lastly, for DAC, the capability of defining authorizations that can be delegated from one user to another allows the access control (LBAC and/or RBAC) at the schema level to be delegated. 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 healthcare, banking, infrastructure and national defense

The result of all of these supports fine-grained access control of instances for: non-destructive (document-level) operations that utilize instances as a source of information (e.g., read and aggregate); destructive (document-level) operations that alter instance(s) to reflect a change (e.g., insert, update, delete); and, other types of operations (policy-level) that act on the instance as a whole and not in the intrinsic data found within (e.g. authorizations). As a result, our work provides a security framework and an associated security model that supports all of the aforementioned access control capabilities for large-scale information applications in domains such as healthcare, e-commerce, national defense, etc. The intent of this dissertation is to define a security for framework for tree-structured information modeling formats that use schemas to concretely specify structure that are enforceable via generated security policies at the instance level to deliver only the information required and nothing more to the end user of an information application across LBAC, RBAC, and DAC access control models. The end result will include a secure information engineering process and the generation of security policies to enforce the define security against an application's schemas (design time) and instances (run time).

1.1 Motivation for the Healthcare Domain

The healthcare domain, where patient information is modeled and utilized by a wide range of stakeholders is a prime example of a large-scale information application that heavily relies on tree-structured information modeling for the operation and exchange of medical data. In the healthcare domain, XML has emerged as the de-facto standard of information exchange. XML provides an extensible tree-structure for information design via documents that is machine readable and easy to process, combined with the ability to design and develop schemas for document instance validation. The existence of XML standards (e.g. HL7 CDA, CCR, etc.) in healthcare for representations and their use by numerous health information technology (HIT) systems dictate the need for a new layer of functionality: the ability to secure and enforce access control of an information application that is designed and constructed to use data from constituent local systems in a more global manner. In the United States, federal laws like the Health Insurance Portability and Accountability Act (Annas, 2003; Baumer, Earp, & Payton, 2006) of 1996 provide guidance on not only who can access information, but on how it is to be transmitted, disclosed and distributed; and countries throughout the world have similar laws. Another example legislation of the United States healthcare system, the Health Information Technology for Economic and Clinical Health (HITECH act enforcement interim final rule.2014) Act of 2009, which aims to promote the adoption and meaningful use of health information technology while addressing the privacy and security concerns of electronic transmission of private patient information and health information exchange (HIE); and again, there are international efforts and ongoing efforts on electronic usage of patient data/HIE. Providing a solution for security in scenarios like these needs to

consider all the elements in play (legal constructs, system's requirements, capabilities, technologies, etc.) and must be able to integrate the essential components in a sensible way. In support of this, as shown in the right hand side of Figure 1.2, the proposed security framework is comprised of multiple access control mechanisms, the algorithms and infrastructure for permission definition that are used for policy generation. The bottom of the figure shows the generated policies that are filtered for delivery to the end user.

To illustrate, consider a healthcare information application – a system of systems - built as a combination of constituent HIT systems (right hand side of Figure 1.2), where each one can have their own local security policies. The example is focused on a healthcare information application composed of: two electronic health records (EHRs), OpenEMR (Sainz de Abajo & Ballestero, 2012) and Practice Fusion (Practice fusion.2014)), utilized by a hospital, clinic or medical provider; a personal health record (PHR) Microsoft HealthVault (Microsoft HealthVault.2014) for patients to manage their own health information; and, a patient portal (PP) that provides a means for appointments, referrals, and interactions with a medical provider. The bottom right shows two information applications: PHA and SMARTSYNC. Personal Health Assistant (PHA) 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. SMARTSync for medication reconciliation (Ziminski, De la Rosa Algarín, A., Saripalle, Demurjian, & Jackson, 2012), takes patient medications from

MSHV and the Harvard SMART Platform Electronic Medical Record (EMR) and from this information is able to generate a summary list of medications/supplements added by patients (in MSHV) with those prescribed by a patient's medical provider to generate a color-coded list of potential overmedication, adverse interactions, and adverse reactions for the patient and provider. Both PHA and SMARTSYNC need to utilize local security policies of the constituent HIT systems (OpenEMR, MSHV, etc.) guided by the security requirements and capabilities (access control models, and/or permissions) of those constituent systems in order to construct a global security policy that integrates and their security capability/model/policy (left side of Figure 1.2).

For the purposes of this dissertation, the healthcare application as shown in the left hand side of Figure 1.2 will support multiple access control models (LBAC, RBAC, and DAC), which in healthcare would support the definition of security policies on an tree-structured documents (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 LBAC 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 healthcare, the use of an XML schema for an information application requires a security framework for that tree-

structured schema that allows the design, implementation, and deployment of an enforceable security policy to allow access to instances to be precisely controlled by role, security level, or delegation authority. The definition of security at the schema level separates the security from the instances, which avoids the overhead required when updating security policies that would otherwise be embedded in the instances themselves. Figure 1.2 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 tree-structured format such as XML in order to define the local security policy and allow the information to be shared, as shown by the DOC-C yellow diamond in Figure 1.2. As a result, for the research in this dissertation, we generalize from a security framework with a tree-structure information model with varied access control (LBAC, RBAC, and DAC) that is able to generate security policies for enforcement to a detailed example of the specialization of our approach to where XML is the information format.

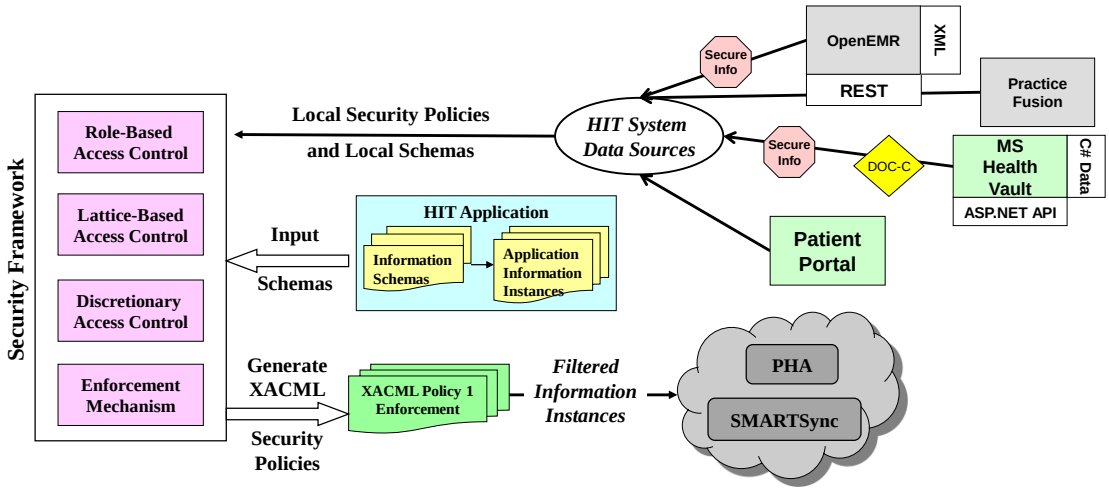

Figure 1.2: Interplay of Information in the Healthcare Domain between Computational Systems.

Security for healthcare 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) (Bilykh, Jahnke, McCallum, & Price, 2006) 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 healthcare standards (CDA, CCR, LOINC, SNOMED, UMLS). Instead, to attain security for healthcare, we will need all of these underlying technologies and standards that must be coupled with a strong understanding of the way that healthcare data is utilized by a wide range of stakeholders, including patients, providers, researchers, etc. In HIT systems, 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 shown in Figure 1.2.

As CDA and CCR 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.

The emerging trends in healthcare 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) (Reid et al., 2009; Zajac, Norris, & Keenum, 2014) 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. Second, accountable care organizations (ACOs) (Accountable care organizations.2011) 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) (Safran et al., 2007) of clinical data that allows both providers and researchers to analyze specific diseases and their treatments across a large patient base via a clinical research data warehouses (CRDW), seeking events such adverse drug reactions, infection monitoring, disease monitoring in a larger population. Fourth, personalized medicine (PM) (Hamburg & Collins, 2010) 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. In support of all of these initiatives, secure information engineering for information applications as shown in Figure 1.2 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 PCMH, ACO, SU, and PM support this infrastructure from three viewpoints: the reconciliation of security (both local and global) within the environment to insure that the required clinical data reaches the medical providers 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 healthcare 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.

1.2 Problem Statement

The security framework in this dissertation is aimed towards a comprehensive approach to modeling the security of an information application in support of efforts such as PCMU, etc., 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 data across these constituent HIT systems, and differences on sharing and security policies across them. Towards this purpose, we advance the information security problem to a software engineering perspective, elevating information security to a first-class citizen of the software design and development process. This *secure information engineering* is accomplished by leveraging and extending the Unified Modeling Language (Fowler, 2004) and the Meta-Object Facility (Poernomo, 2006) M2 layer with new constructs. With these new constructs, the security framework for secure information engineering and enforcement can provide guidance and structure for secure information usage and exchange. By tackling the problem from a perspective of tree-structured documents, any data format that is represented by such a structure (e.g. XML, specialized JSON structures, RDF, OWL, etc.) can be secured. This effectively allows us to provide separation of concerns with respect to information security by defining security requirements in one software process phase and generating enforcement policies in another phase. These enforcement policies are not embedded in the system, on the contraire, they are agents evaluated and enforced in the overall security architecture of the application.

In this dissertation, the initial high-level view of our information security framework (De la Rosa Algarín, A., Demurjian, Berhe, & Pavlich-Mariscal, 2012; De la Rosa

Algarín, A., Ziminski, Demurjian, Kuykendall, & Rivera Sánchez, 2013; De la Rosa Algarín, A., Ziminski, Demurjian, Rivera Sanchez, & Kuykendall, 2013; De la Rosa Algarín, A., Demurjian, Ziminski, Rivera Sanchez, & Kuykendall, 2013) for information usage, sharing, and exchange can be defined as given in Figure 1.3. 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 (Godik, Anderson, Parducci, Humenn, & Vajjhala, 2002) policies (as shown by the Information Security Extensions to UML box). Figure 1.3 illustrate the overall proposed information security framework, which starts with the ability for a designer to pick and choose to create local XACML security policies of constituent information systems (e.g. $Schema_{CIS1}$, $Schema_{CIS1}$, $Schema_{CIS3}$, $Schema_{CIS4}$) and policies for meta-systems (e.g. $Schema_{LSIA1}$ and $Schema_{LSIA2}$) for applications (in HIT, these would be EHRs, PHRs, PPs, etc.). The resulting information security framework achieves granular security by taking the generated XACML policies and enforcing them on the targeted schemas and instances via a combination of schema modeling, policy definition and policy generation, as shown in the second horizontal box in Figure 1.3. These policies support the three major access control models (LBAC, RBAC and DAC), as shown in the third horizontal box in Figure 1.3, and provide a custom or filtered view based on role, user clearance against data classification, and delegated authority.

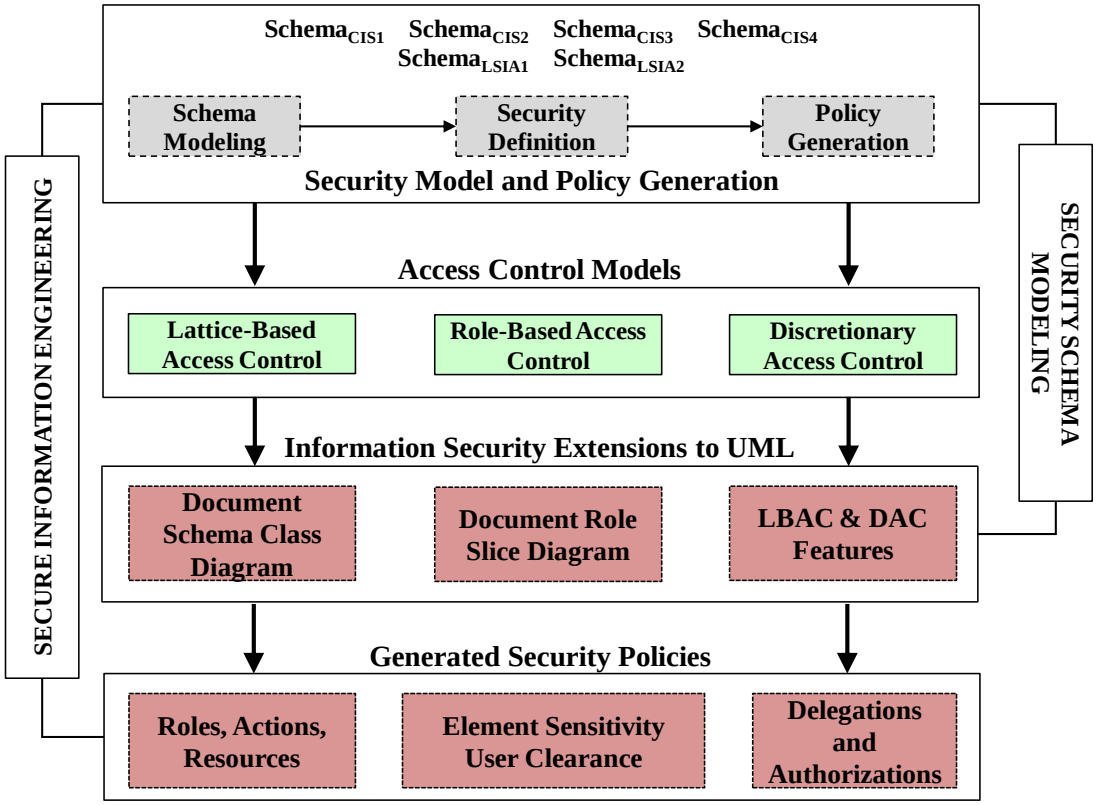

Figure 1.3: Information Security framework in a Healthcare Scenario.

Our approach provides separation of security concerns to tackle the challenge of changing security policies that can apply to multiple instances. This framework is built by leveraging 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 work, we adopt these ideas but apply them in order to create new UML diagrams that can capture tree-structured schemas, roles, sensitivity levels, delegation rules and authorization rules that allow privileges on an application's schemas (see the fourth horizontal box in Figure 1.3) to be allowed and/or denied to different users different times. All of these applications require that the document that is delivered to a user be restricted by role (a filtering of the content of the instance), by security level

(sensitivity 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 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 1.3, are generated security policy models for the new systems.

By achieving this security framework for tree-structured schemas and instances, leveraging three major components (UML, XACML and access control models: LBAC, RBAC and DAC), we present a model-driven approach to document-level security and policy generation. 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 information exchange, the document-level security framework for tree-structured documents produces local security policies that act on the system being engineered and developed. Utilizing the extensions to UML with respect to LBAC, RBAC and DAC, we can also integrate these local policies into an all-encompassing meta-policy that can be utilized by a newly developed meta-system (system of systems).

1.3 Research Objectives and Expected Contributions

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

A. **Security Model and Access Control Integration**: The contribution of this aspect of our work is to provide a security model that leverages the RBAC, LBAC and DAC models with support for their capabilities and features, facilitating the extension of modeling components (such as UML diagrams) to realize granular information security across different access control concepts. Towards this objective, we have defined a proper security model that generalizes away from XML and moves towards the security of tree-structured documents. In this contribution, we assert that RBAC and LBAC features are orthogonal (non-conflicting).

B. **UML Extensions for Tree-structured Document Security**: The contribution of this aspect of our work is to represent a tree-structured schema as an UML-like diagram which is augmented with security features that capture the core aspects of the three main access control models: LBAC, RBAC, and DAC. To date, we have developed two new UML diagrams: the Document Schema Class Diagram (DSCD), which is a generalization of the XML Schema Class Diagram (XSCD), to represent a tree-structured schema as an UML-like diagram; and, the Document Role-Slice Diagram (DRSD), a generalization of the XML Role Slice Diagram (XRSD), to represent RBAC security. Representing the tree-structured 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.

C. **Security Policy Generation**: The intent of this objective is to leverage XML and UML for an automatic policy Creation. This is achieved via a mapping from the new UML security diagrams for LBAC, RBAC and DAC, and the Oasis XACML (Godik et al., 2002) Policy structure, which can be used as part of an enforcement mechanism to assure security. Towards this contribution, we have also introduced a mapping algorithm that considers LBAC, RBAC and DAC security components and integrates them into a single enforcement policy for a required application.

D. **Secure Information Engineering**: This contribution provides a secure information engineering process to systems information by focusing on the perspective of the information to be secured. As shown in Figure 1.3, this secure information engineering process involves the security model by modeling the tree-structured schemas to be secured with the respective access control models (Contribution A), the use of UML diagram and metamodel extensions that realize the security model (Contribution B), and the generation of locally acting security policies (Contribution C) into enforcement policies for a newly developed or existing 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 in their respective application.

1.4 Research Progress to Date

In support of the proposed information security framework, a number of articles have been published or in the process of publication (accepted or in the printing stage). In the first paper, a case for security in digital healthcare is made by considering the current state of affairs in the United States.

- Demurjian, S., De la Rosa Algarín, A., Bi, J., Berhe, S., Agresta, T., Wang, W., Blechner, M. (2013). A Viewpoint of Security for Digital Health Care: What's There? What Works? What's Needed? *(Accepted) To appear in International Journal of Privacy and Health Information Management.*

In the second paper, an approach for functional, collaborative and information security leveraging UML is presented.

- Pavlich-Mariscal, J. A., Berhe, S., De la Rosa Algarín, A. and Demurjian, S. A. (2013). An Integrated Secure Software Engineering Approach for Functional, Collaborative, and Information Concerns. *(Accepted) To appear in State-of-the-Art Concepts and Future Directions in Software Engineering, IGI Global.*

In the third and fourth papers, a methodology to generate XACML enforcement policies from a UML diagram that models role-based access control is discussed.

- De la Rosa Algarín, A., Ziminski, T. B., Demurjian, S. A., Rivera Sánchez, Y. K. and Kuykendall, R. (2013). Generating XACML Enforcement Policies for Role-Based Access Control of XML Documents. *(WEBIST 2013 Selected Papers) (Accepted) To appear in Lecture Notes in Business Information Processing (LNBIP), Springer-Verlag.*

- De la Rosa Algarín, A., Ziminski, T. B., Demurjian, S. A., Kuykendall, R. and Rivera Sánchez, Y. (2013). Defining and Enforcing XACML Role-Based Security Policies within an XML Security Framework. *Proceedings of 9th International Conference on Web Information Systems and Technologies (WEBIST 2013)* (pp. 16-25), doi:10.5220/0004366200160025

In the fifth paper, UML extensions for integrated RBAC are presented. The work in this paper allows security engineers to integrate heterogeneous system's RBAC security into an all-encompassing policy that can be utilized by a newly developed meta-system.

- De la Rosa Algarín, A. and Demurjian, S. A. (2013). An Approach to Facilitate Security Assurance for Information Sharing and Exchange in Big Data Applications. *Emerging Trends in Information and Communication Technologies Security*, pp. 65-83. Elsevier (Kaufman). Editors: Babak Akhgar and Hamid R. Arabnia.

In the last two papers an overview of the general approach of this dissertation is given, specifically focusing on extending UML with RBAC security constructs.

- De la Rosa Algarín, A., Demurjian, S. A., Ziminski, T. B., Rivera Sánchez, Y. K. and Kuykendall, R. (2013). Securing XML with Role-Based Access Control: Case Study in Health Care. *Architectures and Protocols for Secure Information Technology (APSIT)*, pp. 334-365, IGI Global. Editors: Antonio Ruiz Martínez, Fernando Pereñíguez García, and Rafael Marín López.
- De la Rosa Algarín, A., Demurjian, S. A., Berhe, S. and Pavlich-Mariscal, J. (2012). A Security Framework for XML Schemas and Documents for Healthcare.

Proceedings of 2012 International Workshop on Biomedical and Health Informatics (BHI 2012) (pp. 782-789), doi:10.1109/BIBMW.2012.6470239

1.5 Organization of Dissertation

The remainder of this dissertation has six chapters. Chapter 2 provides the motivation and examples from the healthcare domain will be utilized throughout the discussion, as well as background information on the major concepts of the work, including UML, RBAC, LBAC, DAC and XACML. Chapter 3 presents the underlying security model for RBAC, LBAC and DAC of tree-structured documents by discussing each access control model support independently, and then demonstrating an example that utilizes all three. Chapter 4 discusses the portion of the security framework that involves extensions to UML with seven new information diagrams to model RBAC, LBAC and DAC capabilities. More specifically, it presents the *Document Schema Class Diagram*, which serves to model the tree-structured schema as a UML diagram; the *Secure Information Diagram*, which presents a subset of the schema to be secured; the *LBAC Secure Information Diagram*, which supports LBAC; the *Document Role Slice Diagram*, which serves as a generalized version of the XML Role Slice Diagram; the *User Diagram*, which considers LBAC features from the user perspective; the *Authorization Diagram*, which supports schema and instance authorizations; and, the *Delegation Diagram*, which supports DAC. Chapter 5 details the security policy generation process by leveraging the UML extensions from Chapter 4 and XACML, demonstrating the different mapping processes with respect to the supported access control models. Chapter 6 demonstrates the *secure information engineering process* and the prototype of the security model and framework by utilizing a mobile application targeted for healthcare and XML as the tree-

structured document format to be secured. Last, Chapter 7 offers concluding remarks and ongoing/future research areas created by the work of this dissertation.

Chapter 2
Background

This section reviews background concepts for this dissertation in a wide range of areas to support the detailed material in the remaining chapters. First, Section 2.1 briefly reviews the eXtensible Markup Language (XML) (Bray et al., 1998) and its usage in the two dominant healthcare standards: Health Level 7 Clinical Document Architecture (HL7 CDA) (Alschuler et al., 2002) and the Continuity of Care Record (CCR) (Kibbe et al., 2004). Next, Sections 2.2, 2.3 and 2.4 present the various access control models that are utilized in support of the work in this dissertation, respectively: lattice-based access control (LBAC) (Sandhu, 1993) that assigns sensitivity levels to subjects (clearances) and objects (classifications) to control access to information; role-based access control (RBAC) (Ferraiolo et al., 2001) that focuses on the responsibilities of the users for an application per role; and discretionary access control (DAC) (Sandhu & Samarati, 1994) to allow both authority and privileges to be passed from one user to another. Then, Section 2.5 presents a set of assumptions for the healthcare domain, more specifically, detailing the way that health information technology (HIT) systems leverage private and protected information in interacting with patients for care and treatment. Next, Section 2.6 presents a scenario of information usage in the healthcare domain using the two dominant standards (HL7 CDA and CCR) with an emphasis on the way to secure the information across differing degrees of granularity and requirements. Using this as a basis, Section 2.7 introduces the Unified Modeling Language (UML) (Fowler, 2004) and its metamodeling capabilities that are going to be utilized throughout this dissertation to define the new UML diagrams to support the modeling of XML and LBAC/RBAC/DAC.

Lastly, Section 2.8 introduces the eXtensible Access Control Markup Language (XACML) (Godik et al., 2002), a language utilized to create formal and enforceable security policies for XML documents.

2.1 The eXtensible Markup Language (XML)

The eXtensible Markup Language (XML) (Bray et al., 1998) facilitates the modeling and exchange of information between disciplines, offering a flexible means to collect and transmit data between interacting information systems and platforms. XML provides a common, structured language that is independent of the system that utilizes it, and supports information to be hierarchically structured and tagged, with the tags offering the capabilities to represent the semantics of the information. XML supports the definition of an *XML schema* that defines data (elements) to be modeled accompanied by their interdependencies; this is akin to a class diagram or entity in a database design. By allowing the definition of XML schemas (see Figure 2.1) provides the mechanism to both define a standard for the information that can be used both as a blueprint for the content of an instance (which has to obey the schema) and as a means to validate instances that are being exchanged in or to insure that they seeking to comply with the required format. Towards this end, the main mechanism behind XML schemas is the XML Schema Definition (XSD), which follows the XML Schema language.

Figure 2.1 shows an example of an XML schema that could be used in a healthcare scenario. This example, which represents a generic patient, shows the tree-structure of XML documents. In Figure 2.1, the node designated by <xsd:element name=”patient”> acts as the root node of the tree. The direct child of that node is an element designated with the <xsd:complexType> tag, which is in turn followed by a sequence of nodes that

can contain different attributes. For the purposes of this dissertation, we consider non-leaf nodes as those that provide *context*. In turn, leaf nodes provide *value* to their parental context. As one would expect in a healthcare example, information stored on patients include the name (first last), birthdate, vital statistics (height, weight, gender), along with diagnostic data (blood pressure, pulse). Once an XML schema such as patient has been defined, it is then possible to create instances of the schema, representing the actual documents that are being stored and exchanged by an application. Figure 2.2 shows two examples of an XML instance for the schema presented in Figure 2.1, providing both a male and a female patient. The instance on the left hand side of Figure 2.2 is for a female patient called Carol Smith (denoted by the <patient>) contains the name, height in meters, weight in kilograms, etc. Similarly, the instance on the right hand side of Figure 2.2 is for Carol Smith's husband, Joseph Smith, likewise with data for name, height, etc. Note that schemas and instances in XML follow a tree structure; and while the work in this dissertation assumes such a tree structure will be applied to XML, it can also be generalized to other document formats such as JSON (Crockford, 2006), OWL (McGuinness & Van Harmelen, 2004), etc.

```
<xsd:element name="patient">
 <xsd:complexType name="patientInfo">
 <xsd:element name=”firstName”/>
 <xsd:element name=”lastName”/>
 <xsd:element name=”birthdate”/>
 <xsd:element name=”height”/>
 <xsd:element name=”weight”/>
 <xsd:element name=”gender”/>
 <xsd:element name=”bloodType”/>
 <xsd:element name=”systolic”/>
 <xsd:element name=”diastolic”/>
 <xsd:element name=”pulse”/>
 </xsd:complexType>
</xsd:element>
```

Figure 2.1: Generic Schema Segment for a "patient".

```
<patient>                                   <patient>
 <patientInfo>                               <patientInfo>
 <firstName>Carol</firstName>                <firstName>Joseph</firstName>
 <lastName>Smith</lastName>                  <lastName>Smith</lastName>
 <birthDate>                                 <birthDate>
            1983-04-13T00:00:00Z                        1990-05-26T00:00:00Z
     </birthDate>                                    </birthDate>
 <height>1.70</height>                       <height>2.1</height>
 <weight>40</weight>                         <weight>80</weight>
 <gender>F</gender>                          <gender>M</gender>
 <bloodType>O</bloodType>                    <bloodType>A</bloodType>
 <systolic>110</systolic>                    <systolic>95</systolic>
 <diastolic>70</diastolic>                   <diastolic>50</diastolic>
 <pulse>85</pulse>                           <pulse>75</pulse>
 </patientInfo>                              </patientInfo>
</patient>                                  </patient>
```

Figure 2.2: Instantiated "patient" Segment**.**

2.2 Lattice-Based Access Control (LBAC)

The lattice-based access control model (LBAC) (Sandhu, 1993) and the mandatory access control model (Bell & La Padula, 1976) were simultaneously proposed in 1976. Both LBAC and MAC share the approach that there are security sensitivity levels that are assigned to subjects (clearance) and objects (classification) with the permissions for the subject to read and/or write an object dependent on the relationship between clearance and classifications. MAC typically is modeling using four sensitivity levels which are hierarchically ordered from most to least secure: top secret (TS) < secret (S) < classified (C) < unclassified (U)). LBAC generalizes this approach by ordering the sensitivity levels in a lattice that determines the relative ranking of each sensitivity level vs the others. Security policies in LBAC and MAC are using defined by a security administrator with the intent to control information flow in computer systems where users prohibited from changing their security attributes. In LBAC and MAC, 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 both LBAC (IBM Informix (IBM-informix database software.2014) and DB2 LUW

(Rjaibi, 2006)) and MAC (FreeBSD as part of the TrustedBSD, (Dinh-Trong & Bieman, 2005), Windows with Windows Vista (Conover, 2006), Trusted Solaris (Faden, 2006), and the NSA SELinux research project (McCarty, 2004)).

In both LBAC and MAC, security features can be selected to govern the read and write permissions of users with certain clearance levels and elements with certain classification levels to dictate the way that information at one level is allowed to be read or written at another level. In fact, as outlined in (DoD trusted computer system evaluation criteria.2014):

> "A subject can read an object only if the hierarchical classification in the subject's security level is greater than or equal to the hierarchical classification in the object's security level and the non-hierarchical categories in the subject's security level include all the non- hierarchical categories in the object's security level. A subject can write an object only if the hierarchical classification in the subject's security level is less than or equal to the hierarchical classification in the object's security level and all the non-hierarchical categories in the subject's security level are included in the non-hierarchical categories in the object's security level. "

Today, these rules have been realized as a set of security properties. *Simple security (SS),* or read-down, no read-up, is the permission to read at equal or lower levels. That is, a user is allowed to read elements with a sensitivity level lower than their clearance level, but not those elements with a higher sensitivity level. *Simple integrity (SI),* or write-down, no write-up, is the permission to write to equal or lower levels. That is, a user can write elements with a lower sensitivity level when compared to their clearance level, but not to those elements with a higher sensitivity. *Liberal star (LS),* or write-up, no write-

down, is the permission to write to equal or greater levels (the opposite of SI). Finally, *Strict Star Write (SSW*) and *Strict Star Read (SSR),* or write (read) equal, is the permission to write (read) only to equal levels. From a definition and management perspective, a security administrator would set the clearance level of users following the predefined sensitivity levels (e.g., TS, S, C, and U) to establish the levels for both subjects and objects. These levels are then augmented on a user-by-user basis by assigning the ability to read (via SS or SSR) and the ability to write (via SI, LS, or SSW). Once this has all been established for an application it then supports the definition of permissions and levels (e.g., the elements of a patient's health record) in order to maintain confidentiality by preventing an unauthorized provider to access sensitive information (e.g., not all providers are able to access mental health history) and to prohibit a patient from changing their own record.

2.3 Role-based Access Control (RBAC)

In the role-based access control model (RBAC) (Ferraiolo et al., 2001), the intent is to allow the responsibilities of the role within the application to dictate the permissions for a given user. Roles are defined in an application in order to name entities that are often related to job categories and responsibilities that can be extrapolated from individual users and established as a reusable role. For example, in a healthcare application, a role for Nurse or Physician could be defined to capture all of the capabilities in terms of access and modification of information that are available for a user assigned a given role. Unlike LBAC, in RBAC, the users as subjects are assigned roles with the roles then assigned permissions as shown in Figure 2.3. In that way, the role(s) that a user are assigned to can vary based on what the user is doing in a given application at a specific

time. For example, a nurse Sara might have a NurseManager role to be able to manage the unit (all other nurses, staff, and patients) that they are supervising in a hospital and have a NurseCare role to be able to manage the information associated with her assigned patients. Sara can plan either NurseManager or the NurseCare role at any one point in time.

The standardized version of RBAC, defined by the National Institute for Standards (NIST) (Ferraiolo et al., 2001) contains four reference models, as shown in Figure 2.3. $RBAC_0$ allows for policies to be denied at the role level instead of the individual level. This is akin to the Nurse or Physician role above. To handle role hierarchies, $RBAC_1$, shown in the upper middle portion of Figure 2.3, allows for roles to be organized in a hierarchical fashion and as a result, 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. In the example above, Nurse could be a parent role of both NurseManager and NurseCare and contain the privileges that would be shared by both roles. This allows the definition of roles to be more clearly organized into a hierarchy with privileges defined in one location passing down the tree to other roles and sub-roles.

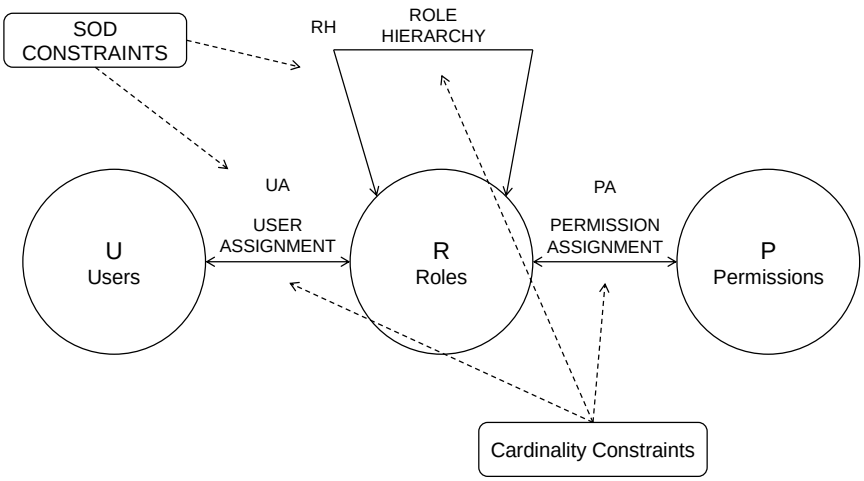

Figure 2.3: NIST $RBAC_0$, $RBAC_1$, and $RBAC_2$.

The next level of RBAC, $RBAC_2$, shown in the upper left of Figure 2.3, provides for the definition of constraints, such as separation of duty (SoD), mutual exclusion (ME), and cardinality. Separation of duty mandates that in order to complete a particular task, a number of individuals must participate, and, as a result, have a separation in their actual duties to accomplish a given task. For example, a patient arriving at an emergency room with heart symptoms is triaged and evaluated by a nurse, the ER physician, and a cardiologist; all have users have their separate roles and their specific duties to accomplish to treat the patient. Each will have the ability to write their own portion of the patient record (e.g., Nurse for history, ER physician for ordering tests, and cardiologist for evaluating an EKG) involving the need to all read the information in real-time, but may be prohibited from seeing the information (the nurse can't see the EKG results) until such time as allowed another use (cardiologist). 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.

2.4 Discretionary Access Control (DAC)

In the discretionary access control model (DAC) (Sandhu & Samarati, 1994), access to objects is permitted or denied based on the subject's identity or the group they are part of with the ability to have privileges pass among users either under the control of a user or as dictated by a security administrator. DAC utilizes the concept of delegation to be able to pass privileges among users in certain situations and under specific constraints. A user has both the authority to delegate his/her permissions to another user as well as to authority to pass on the ability to delegate to another user who can then delegate to a third sure. There are two types of delegation authority in DAC. The first, administrative delegation, has a security office control who to delegate to whom and when to delegate. This would be useful in a university application where the head of an academic department may delegate his authority to another faculty member in his/her absence in order to run the department. The second, user-directed delegation puts the capability to both authorize and delegate to the user. In the above healthcare application, a physician John would delegate his authority to another physician Tom who is covering John's patients for the evening as the on-call physician. In both of these cases, the delegation would essentially transfer the permissions (either LBAC or RBAC) and the involved objects from one user to another. So John would specifically transfer his OnCallMD role to Tom along with access to his patients.

There are different ways to implement DAC under the assumptions of either LBAC or RBAC. In one approach, the permissions and the objects they apply to can be delegated. That is, every object would contain an owner that controls the permissions on that object, and would as a result control which users can access the object, and which permissions are allowed. This may not be manageable in many applications since it assumes that the user plays a significant role in both the definition and management of permissions, objects, and delegations. A second approach would permit users to delegate (transfer) their permissions (access) that target certain objects with other users. In this situation, it makes sense for a physician to pass on his/her OnCallMD role along with all of his/her patients to another physician; this leverages RBAC concepts and allows a user to pass on a role. In an LBAC setting, authorizations would be established to allow an individual with a clearance to have access to objects and their elements as constrained by classifications on the complete objects. This security capability can be utilized as an extra layer of security (only those authorized objects are the ones a user can perform any operations on), or as a fallback capability (delegating authorizations when a primary care provider is not available and someone with equal credentials is).

2.5 Healthcare Domain Assumptions

The work of this dissertation involves several assumptions with regards to security and the healthcare domain in HIT applications. The first assumption involves the idea that there are a set of HIT applications that are relevant in the healthcare domain. These include: electronic health records (EHRs), which are typically used in provider institutions such as hospitals, clinics and private practices, and serve the purpose of storing patient health information for continuous care; personal health records (PHRs),

which are provided by several institutions (e.g. Microsoft HealthVault (Microsoft HealthVault.2014), WebMD (WebMD.2014), etc.) for patients to use in order to maintain their own record of health information that can potentially contain information not found in EHRs; patient portals (PPs), which act as a door to an EHR from the perspective of the patient and have functionalities such as medical appointments managing, prescription repositories, etc.; and, a wide range of ancillary systems for pharmacy, diagnostic laboratories, therapy centers, and so on. These applications are candidates for the usage of our security framework presented in this dissertation to allow for the information that must be exchanged to be appropriately modeled.

The second assumption considers the conditions under which information is modeled. Specifically, we assume that HIT applications model information utilizing a tree structure of index and entity nodes that allows for information to be modeled via schemas (that define structure) which can be used as blueprints for the creation of new documents (instances) and their validation (enforcement). In such settings, where sensitive data is utilized by users for time-critical applications, security that is achieved via access control is a paramount concern. Once such example of a tree-structured document is in XML, as was discussed in Section 2.1, including XML's usage in standards such as HL7 CDA and CCR.

Our third assumption is that a myriad of users with potentially different roles (RBAC) and clearance levels (LBAC) are interested in accessing the information to be secured from the set of HIT systems. There is a need to bring together all of the information found in heterogeneous HIT applications (e.g. EHRs, PHRs, PP, etc., in the first assumption) in a way that appropriate and required access control across the integrated

set of HITs is achieved. This dissertation thrives under the assumption that different information systems exist, some of them possibly built as large-scale information systems (LSIS, or systems-of-systems) that are comprised of EHRs, PPs, etc. as given in the first assumption, and that information must be secured for a new health information application that will be used by multiple users under multiple roles and multiple clearance levels. Using this as a basis, we list a number of high-level assumptions regarding that information's structure, users, roles and general access control features:

1. Security administrators will determine the security requirements of a new information system LSIS by considering a variety of access control models, namely: LBAC, RBAC and DAC.
2. LBAC and RBAC are orthogonal. That is, features from either of the access control models does not affect the other. This allows for the definition of security for a new information system that supports LBAC and not RBAC, RBAC and not LBAC, or LBAC and RBAC.
3. With the purpose of facilitating modeling, security administrators will be able to graphically design and model the major concepts of an information system. This includes the users, roles, lattice sensitivity levels, and authorizations.
4. All instantiated patient information to be secured is found in a tree-structured format that contains a schema that easily validates its structure.
5. Access to the patient information is allowed through the secured information system.
6. Generated external policies are used to enforce the defined security requirements over the information system.

7. Mobile applications geared towards health information management are used to motivate and demonstrate the work presented herein.

These assumptions set the context for the dissertation in terms of both the security framework to be presented as well as supporting the examples throughout.

2.6 Healthcare Scenario

This section presents a plausible scenario of health information modeling and the needed security using current and standardized technologies in healthcare. The scenario explained in this section will be utilized to illustrate both the modeling and the implementation of security for information systems in support of the proposed security framework. While the work presented in this dissertation is domain independent, most of its capabilities are driven by healthcare requirements, and the majority of HIT applications utilize XML in modeling or to be able export and import data sets (a patient or a set of patients) for information exchange. The scenario described herein covers two of the dominant standards in healthcare: Health Level 7 Clinical Document Architecture (CDA) (Alschuler et al., 2002) and the Continuity of Care Record (CCR) (Kibbe et al., 2004). Both utilize XML and follow the tree-structured assumptions in this dissertation.

To begin, in Figure 2.4, a segment of the HL7 CDA schema is shown. CDA is capable of modeling different aspects of a health provider transaction, ranging from a patient's demographic data to medical diagnoses to medications to billing. In Figure 2.4 for the CDA schema, each xsd:element refers to other XML complex types, allowing for the inclusion into a patient's medical record: the encounter with the physician (patient_encounter element), the organization of the physician (provider element), and the patient him/herself (patient element). Likewise, in Figure 2.5, a CCR schema for a patient

record represents the structure for a patient record to be instantiated for each actual patient. In Figure 2.5, again, there are references to xsd:element that refer to other complex types such as the body element that contains the payer information (insurance type, company details, etc.), the problems element that serves as a list of individual problems, and the family history element that houses specific instances of family history problems specific for the patient.

```
<xsd:element name="clinical_document_header">
 <xsd:complexType>
 <xsd:sequence>
 <xsd:element ref="id"/>
 <xsd:element ref="set_id" minOccurs="0"/>
 <xsd:element ref="version_nbr" minOccurs="0"/>
 <xsd:element ref="document_type_cd"/>
 <xsd:element ref="service_tmr" minOccurs="0"/>
 <xsd:element ref="origination_dttm"/>
 <xsd:element ref="copy_dttm" minOccurs="0"/>
 <xsd:element ref="confidentiality_cd"
 minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="document_relationship"
 minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="fulfills_order" minOccurs="0"/>
 <xsd:element ref="patient_encounter" minOccurs="0"/>
 <xsd:element ref="authenticator" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="legal_authenticator" minOccurs="0"/>
 <xsd:element ref="intended_recipient" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="originator" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="originating_organization" minOccurs="0"/>
 <xsd:element ref="transcriptionist" minOccurs="0"/>
 <xsd:element ref="provider" maxOccurs="unbounded"/>
 <xsd:element ref="service_actor" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="patient"/>
 <xsd:element ref="originating_device" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="service_target" minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element ref="local_header" minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attributeGroup ref="common_atts"/>
 <xsd:attribute name="HL7-NAME" type="xsd:string"
fixed="doc_serv_as_clin_doc_header"/>
 <xsd:attribute name="T" type="xsd:string" fixed="service"/>
 <xsd:attribute name="RIM-VERSION" type="xsd:string" fixed="0.98"/>
 </xsd:complexType>
```

Figure 2.4: HL7 CDA 'clinical_document_header' Schema Segment.

To illustrate both CDA and CCR, Figure 2.6 contains an instance of the CDA schema, while Figure 2.7 contains an instance of the CCR schema. In Figure 2.6, a CDA instance for the patient Carol Smith is shown as a female (GenderCode is F) and seeing Dr. Brock Ketchum who is a general physician of the assignedEntity (the physician's office) and has

written a note on Carol as part of her medical history. As shown in the note of the CDA in Figure 2.6, Carol was hospitalized twice this year and last year, and has not been able to reduce dependency on steroids for the last month of treatment recorded. She was assigned to Dr. Elisa Fakington for the treatment received in each visit. Likewise, in Figure 2.7, a CCR instance for CAROL SMITH contains identified health problems (Asthma) as well as medications (Zithromax).

Given this brief introduction and examples for CDA and CCR, the intent of the security framework in this dissertation is to secure a patient's data as it exists in the CDA/CCR documents that are instantiated and validated to exchange that data with different stakeholders (other medical providers) and the patient him/herself. To set the stage for this, we consider the creation of a new information system that utilizes both LBAC and RBAC in conjunction with DAC as presented in Sections 2.2, 2.3, and 2.4. The scenario starts with a Dr. Ketchum's private medical practice, called GetBetter Clinic, where there is an interest to have a mobile patient application that allows a provider to access select portions of a patient's medical information in his EMR, allowing the prescription of medications and other activities such as secure email, making appointments, etc. For our purpose, suppose that GetBetter Clinic, commissions the new information system, a mobile patient application, to manage their patient's health data, which may be modeled by either CCR or HL7 CDA schemas with data exchanged back and forth between the patient and the system using XML instances. As her healthcare providers will utilize the provider version of the mobile app, Carol will make extensive use of the patient version in order to maintain continuous contact for her asthma condition. This dissertation will utilize Carol Smith's health record in CDA and CCR

```
<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="Body">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Payers" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Payer" type="InsuranceType" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="AdvanceDirectives" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="AdvanceDirective" type="CCRCodedDataObjectType"
                            maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="Support" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="SupportProvider" type="ActorReferenceType"
                            maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="FunctionalStatus" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Function" type="FunctionType" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="Problems" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Problem" type="ProblemType" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="FamilyHistory" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="FamilyProblemHistory" type="FamilyHistoryType"
                            maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
            …
```

Figure 2.5: CCR – Continuity of Care Record Schema Segment.

structure, as well as the information system (mobile app) commissioned by GetBetter Clinic, to discuss the major contributions.

To control access to the information supplied to/from the provider mobile application, there will be a need to utilize LBAC to define sensitivity levels (clearances for each user and classifications for the data elements) in the CCR/CDA instances to control access to Carol Smith's data. The security administrator of GetBetter Clinic uses the four levels from Section 2.2: *top-secret*, *secret*, *classified* and *unclassified*. Some of the patient data such as mental health records will require a top-secret classification to insure that the information is protected for use by anyone except for the treating physician (available to Dr. Ketchum but not available to Dr. Fakington); this information has to be shared under specific rules. Other patient data such as medications or allergies may be secret; it could be the ability for a physician such as Dr. Ketchum to write or alter a prescription, which would be prohibited from a nurse such as Jenkins. Nurses like Jenkins may have access to classified data that would include the ability to enter a medical history and record vital signs. Office staff could have unclassified access to demographic data in order to contact a patient for appointments or do insurance billing. These specific privileges are augment with extra layer of security which will further defined the actual instances that will be authorized, which in turn are filtered by the privileges.

```
<ClinicalDocument>
…
 <patient>
 <name>
 <given>Carol</given>
 <family>Smith</family>
 <suffix></suffix>
 </name>
 <administrativeGenderCode code="F" codeSystem="2.16.840.1.113883.5.1"/>
 <birthTime value="19830413"/>
 </patient>
 …
 <assignedPerson>
 <name>
 <given>Brock</given>
 <family>Ketchum</family>
 <suffix>MD</suffix>
 </name>
 </assignedPerson>
 …
</author>
<custodian>
 …
</custodian>
<documentationOf>
 …
 <performer typeCode="PRF">
 …
 <assignedEntity>
 …
 <code code="59058001" displayName="General Physician"/>
 …
 <assignedPerson>
 <name>
 <prefix>Dr.</prefix>
 <given>Elisa</given>
 <family>Fakington</family>
 <suffix/>
 </name>
 </assignedPerson>
…
 <component>
 …
 <title>History of Present Illness</title>
 <text>
 <content styleCode="Bold">Carol Smith</content>
 is a 31 year old female referred for further asthma management.
 Onset of asthma in her <content revised="delete">twenties</content>
 <content revised="insert">teens</content>.
 She was hospitalized twice last year, and already twice this year.
 She has not been able to be weaned off steroids for the past several
 months.
 </text>
 </section>
 </component>
 </structuredBody>
</component>
</ClinicalDocument>
```

Figure 2.6: Instantiated part of CDA for Carol's Health Record.

Due to the initial design work that will go into the provider version of the app, GetBetter Clinic has also commissioned a patient version that will serve as a mobile patient portal. The information utilized in the patient version of the app like Carol Smith

will follow CDA and CCR schemas, but be limited to medications, allergies, and conditions; there will also be the ability to securely communicate with Dr. Ketchum, request medication refills, make appointments, etc. The security requirements of the patient version of the app will be similar to that of the provider version. One distinction in security requirements is that medical notes created by physicians, psychiatrists, and other health providers, will not be accessible to the patient; these may be classified so that Nurses like Leroy and all of the GetBetter Clinic physicians can access, but office staff and patients would be prohibited. A patient like Carol Smith that utilizes the app will be authorized to her health record instances only, and their clearance and permissions will be governed by those requirements set by the GetBetter Clinic security administrator. The patient version of the app will not allow for patients to add their own information, as GetBetter Clinic wants it to serve strictly as a window to the user's health information in their EMR. In addition to the main functionality of browsing the permitted segments of their health record, the patient app includes capabilities such as secure emailing for continuous contact with their healthcare providers, the ability to check/request appointments and a module to manage medication refills.

The practice has a set of personnel including: Brock who is a psychiatrist, Elisa who is a family practice physician, Leroy and Jenkins who are nurses, and Gail who is an office staff member. Brock with a clearance level of *top-secret* which allows him to read and write mental health notes. GetBetter Clinic has several employees that are part of the daily workflow. Brock is a provider (psychiatrist) with a clearance level of *top-secret* that allows him to read and write mental health notes. Elisa has a clearance level of *secret* able to read and write all of the patient information (except mental health notes). Leroy

and Jenkins are nurses with clearance levels of *classified* who are able to see most portions of the patient record (except mental notes) and able to perform limited edits and additions. These privileges are defined by the security administrator in consultation with the personnel of GetBetter Clinic. Overall, most can Physicians can read the complete patient record, but may be limited in writing certain parts. Nurses can also read the complete patient record, and have limited write access. Parts of the patient's health record will be tagged with sensitivity levels so that it is easy to understand which sections are top-secret, secret, classified and unclassified. At the same time, users will be assigned specialized permissions for reading and writing. Users will be able to read-down their clearance levels, and will also be able to write-up their clearance levels.

The information system (mobile app) commissioned by GetBetter Clinic needs to support the major access control models and needs to be developed with potentially changing security policies (i.e., new laws can affect existing guidelines). The mobile app will also needs to support two different viewpoints, one from the provider perspective in which health information is secured with respect to roles and clearance levels; and, one from the patient perspective in which health information is authorized on an instance basis and certain parts of the health record are completely disallowed regardless of ownership.

```
<ContinuityOfCareRecord>
    …
    <Purpose>
    <Description>
    <Text>Summary of patient information</Text>
    </Description>
    </Purpose>
    <Body>
    <Problems>
    <Problem>
                …
    <Description>
    <Text>Asthma</Text>
    </Description>
    <Status>
    <Text>Active</Text>
    </Status>
                …
    </Problem>
    </Problems>
    <Alerts>
    <Alert>
          …
    </Alert>
    </Alerts>
    <Medications>
    <Medication>
    …
    <Product>
    <ProductName>
    <Text>Zithromax</Text>
    <Code>
    <Value></Value>
    <CodingSystem>RxNorm</CodingSystem>
    </Code>
    </ProductName>
    </Product>
    </Medication>
    </Medications>
         …
    <Actor>
    <ActorObjectID>A1234</ActorObjectID>
    <Person>
    <Name>
    <CurrentName>
    <Given>CAROL</Given>
    <Family>SMITH</Family>
    <Suffix/>
    </CurrentName>
    </Name>
    <DateOfBirth>
    <ExactDateTime>1983-04-13T00:00:00Z</ExactDateTime>
    </DateOfBirth>
    <Gender>
    <Text>Female</Text>
    <Code>
    <Value/>
    </Code>
    </Gender>
    </Person>
         …
  </ContinuityOfCareRecord>
```

Figure 2.7: Part of Carol's Health Record Represented with the CCR Standard.

2.7 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), 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. Finally, 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.

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 2.8, 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 (UML ISO standard.2014) for software systems, several tools (and development environments) exist to aid in UML modeling, including: ArgoUML (Ramirez et al., 2003), StarUML (Lee, Kim, Kim, & Lee, 2005), Eclipse (Moore, Dean, Gerber, Wagenknecht, & Vanderheyden, 2004), Visual Studio (Randolph, Gardner, Anderson, & Minutillo, 2010), NetBeans (Boudreau, Glick, Greene, Spurlin, & Woehr, 2002), and others. The UML meta-model will be utilized in this dissertation to support the definition of new UML diagrams that are capable of modeling LBAC, RBAC, and DAC for tree structured documents and their instances, like as supported in XML.

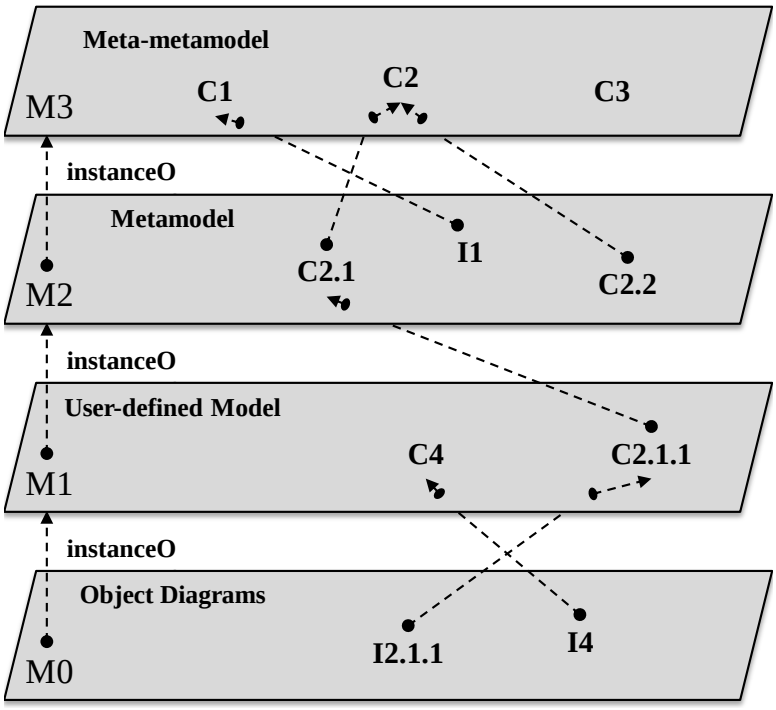

Figure 2.8: UML's Meta-Object Facility Layers.

2.8 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) (Godik et al., 2002) defines 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 2.9) 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*.

The architecture of a typical security system that utilizes XACML for enforcement as given in Figure 2.10 also has a number of components that we can explain and relate to our ongoing example. First, the Policy Enforcement Point (PEP) allows a request to be made on a resource such as Dr. Ketchum playing a Physician role to access a continuity of care record instance of Carol Smith. Next, the Policy Decision Point (PDP), evaluates the request and provides a response according to the policies in place, and in our example, a Physician role played by Dr. Ketchum can access (read and/or write) a portion of a continuity of care record schema. Then, the Policy Administration Point (PAP) is utilized to write and manage policies which in our example would be applied against the CCR schema and its associated instances to deliver the appropriate subset of information to a Nurse or Physician role and the individual who is playing that role. Last, the Policy

Information Point (PIP) can be utilized to arbitrate very fine grained security issues which in our continuing example would be to control access to mental health data of Carol Smith, allowing Ketchum, while denying Fakington In support of enforcement, an XACML PolicySet allows the ability to make the authorization decision via a set of rules in order to allow for access control decisions that may contain multiple Policies, and each Policy contains the access control rules. Note that multiple XACML Polices may be generated, resulting in a PolicySet for a specific set of XML schemas that comprise a given application.

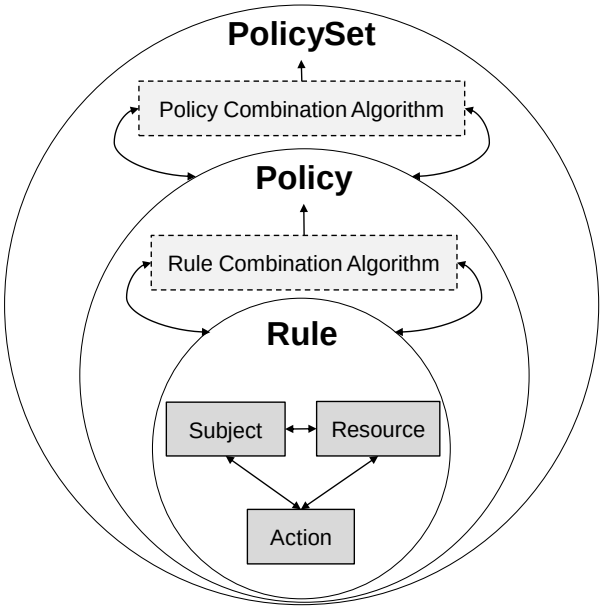

Figure 2.9: Layered Representation of XACML's PolicySet, Policy and Rule Constructs.

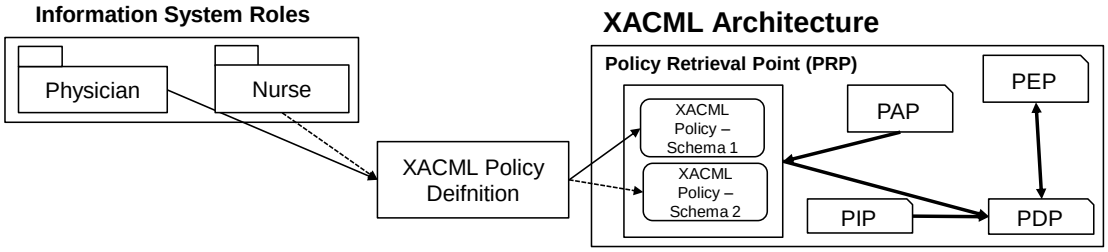

Figure 2.10: Security Architecture that Utilizes XACML Policies for Enforcement.

As an example to the way that a generated XACML policy would be for our scenario, consider Figure 2.11, which illustrates the structure of a basic RBAC policy. In this

policy, the target sets the subject to the user Elisa Fakington (<Subject> element and <user> child element), any set of resources at a policy level (<Resources> element and <AnyResources/> child element) and the possible actions (<Actions> element and <AnyResources/> child element). The first rule pertains to the role of Physician (denoted by the <roleName> tag inside the <role> sub-element of <Subjects> in the <Target> subtree) with an identifier 5 (denoted by the <roleID> tag) when considering any resources (<AnyResource/>) and any action (<AnyAction/>). The rule of this policy (<Rule> element) dictates that Elisa Fakington, acting as a Physician, is permitted (Effect="Permit") to insert (<operation> element under <Actions>) the element denoted by the name "Past Medical History" (<element> element under <Resources>). This policy then allows any Elisa Fakington to insert new data in a patient's Past Medical History.

```
<Policy PolicyId="ada-example-policy" RuleCombiningAlgId="deny-overrides">
 <Description>
     This is a pseudocode example of an XACML policy with LBAC, RBAC and DAC
 capabilities for user Elisa and role Physician.
 </Description>
 <Target>
 <Subjects>
      <user><id>6</id><name>Elisa Fakington</name></user>
 </Subjects>
 <Resources>
      <AnyResource/>
 </Resources>
 <Actions>
      <AnyAction/>
 </Actions>
 </Target>

 <Rule RuleId="simple-RBAC+LBAC-rule" Effect="Permit">
 <Target>
 <Subjects>
              <role><roleID>5</roleID><roleName>Physician</roleName></role>
 </Subjects>
 <Resources>
             <element>
              <elementID>el-3</elementID>
              <elementName>Past Medical History</elementName>
             </element>
 </Resources>
 <Actions>
             <operation>
              <operationName>insert</operationName>
              <opAccessMode>write</opAccessMode>
             </operation>
 </Actions>
 </Target>
        <Condition>
        <Apply FunctionId="…:integer-greater-than-or-equal">
        <Apply FunctionId="…:integer-one-and-only">
        <AttributeValue DataType="…#integer">
        Secret
        </AttributeValue>
        </Apply>
        <AttributeValue DataType="…#integer">
        Secret
        </AttributeValue>
        </Apply>
        </Condition>
    </Rule>

 <Rule RuleId="simple-DAC-rule" Effect="Permit">
 <Target>
 <Subjects>
             <user><id>6</id><name>Elisa</name></user>
 </Subjects>
 <Resources>
              <Schemas>
              <schema><schemaID>4</schemaID><schemaName>Schema
4</schemaName></schema>
              </Schemas>
              <Instances>
              <instance>
                    <instanceID>4,2</instaneID>
                    <instanceName>Carol Smith Health Record</instanceName>
 </Resources>
 </Target>
 </Rule>
```

Figure 2.11: XACML Policy for User Elisa Fakington and Role Physician.

Chapter 3
RBAC, LBAC and DAC Security Model for Tree-Structured Documents

The objective of this chapter is to propose a security model for tree-structured documents that includes the ability to define role-based (RBAC) (Ferraiolo et al., 2001), lattice-based (LBAC) (Sandhu, 1993), and discretionary (DAC) (Sandhu & Samarati, 1994) access control for information systems. This model will define the underlying concepts and capabilities that will serve as a foundation for the definition of new UML diagrams to model RBAC, LBAC, and DAC for tree-structured documents, with the intent to achieve fine-grained information security via access control as part of the overall software engineering process for information systems. The inclusion of security as part of an information system's design facilitates the modeling of RBAC, LBAC, and DAC at a schema-level that is then realizable against its instances. The key intent of our approach is for a schema-level security solution that defines and enforces fine-grained control of an information system's instances for: non-destructive (document-level) operations that utilize instances as a source of information (e.g., read and aggregate); destructive (document-level) operations that modify instance(s) to reflect a change (e.g., insert, update, delete); and, other types of operations (policy-level) that act in the instance as a whole and not in the intrinsic data found within.

In order to achieve this for an information system, there is the need to provide all relevant stakeholders with some degree of assurance on the different capabilities of the three access control models that are being supported. First, from a usability perspective, the security model should be able to support any document format that follows a tree-

structure for representation which would allow the design of an information system without regard to the underlying data format. Candidate formats include, but are not restricted to, the eXtensible Markup Language (XML) (Bray et al., 1998), JavaScript Object Notation (JSON) (Crockford, 2006), the Resource Description Framework (RDF) (Klyne et al., 2004), the Web Ontology Language (OWL) (McGuinness & Van Harmelen, 2004), etc. The security model in this dissertation casts these formats in their most generalized tree form, specifically, supporting a document model where each document has a schema and each schema has a single root node and potentially many children nodes. Second, the a majority of the RBAC capabilities in the NIST model (see Section 2.3) are supported through the definition of roles, the assignment of operations as permissions, and the determination of constraints in support of role delegation with mutual exclusion (ME). Third, a wide range of LBAC capabilities will be supported by the security model, allowing a broad range of information systems and their security needs to be supported. This includes, as indicated in Section 2.2, the ability to assign classifications to all application schemas and their elements and define clearances for users. This assignment extends the schema via a decoration operation that acts over a determined criteria and results in an extension schema and its elements (e.g., the entire document tree) with classifications as tags that are then enforced at the instance level for actual users holding the appropriate clearance. When RBAC and LBAC are combined, a given role can access each element in a specific way via a particular operation, achieved by filtering schemas and instances utilizing the role as the criterion. Fourth, the ability to support DAC that includes the delegation of role from user to user and the ability to pass on the delegation. The model completes with a discussion from the user perspective that

includes authorizations over schemas and instances. Finally, for the purposes of this chapter, we note the distinction between document-level security and policy-level security. *Document-level security* includes those concepts that act on the tree-structure (e.g., classification levels for elements, access modes of operations that target the elements, operations and permissions for a role that are tied to an element, etc.), while *policy-level security* pertains to those capabilities that are defined at a higher level (e.g., authorizations over schemas and instances). The main assertion towards the research in this chapter is that RBAC and LBAC are orthogonal. That is, their capabilities do not conflict with one another. This is achieved by assigning clearance levels to users and not roles. Ultimately, when unifying RBAC, LBAC and DAC, LBAC requirements take a higher-level priority and dictate the final security effect; but from a construction perspective, the security can be added in any order with the end result the same in terms of the defined and enforced permissions.

The remainder of this chapter has seven sections to define and explain the security model and compare our work to other efforts. Section 3.1 discusses the general information system structure that will make use of this security model, presenting the assumptions and the form of the major components in terms of schemas, an application, and the users. Next, Section 3.2 presents the general security architecture (as a basis) that the information system will utilize in order to leverage the security model presented in this dissertation. Sections 3.3, 3.4 and 3.5 discuss the RBAC, LBAC and DAC capabilities respectively, taking special consideration to the structure of the *User* object that is constructed as more security capabilities are considered. Section 3.6 presents user

authorization capabilities that are utilized in an information system. Finally, Section 3.7, which provides a general overview of previous work security and access control.

3.1 Model: Application, Schema, Instances, and Users

In this section, we define the aspects and components of an information system in terms of the supported schema, the composition of an application, and the intended users. We assume that an information system that will need access control security across any combination of RBAC, LBAC, and/or DAC. Towards this purpose, we make an initial set of assertions, given in Table 3.1. The intent of these assertions is to represent the types of information system that whose security is to be modeled at a type level an enforced at an instance level.

Information and data is represented in a tree structure with a root node, where non-leaf nodes provide context (metadata, element name, typing information, cardinality constraints, etc.), and leaf nodes offer a place to store data values.
A schema is defined to organize the structure and content of a tree and act as a blueprint for instances. Schemas have the basic structures and context information (such as cardinality constraints, etc.). Akin to classes in an object-oriented programming languages or relations in a database, they contain the meta-data that describes the structure, elements, typing, etc. of a schema.
A schema can be instantiated which means that the actual data is created and defined. The structure and data in an instance is validated against it respective schema. This is equivalent to classes in object-oriented programming languages (create an instance of the person class) and to databases (create a tuple in a relation).
RBAC and LBAC are orthogonal. That is, each one can live independently from the other in case an application only needs RBAC or LBAC security.

Table 3.1: Information System Assertions.

The third assertion is important since the security model should provide support to information systems that may utilize either of those access control models. Note that we do not consider such an assumption for DAC. The reason for this is due to the fact that DAC capabilities live at a policy-level (authorizations) and not at the document level

(such as read/write operations controlled by RBAC, LBAC, or both); DAC involves the way that users utilize instances and pass on permissions among themselves.

Building on these assertions, the first set of definitions involves a characterization of the information system application, its structure via a set of schemas and its instances (data) that are authorized to a set of users. To assist the discussion, Figure 3.1 shows a sample application, its instances, and its users, for the `GetWellMobile` application of Section 2.6 utilizing the same users, Health Level 7 Clinical Document Architecture (CDA) (Alschuler et al., 2002) and the Continuity of Care Record (CCR) (Kibbe et al., 2004) as schemas, and `Carol Smith's` health record as instances.

Defn. 1: An element $e = < e_{ID}, e_{NAME} >$ is defined as an ordered pair component in a hierarchical data structure that contains information against which operations are performed, where e_{ID} is a unique identifier for the element and e_{NAME} is the tag of the element.

Example: The elements in Figure 3.1 have unique identifiers e_{ID} such as `eId1`, `eId2`, etc., and names e_{NAME} such as `ClinicalDocument`, `patient`, etc.

Defn. 2: A schema $S = < S_{ID}, S_{NAME}, E >$ is defined as a hierarchical organization of *n* elements represented as a set $E = \{e_1, e_2, ..., e_n\}$ for a given purpose in the form of a tree, where every $e_i \in S$ and there is a single root node in the tree of the schema S. In this case, S_{ID} is a unique identifier for the schema S, and S_{NAME} is the tag of the schema.

Example: The schemas utilized in the healthcare scenario are from the CCR and HL7 CDA as given previously in Figures 2.4 and 2.5. Figure 3.1 shows the way

that these schemas would be represented in the notation of the model, with identifiers `sId1` for the CCR, and `sId2` for the CDA. In addition, Figure 3.1 also contains the two element sets E_1 and E_2 for the instantiation of sets S_1 and S_2, respectively.

Defn. 3: An instance $i = < i_{ID}, i_{NAME} >$ of a schema S is referred to as a document that contains values for each of the n elements; instances must follow the structure of a schema, where i_{ID} is the unique identifier for the instance and i_{NAME} is the tag that describes the instance

Example: The instances utilized in the healthcare scenario are the CCR and HL7 CDA instances of Carol Smith's medical record as given in Figures 3.2 and 3.3. Using the model, in Figure 3.1, the two instance sets for `sId1` and `sId2` are given.

Defn. 4: Each schema S_j has an instance set I_j which is defined as $I_j = \{i_{j_1}, i_{j_2}, ..., i_{j_{ms}}\}$ that contains ms instances where each instance follows the format of the n elements that are defined by E and is validated against S_j.

Example: Since the information system utilizes the CCR and HL7 CDA representation of Carol Smith's medical record, the schema set for the `GetWellMobile` application consists of the two instance sets `sId1` and `sId2`. In this case, the instance set for CCR is Carol Smith's CCR instance, and for HL7 CDA it is the respective instance.

Defn. 5: An application A is comprised of set of k schemas and g instance set pairs $A = \{< S_1, I_1 >, < S_1, I_2 >, < S_2, I_1 >, ... < S_k, I_g >\}$ where each pair represents one aspect of the information needs of an application.

Example: Figure 3.1 shows the information system application as the `GetWellMobile`, consisting of the set of paired schemas and their respective instances {`<sId1, CarolSmithCCR>, <sId2, CarolSmithCDA>`}.

Defn. 6: A user u is defined as a pair $< u_{ID}, u_{NAME} >$ that will have the ability to access portions of an application and uniquely identifies each user by u_{ID}.

Example: In Figure 3.1, the users of the `GetWellMobile` application are given for the scenario in Section 2.5. Each user has his/her own identifier, as exemplified by `<uId1, Brock Ketchum>` for Dr. Ketchum.

Defn. 7: Let $U = \{u_1, u_2, ..., u_j\}$ be defined as the set of j users for a given application A, where $u_j \in U$ and $u_j =< u_{ID_j}, u_{NAME_j} >$.

Example: The list of users for an application is a set of pairs that contain each user's identifier and name. In our scenario of Section 2.5, and as given in Figure 3.1, the set of users consists of Brock Ketchum, Elisa Fakington, Leroy, Jenkins and Gail represented as {`<uId1, Brock Ketchum>, <uId5, Elisa Fakington>, <uId10, Leroy>, <uId11, Jenkins>, <uId20, Gail>`}.

In summary, as given in Figures 2.4 and 2.5 for the CDA and CCR schemas, respectively, please consider Figure 3.2 which contains an instance of the CDA schema and Figure 3.3 which contains an instance of CCR. CDA has <xsd:element name="clinical_document_header"> as a root, while CCR schema has <ContinuityOfCareRecord>. In turn, each of these structures have several children node (shown partially in Figures 3.2 and 3.3), which make the set of elements that are part of a schema *S* and instance *i*.

Elements e	<eId1, ClinicalDocument>, <eId2, patient>, <eId3, name>, <eId4, given>, <eId5, family>, …, <eId8, clinical_document_header>, <eId9, patient>, <eId10, name>
Element Sets E	E_1 ={<eId1, ClinicalDocument>, <eId2, patient>, <eId3, name>, <eId4, given>, <eId5, family>}
	E_2 ={<eId8, clinical_document_header>, <eId9, patient>, <eId10, name>}
Schemas S	S_1={<sId1, Continuity of Care Record, E1>}
	S_2= {<sId2, Clinical Document Architecture,E2>}
Instances i	<iId1, CarolSmithCCR>, <iId2, CarolSmithCDA>
Instance Set I	ccrRppt={<iId1, CarolSmithCCR>} cdaJeaderRppt= {<iId2, CarolSmithCDA>}<iId1<iId2
Application A	GetWellMobile = {<sId1, iId1>, <sId2, iId2>}
Users u	<uId1, Brock Ketchum>, <uId5, Elisa Fakington>, <uId10, Leroy>, <uId11, Jenkins>, <uId20, Gail>
User Set U	{<uId1, Brock Ketchum>, <uId5, Elisa Fakington>, <uId10, Leroy>, <uId11, Jenkins>, <uId20, Gail>}

Figure 3.1: Sample Elements, Schemas, Instances, Application, and Users.

```
<ClinicalDocument>
…
 <patient>
 <name>
 <given>Carol</given>
 <family>Smith</family>
 <suffix></suffix>
 </name>
 <administrativeGenderCode code="F" codeSystem="2.16.840.1.113883.5.1"/>
 <birthTime value="19830413"/>
 </patient>
 …
 <assignedPerson>
 <name>
 <given>Brock</given>
 <family>Ketchum</family>
 <suffix>MD</suffix>
 </name>
 </assignedPerson>
 …
</author>
<custodian>
 …
</custodian>
<documentationOf>
 …
 <performer typeCode="PRF">
 …
 <assignedEntity>
 …
 <code code="59058001" displayName="General Physician"/>
 …
 <assignedPerson>
 <name>
 <prefix>Dr.</prefix>
 <given>Elisa</given>
 <family>Fakington</family>
 <suffix/>
 </name>
 </assignedPerson>
…
 <component>
 …
 <title>History of Present Illness</title>
 <text>
 <content styleCode="Bold">Carol Smith</content>
 is a 31 year old female referred for further asthma management.
 Onset of asthma in her <content revised="delete">twenties</content>
 <content revised="insert">teens</content>.
 She was hospitalized twice last year, and already twice this year.
 She has not been able to be weaned off steroids for the past several
 months.
 </text>
 </section>
 </component>
 </structuredBody>
</component>
</ClinicalDocument>
```

Figure 3.2: Instantiated Portion of CDA for Carol's Medical Record.

```
<ContinuityOfCareRecord>
   …
   <Purpose>
   <Description>
   <Text>Summary of patient information</Text>
   </Description>
   </Purpose>
   <Body>
   <Problems>
   <Problem>
              …
   <Description>
   <Text>Asthma</Text>
   </Description>
   <Status>
   <Text>Active</Text>
   </Status>
              …
   </Problem>
   </Problems>
   <Alerts>
   <Alert>
        …
   </Alert>
   </Alerts>
   <Medications>
   <Medication>
   …
   <Product>
   <ProductName>
   <Text>Zithromax</Text>
   <Code>
   <Value></Value>
   <CodingSystem>RxNorm</CodingSystem>
   </Code>
   </ProductName>
   </Product>
   </Medication>
   </Medications>
        …
   <Actor>
   <ActorObjectID>A1234</ActorObjectID>
   <Person>
   <Name>
   <CurrentName>
   <Given>CAROL</Given>
   <Family>SMITH</Family>
   <Suffix/>
   </CurrentName>
   </Name>
   <DateOfBirth>
   <ExactDateTime>1983-04-13T00:00:00Z</ExactDateTime>
   </DateOfBirth>
   <Gender>
   <Text>Female</Text>
   <Code>
   <Value/>
   </Code>
   </Gender>
   </Person>
      …
  </ContinuityOfCareRecord>
```

Figure 3.3: Instantiated Portion of CCR for Carol's Medical Record.

3.2 Model: Schema Operations for RBAC, LBAC, and DAC

In this section, the work of Section 3.1 is extended to introduce concepts related to the ability to secure the schemas of a document-based application using RBAC, LBAC, and/or DAC in order to enforce the defined security at the instance level thereby customizing instances that are delivered to authorized users. Specifically, the schema operation "|" is defined and used to transform a schema from one state to the next state under some set of assumptions and/or constraints as dictated by security that is defined for RBAC, LBAC, and DAC. The *schema projection operation (SPO)* is used to transition an original schema (tree-structured) to a projected schema which identify those portions of the original schema that require access control constrained by various factors; the projected schema is a subset of the original one. The SPO is identifying those limited portions (up the entire schema) to have security definition and enforcement. Projection takes an original schema and removes those elements which do not need protection resulting in a schema that contains elements that require security. SPO prunes the original schema's tree-structure to produce a valid subtree as a projected schema which is contained in and may be equal to the original schema. For RBAC, one type of projection is based on role to order to identify that subset of a schema can be available to that role creating a separate projected version for each role of every schema that the role requires. For LBAC, SPO identifies the secure subset of the schema which needs control by assigning classifications (CLSs) to elements (see Section 2.2 again) that is then followed by decoration (see below).

To complement SPO, the *schema decoration operation (SDO)* is used to augment an original schema with new data related to security that is associated with a schema's

elements, resulting in a decorated schema that exceeds in structure the original schema. For LBAC, decoration adds CLSs to the elements of schema, transitioning an element's definition from a two tuple $el =< e_{ID}, e_{NAME} >$ (see **Defn. 1**) to the triple: $el =< e_{ID}, e_{NAME}, e_{SL} >$ where e_{SL} is the classification of the element (see Section 2.2 again). As a result, decoration enlarges the schema with the addition of a CLS field for each element that needs a CLS, so the decorated schema actually gets larger and contains the undecorated schema (whether it is an original or a projected one). While SPO and SDO aimed to support RBAC and LBAC, the schema operator "|" can be modified to define new criterion over which the transformation will occur per the discretion of the security administrator. This opens the possibilities to prune and/or decorate a tree-structured document with respect to other attributes (e.g. user tenure, time context, value constraints, etc.). Below we briefly summarize the assertions for this portion of the model.

In the design process for an application, $A = \{< S_1, I_1 >, < S_1, I_2 >, < S_2, I_1 >, ... < S_k, I_g >\}$ serves as a starting point from which a security administrator will being to define privileges. Specifically, using SPO an SDO a designer can project/decorate to arrive a version of the application (which schemas projected/decorated) resulting in a projected (decorated) application where $A^{P(r|c|e)}$ ($A^{D(c)}$) that contains $m \leq k$ schema/instance pairs from S, where $m = \left|A^{P(r|c|e)}\right|$ ($m = \left|A^{D(c)}\right|$), that have been projected/decorated in order to identify the portions of A that require a level of security control as realized in $A^{P(r|c|e)}$ ($A^{D(c)}$). The remainder of this section reviews definitions involving schemas for SPO and SDO.

A schema is a hierarchically structured collection of elements which can be altered via a schema projection operation (SPO), denoted as $\|^{P}_{r\|c\|e}$, that can filter the tree into a proper subtree.
$S_i^{P(r\|c\|e)}$ is the result of a projection $\|^{P}_{r\|c\|e}$ of S_i that effectively identifies the subset of S_i that has been filtered. Using this notation: $S_i^{\Pr}$ means that a role has been utilized to create a subtree of the original schema that represents all of the elements allowed for role r; S_i^{Pc} means that a CLS c (e.g., TS, C, etc.) has been utilized to create a subtree the original schema that represents all of the elements allowed for that c; and, S_i^{Pe} means that a subtree of the original schema) that represents all of the elements of the schema that need to be protected.
Given a projection $S_i^{P(r\|c\|e)}$ of schema S_i, the instance set is also projected, creating $I_i^{P(r\|c\|e)}$ which are now filtered instances for the schema that must follow the structure of the filtered schema. This yields an instance that has been project by role r, CLS c, elements e of the schema.
A schema which is a hierarchically structured collection of elements can be extended via a decoration operation, denoted as $\|^{D}_{c\|t}$, over a criterion that alters the form of the elements by adding new information in the form of LBAC sensitivity levels or time constraints.
$S_i^{D(c\|t)}$ is the result of a decoration $\|^{D}_{c\|t}$ of S_i that effectively identifies the decorated version of S_i based on LBAC sensitivity levels. Using this notation: S_i^{Dc} means that CLSs chosen from the set of all possible sensitivity levels (e.g., S, TC, C, U) have been added to all of the elements of the original schema; and, S_i^{Dt} means that time constraints have be included that represent the allowable timeframe for each element.
Given a decoration $S_i^{D(c)}$ of schema S_i, the instance set is also decorated, creating $I_i^{D(c)}$ which are now decorated instances for the schema that must follow the new structure of the decorated schema. This yields an instance that has expanded by classification or time.

Table 3.2: Schema Security Assertions.

Defn. 8a. $\|^{P}_{r}$ is a *schema projection operation (SPO)* over an RBAC role r and tree such that the end result of the action is the creation of a pruned tree by RBAC role that is a subset of the tree being projected, meaning that the projected subtree is a subset of the original schema to represent which portions of the original schema can be access by role. In our notation, for a schema S_i, $\tilde{S}_i^{\Pr} = S_i\|^{P}_{r}$ means that $\tilde{S}_i^{\Pr}$ is a projected version of S_i with respect to a role r, the results of a filtering action.

Defn. 8b. $|_c^P$ is a *schema projection operation (SPO)* over an LBAC sensitivity c and tree such that the end result of the action is the creation of a pruned tree by LBAC that is a subset of the original schema to represent the portion of original tree that has a sensitivity level c. In our notation, for a schema S_i, $\widetilde{S}_i^{Pc} = S_i|_c^P$ means that $\widetilde{S}_i^{Pc}$ is a projected version of S_i with respect to an c, the results of a filtering action.

Defn. 8c. $|_e^P$ is a *schema projection operation (SPO)* over elements e and tree such that the end result of the action is the creation of a pruned tree that is a subset of the tree being projected, meaning that the projected subtree is a subset of the original schema to represent the portion of original tree that has had elements deleted resulting in a subtree of the original tree that has the elements to be secured.. In our notation, for a schema S_i, $\widetilde{S}_i^{Pe} = S_i|_e^P$ means that $\widetilde{S}_i^{Pe}$ is a projected version of S_i with respect to elements e, the results of a filtering action.

Defn. 9: $|_c^D$ is a *schema decoration operation (SDO)* over LBAC and tree such that the end result of the decoration is the creation of an expanded tree with sensitivity level or classifications on the elements of the tree. In our notation, for a schema S_i, $\widetilde{S}_i^{d} = S_i|_c^D$ means that $\widetilde{S}_i^{Dc}$ is an expanded version of S_i with respect to LBAC, the results of a decoration action.

Defn. 10: $\widetilde{A}^{P(r|c|e)|D(c)} \subseteq A$ is a subset of the information in an application's schemas that need to be protected through a process of SPO and/or SDO. Specifically, $\widetilde{A}^{P(r|c|e)|D(c)} = \{< \widetilde{S}_1^{P(r|c|e)|D(c)}, \widetilde{I}_1^{P(r|c|e)|D(c)} >,..., < \widetilde{S}_m^{P(r|c|e)|D(c)}, \widetilde{I}_n^{P(r|c|e)|D(c)} >\}$ is the result of

an SPO and/or SDO on the original k schema/instance set pairs, that results in the creation of a set of m schemas where the information that needs to be secured and is available for access control has been identified.

To illustrate the concepts related to projection and decoration, Figures 3.4 and 3.5 have an organization of the resulting tree-structured schema from Figures 2.4 and 2.5, respectively, that after projection and decoration operations over RBAC and LBAC criteria. In Figure 3.4, the `<xs:element name=”patient_encounter”>` element (and others) in the original CDA schema has been removed yielding $\tilde{CDA}$; in Figure 3.5, the `<xs:element name = ”FamilyHistory”>` element (and others) in the original CCR schema has been removed (explain further once final) yielding $\tilde{CCR}$.

```
<xs:element name=”ClinicalDocument”>
 <xs:element name=”patient”>
 <xs:complexType name=”name”>
 <xs:element name=”given”/>
 <xs:element name=”family”/>
 <xs:element name=”suffix”/>
 </xs:complexType>
 <xs:element name=”administrativeGenderCode"/>
 <xs:element name=”birthTime”/>
 </xs:element>
 <xs:complexType name=”component”>
 <xs:element name=”title”/>
 <xs:element name=”text”/>
 </xs:complexType>
</xs:element>
```

Figure 3.4: SPO and SDO to Generate $\tilde{CDA}$ Schema.

To accompany these two figures, we provide their corresponding instances. Specifically, Figure 3.6 shows the result of a projection operation over a human-guided criteria, such as one from the security administrator of the practice from the scenario in Section 2.6, for Carol Smith’s CDA, with `<ClinicalDocument>` as the root of the tree and two direct children: `<patient>` and `<component>` (each with their own set of context children nodes). Likewise, Figure 3.7 shows the result of a similar operation of

human-guided criteria from the security administrator of practice A) for the CCR, with a root node of <ContinuityOfCareRecord> and one direct child node: <Body> (with <Problems> and <Medications> children.

```
<xs:element name="ContinuityOfCareRecord">
 <xs:complexType>
 <xs:element name="Body">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Problems" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Problem" type="ProblemType"
                            maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:element name="Medication" minOccurs="0">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="Medication" type="MedicationType"
                             maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
      </xs:sequence>
      </xs:complexType>
     </xs:element>
 </xs:complexType>
 </xs:sequence>
</xs:element>
```

Figure 3.5: SPO and SDO of the $\tilde{CCR}$ Schema.

```
<ClinicalDocument>
 <patient>
 <name>
 <given>Carol</given>
 <family>Smith</family>
 <suffix></suffix>
 </name>
 <administrativeGenderCode code="F" codeSystem="2.16.840.1.113883.5.1"/>
 <birthTime value="19830413"/>
 </patient>
     <component>
      <title>History of Present Illness</title>
      <text>
      <content styleCode="Bold">Carol Smith</content>
      is a 31 year old female referred for further asthma management.
      Onset of asthma in her <content revised="delete">twenties</content>
      <content revised="insert">teens</content>.
      She was hospitalized twice last year, and already twice this year.
      She has not been able to be weaned off steroids for the past several
      months.
      </text>
     </component>
</ClinicalDocument>
```

Figure 3.6: Applying Figure 3.4 to Generate Carol's CDA Instance.

```
<ContinuityOfCareRecord>
 <Body>
   <Problems>
    <Problem>
     <Description>
      <Text>Asthma</Text>
     </Description>
     <Status>
      <Text>Active</Text>
     </Status>
    </Problem>
   </Problems>
   <Medications>
    <Medication>
     <Product>
      <ProductName>
       <Text>Zithromax</Text>
       <Code>
        <Value></Value>
        <CodingSystem>RxNorm</CodingSystem>
       </Code>
      </ProductName>
     </Product>
    </Medication>
   </Medications>
  </Body>
</ContinuityOfCareRecord>
```

Figure 3.7: Applying Figure 3.5 to Generate Carol's CCR Instance.

3.3 Model: RBAC Security

In the section, we continue with the definition of RBAC in our security model for tree structured documents. This set of definitions involves designing a RBAC security policy that supports the ability to specify custom versions of the schemas that are projected on a role-by-role basis. The allows the permissions that are defined on the various elements of a schema to be of as authorized to a user by role, we call these *role-based permissions (RBP)*; these definitions represent a formalization of a portion of the NIST RBAC model, which involves roles, objects, operations, permissions, and mutual exclusion. The set of assertions for RBAC is given below.

A schema that contains a set of elements has defined, for each of the elements, a set of operations that are allowable, namely: read, aggregate, insert, update, and delete. These operations dictate the way that each element can be utilized.
A role is a means to characterize privileges that are based on usage behaviors of the application coupled with needed functionality to arrive at a construct (role) that is able to quantify a common set of responsibilities that are shared by multiple users. Roles will be authorized to utilize different portions of the schema and instances for an information application. Each application has a set of roles.
A permission is defined on each element in a schema by associating an element (by id) with an operation. The collection of all permissions for all roles across all schemas of an application is the role-based permission set, where each set member binds a permission to a role.
The schema projection operation (SPO) (see **Defn. 8a)** can be applied by role in order to identify the elements of each schema and the associated permissions per role.

Table 3.3: RBAC Assertions.

Defn. 11: A role r is defined as a two-pair $r = < r_{ID}, r_{NAME} >$ representation of the responsibilities for a user $u \in U$ to access some portion (entire or further restricted) of a schema.

Example: Consider Figure 3.8 with the sample roles for the scenario of Section 2.6. The roles, with identifiers, for `GetWellMobile` would be `{<rId1,Psychologist>, <rId2,Physician>, <rId3,Nurse>, <rId4,Staff>}`.

Defn. 12: Let $R = \{r_1, r_2, ..., r_j\}$ be defined as the set of j roles for a given application A, where $r_j \in R$ and $r_j = < r_{ID_j}, r_{NAME_j} >$.

Defn. 6 (RBAC Version): A user that has been authorized to RBAC (but not LBAC) redefines the *user* u as a tuple $< u_{ID}, u_{NAME}, u_{r_{ID}} >$, where $u_{r_{ID}}$ is the role as defined in **Defn. 26**. Note that a user can be authorized to more than one role, but is limited to playing one role in any application session.

Example: Consider the users of `GetWellMobile` from Figure 3.8. When their roles are assigned, the user is augmented with a role identifier. In the case of the

users of `GetWellMobile`, their users are:

```
{<uId1, Brock Ketchum, rId1>, <uId5, Elisa Fakington, rId2>,
<uId10, Leroy, rId3>, <uId11, Jenkins, rId3>,
<uId20, Gail, rId4>}.
```

Defn. 13: There exists a schema projection operation $|_r^P$ by role that acts on a schema S_i, and yields a filtered version $\widetilde{S}_i^{\Pr}$ for any given role $r \in R$. This results in what is referred to as a role-secured schema.

Defn. 14: The set of role-secured schemas $\widetilde{A}_S^{\Pr} = (\widetilde{S}_1^{\Pr}, \widetilde{S}_2^{\Pr}, ..., \widetilde{S}_i^{\Pr})$ is the set of role projections of $\widetilde{S}_i^r$ for role r, for any given schema i, where $i = 1,2,...,m$.

Defn. 15: Let $O = \{read, aggregate, insert, update, delete\}$ be the set of operations that can be performed against an element in each schema from either an original or a projected and/or decorated tree. The operation is defined at the schema level to be applied at the instance level. For permissions, each $op \in O$ will be assigned to individual roles.

Defn. 16: A permission p is represented by the four tuple $p =< p_{ID}, s_{ID}, e_{ID}, op >$, where p_{ID} is the unique identifier of the permission (akin to u_{ID} and r_{ID}), S_{ID} is the identifier of a schema $\widetilde{S}_i^{P(r|c|e)|D(c)} \in \widetilde{A}_S^{P(r|c|e)|D(c)}$ and e_{ID} is the identifier of an element $el \in \widetilde{S}_i^{P(r|c|e)|D(c)}$ for some $i = 1,2,...,m$, meaning that operation *op* can be performed on the element (node) identified by e_{ID} constrained to the schema $\widetilde{S}_i^{P(r|c|e)|D(c)}$ of the application.

Defn. 17: Let $P = \{p_1, p_2, ..., p_z\}$ be defined as the set of all granular permissions in a given application A, where a permission $p \in P$. That is, *P* is the set of all

permissions defined by the security administrator in a given application *A*.

Example: A set of permissions for the scenario of Section 2.6. is shown in Figure 3.8. The set `{<pId1, sId1, eId1, read>, <pId2, sId1, eId1, insert>, …, <pId6, sId2, eId9, update>, <pId7, sId2, eId10, update>}` denotes a read permission over the CCR schema's `eId1` element, an insert permission over the CCR schema's `eId1` element, an update permission over the CDA schema's eId9 element, and an update permission over the CDA schema's `eId10` element.

Defn. 18: The set of all permissions, the role-*permission assignments* $RPA = \{< r_{ID_1}, p_{ID_1} >,..., < r_{ID_k}, p_{ID_n} >\}$, where r_{ID} is the identifier of a role *r* as defined in **Defn. 12**, and p_{ID} is the identifier of a permission *p* as defined in **Defn. 16**, contains all the role-permissions pairs meaning that role with identifier r_{ID} can perform the permission with identifier p_{ID}.

Example: Consider Figure 3.8, which follows the healthcare scenario utilized so far. Sample role-permission assignments would be in the form of `{<rId1, pId1>, <rId1, pId6>, <rId1, pId7>, …, <rId2, pId1>, <rId2, pId2>, <rId2, pId7>, …, <rId3, pId1>, <rId3, pId2>}`. These pairs represent the following: the role Psychologist can perform the permissions with the identifiers `pId1`, `pId6` and `pId7`. The role Physician can perform the permissions with the identifiers `pId1`, `pId2` and `pId7`. Last, the role Nurse can perform the permissions with the identifiers `pId1` and `pId2`.

It is important to note that RPA can have multiple entries for a given role, where each entry represents one of the permissions that have been assigned to a role with a unique

r_{ID} . Recall that a permission is defined as a tuple $p =< p_{ID}, s_{ID}, e_{ID}, op >$, where p_{ID} acts as the identifier for the whole permission p. Role secured schemas are in turn the result of a projection of a role's permissions and the elements they target. This $\widetilde{S}_i^{P(r|c|e)|D(c)}$ can be empty, meaning that a role $r \in R$ has no nodes to act upon.

```
Roles:              {<rId1,Psychologist>, <rId2,Physician>, <rId3,Nurse>,
                    <rId4,Staff>}
Elements:           {<eId1, ClinicalDocument>, <eId2, patient>,
                    <eId3, name>, <eId4, given>, <eId5, family>, …,
                    <eId8, clinical_document_header>, <eId9, patient>, <eId10,
                    name>}
Operations:         {read, aggregate, insert, update, delete}
Permissions:        {<pId1, sId1, eId1, read >,
                    <pId2, sId1, eId1, insert >, …,
                    <pId6, sId2, eId9, update >,
                    <pId7, sId2, eId10, update >}
Role-Permissions    {<rId1, pId1>, <rId1, pId6>, <rId1, pId7>, …, <rId2, pId1>,
Assignments:        <rId2, pId2>, <rId2, pId7>, …, <rId3, pId1>, <rId3, pId2>}
Users:              {< uId1, Brock Ketchum>, <uId5, Elisa Fakington>,
                    <uId10, Leroy>, <uId11, Jenkins>, <uId20, Gail>}
```

Figure 3.8: Sample Roles, Operations, Permissions, Role-permission Assignments and Users.

The next set of definitions involve the definition of separation of duty and mutual exclusion as defined between roles and their associated permissions. *Separation of duty* between two users playing different roles means that in the process of working on a specific task, their actual duties (privileges) must be separated. For instance, in a hospital setting, when administering significant intravenous medications (say, for cancer treatment), there are two separate individuals in the process, one in a verifier role that checks that the dosage and patient name is correct and another in a patient care role that administers the IV medication; these two roles have a separation (has to be two separate persons) in order to offer protection to the patient receiving the medication with the permission to verify the mediation separate from the permission to administer the medication. Mutual exclusion ensures that two or more specific roles may not be assigned to any particular user. Continuing with the example, the verifier and patient care roles can never be assigned to the same user.

Defn. 19: $\Diamond_{SoD}$ denotes separation of duty between roles which means that, given r_x $\Diamond_{SoD}$ r_y, with $r_x \in R$ and $r_y \in R$, a user with role r_x cannot be assigned and perform any permissions of role r_y, where the two tuple $< r_x, r_y > \in SoD$ represents the SoD between the two roles. The set *SoD* contains all the pairs of roles that have a separation of duty relation.

Defn. 20: For a given role r_x, the set $SoD^{r_x} = \{< r_x, r_i > | < r_x, r_i > \in SoD \forall i\}$ where *i* iterates over all of the roles in *R*.

Example: As discussed above, consider that r_x is the verifier role and that r_y is the patient care role. The duties of these two roles are different, and the security administrator at the hospital has decided that for security purposes, an explicit separation of duty must be defined. In this case, $SoD^{r_{Verifier}} = \{< r_{Verifier}, r_{PatientCare} >\}$.

Defn. 21: $\Diamond_{ME}$ denotes separation of duty between roles that is strictly defined as two roles not allowed to have any permissions in common which means that, given r_x $\Diamond_{ME}$ r_y, with $r_x \in R$ and $r_y \in R$, the set of *RPA* does not have pairs with r_x and r_y where the p_{ID} are the same. This defines a set *ME* that contains all the pair of roles that have a mutual exclusion relation. More specifically, $< r_x, r_y > \in ME$.

Defn. 22: For a given role r_x, the set $ME^{r_x} = \{< r_x, r_i > | < r_x, r_i > \in ME_A \forall i\}$ where *i* iterates over all of the roles in *R*.

Example: Consider the roles of `Nurse` and `Staff` from Figure 3.8. The security administrator has realized that a staff person (e.g. `secretary`) cannot be assigned the role of nurse because of the professional requirements. In this case,

$ME^{r_{Nurse}} = \{< r_{Nurse}, r_{Secretary} >\}$

The previous definitions create a user object that redefines **Defn. 6** as follows:

Defn. 6 (RBAC with SoD and ME): A user that has been authorized to RBAC with *SoD* and *ME* constraints redefines the *user* u as a tuple $< u_{ID}, u_{NAME}, r_{ID}, SoD^{r_{ID}}, ME^{r_{ID}} >$, where $SoD^{r_{ID}}$ is the resulting set of roles that have a separation of duty for role r_{ID} and $ME^{r_{ID}}$ is the resulting set of roles that have a mutual exclusion with r_{ID}.

Example: In the *SoD* example above, assume the user $u_{JaneSmith}$ with role $r_{Verifier}$ would have the user tuple:

$< u_{Id7}, u_{JaneSmith}, r_{Verifier}, \{< r_{Verifier}, r_{PatientCare} >\}, \varnothing >$.

Likewise, for the *ME* example above, the user with role would have the user tuple:

$< u_{Id9}, u_{JaneJones}, r_{Nurse}, \varnothing, \{< r_{Nurse}, r_{Secretary} >\} >$

Note that in order to correctly generate the security enforcement (see Chapter 5), it is necessary to be able to compute a transitive closure for r_x of both SoD^{r_x} and ME^{r_x}. For SoD^{r_x}, the transitive closure would be generated by starting with the tuple $< r_x, r_y >$ and using r_y to search for all of the separation of duties in SoD that are of the form $< r_x, r_z >$ for all *z* and continuing to recursively seek all of the other possible SoDs until the process terminates. The generation of the transitive closure for SoD^{r_x} will yield a set that may have conflicts, and if it doesn't, would then be utilized in the generation of the security enforcement. A similar process would apply for ME^{r_x}.

3.4 Model: LBAC Security

Given the general security definitions in Section 3.1, the relevant operations related to schemas in Section 3.2, and the RBAC capabilities in Section 3.3, this section focuses on LBAC. From the perspective of the research in this proposal, the security model and framework is realized as a combination of the three major access control models: RBAC, LBAC and DAC. In this section, we present the way that our security model supports LBAC features. The overriding premise of our security approach for LBAC, as reviewed in Section 2.2 is to provide the overall infrastructure and concepts that are needed to allow us to assign sensitivity levels, namely classifications to elements of schema and clearances to users. This will allow a user to only access those portions of instances for the schemas that they have been authorized to. As in the previous two sections, we start off with a set of assertions, given in Table 3.4.

There exists a lattice of sensitivity labels that covers mandatory access control features in a generalized manner. These sensitivity levels correspond to classifications that are assigned to schema elements (see **Defn. 1**) and clearances that are assigned to users (see **Defn. 6**).
The lattice L of sensitivity levels is constructed from sensitivity levels that are bound by upper (most secure) and lower (least secure) values (see **Defns. 19** to **23**).
In support of our security model, we specialize from a lattice L of sensitivity levels to an ordered set $SL = \{x_1, x_2, ..., x_q\}$ of sensitivity levels, where SL, $x_1 < x_2 < ... < x_q$. This means that x_q represents the most secure sensitivity level and x_1 represents the least secure sensitivity level (see **Defn. 24**).
An element of a schema can be assigned a sensitivity level (classification) with a user being assigned a clearance. Since schema elements can be either be context nodes (refer to other elements) or data nodes, and each can be assigned a classification. In a subtree of a schema, the classification of a node cannot be more secure than the classifications of its children, and the entire tree's schema when annotated with classifications must satisfy the lattice relationship of sensitivity levels (e.g., in MAC, TS > S > C > U) with the least secure level tending toward the top of the tree and the most secure level tending towards the bottom of the tree.

Table 3.4: LBAC Assertions.

Given the assertions in Table 3.4, the following set of definitions for LBAC are provided. We assume there are q sensitivity levels; all represented in a chain (see **Defn. 24**). This set, denoted as SL, creates a linearly ordered set of levels that transitions the order operation $\lhd$ as defined in **Defn. 19** to a set of positive integers ordered by <, which is now the symbol to do ordering between a clearance and a classification. Note that while we consider q sensitivity levels, the case where $q = 4$ represents the typical MAC sensitivity levels in implementation: unclassified (U=1), classified (C=2), secret (S=3) and top-secret (TS=4).

Defn. 23: A *partial order relation* $\lhd$ is a binary relation that is reflexive, antisymmetric, and transitive.

Defn. 24: A *lattice* is a structure with a set L_S, a partial order $\lhd$, and two binary operations: *infimum* and *supremum*.

Defn. 25: Let $a,b \in L_S$. *infimum*(a,b) is called the *greatest lower bound* of a and b, or *meet*, and, *supremum*(a,b) is called the *least upper bound*, or *join* of a and b, such that:

a. For any $a,b \in L_S$, *infimum*(a, b) $\lhd$ *infimum*(a, b) $\lhd$ b, and for any $g \in L_S$, if g $\lhd$ a and g $\lhd$ b then g $\lhd$ *infimum*(a, b).

b. For any $a,b \in L_S$, a $\lhd$ *supremum*(a, b) and b $\lhd$ *supremum*(a, b), and for any $g \in L_S$, if a $\lhd$ g and b $\lhd$ g, then *supremum*(a, b) $\lhd$ g.

Defn. 26: If for any given $a,b \in L_S$, with $\lhd$ as the partial order relation, we have that $a \lhd b$ or $b \lhd a$, then we call L_S a *chain* or *totally ordered set*.

Defn. 27: We call a lattice $L = (L_S, \lhd)$ *bounded* if there exist elements *top* ($\top$) and

bottom ($\perp$) such that for any a $\in L_S$, $\perp \lhd a \lhd \top$.

Defn. 28: Define $SL = \{sl_1, sl_2, ..., sl_q\}$ to be the partially ordered set of q consecutive *sensitivity levels* for the application where sl_1 represents the least secure and sl_q represents the most secure; an individual level is referred to as $sl \in SL$.

Example: Consider the list of sensitivity levels in Figure 3.9. These sensitivity levels, `{unclassified (U), classified (C), secret (S), top-secret (TS)}` are a partially ordered set in which top-secret represents the most secure and unclassified represents the least secure level.

Defn. 29: Given $a,b \in SL$, the expression $a < b$ denotes that b has a higher sensitivity (classification or clearance) than a, which means that b is more secure. Similarly, the expression $a = b$ denotes that a and b have the same sensitivity.

Example: Consider the list of sensitivity levels from Figure 3.9. In this example, the relation between levels would be `U < C < S < TS`.

Defn. 30: We define a lattice $L = (SL, <)$ as the lattice of sensitivity levels in an application A, ordered by the relation < which has replaced $\lhd$ as defined in **Defn. 19**.

Defn. 31: Let $\tilde{S}_i^{Dc}$ be the subtree that results from the decoration $S_i|_c^D$ as defined in **Defn. 9**. The nodes of $\tilde{S}_i^{Dc}$ are LBAC decorated and in the form of $el = <e_{ID}, e_{NAME}, e_{SL}>$, where $el \in S_i$ or $el \in S_i^{P(r|c|e)}$ and $e_{SL} \hat{1} SL$ for any i. As a result, the schema has been extended due to the addition of an element for e_{SL}.

Example: Recall the elements for the CCR and CDA schemas following the healthcare scenario and as shown in Figures 2.4 and 2.5. When a decoration

operation is performed over the schemas, instances with their elements would look as follows (as shown in Figures 3.10 and 3.11): {<eId1, ClinicalDocument, C>, <eId2, patient, C>, <eId3, name, C>, <eId4, given, C>, <eId5, family, C>, …, <eId8, clinical_document_header, U>, <eId9, patient, S>, <eId10, name, S>}.

Defn. 32: If some node in a secure schema, $e \in \tilde{S}_i^{P(r|c|e)}$ with classification e_{SL}, has children nodes e_1 and e_2, then the classification $e_{1,SL}$ and $e_{2,SL}$ must be equal to or higher than the classification e_{SL}. That is, $e_{SL} \lhd e_{1,SL}$ or $e_{SL} == e_{1,SL}$ and $e_{SL} \lhd e_{2,SL}$ or $e_{SL} == e_{2,SL}$.

This definition means that the least secure levels migrate towards the top of the tree of the schema while the more secure level migrate towards the bottom leaf nodes.

Example: Following the previous example, the patient element in the CDA schema (and resulting instances as in Figure 3.10) would need to have a classification higher or equal than that of unclassified.

The result of a decoration operation with an LBAC criterion is exemplified in Figure 3.10 for Carol's CDA instance, and in Figure 3.11 for her CCR instance (decorations shown in bold red text). In these examples, we consider a set of sensitivity labels as SL= {TS, S, C, U}, the typical sensitivity levels for MAC (Bell & La Padula, 1976), a use-case of LBAC. Note that for the CDA, the <patient> element could be given a classification of secret (S), but the <family> element could have a classification of top-secret (TS). This is in order with **Defn. 28**, where a child node can be classified at the same or higher level of sensitivity than its parent. In Figure 3.11, the CCR has a base

classification of classified (C), as shown with the sl="C" tag in the root node.

```
Sensitivity Levels:    {top-secret, secret, classified, unclassified}
Elements:              {<eId1, ClinicalDocument>,
                       <eId2, patient>,
                       <eId3, name>, <eId4, given>, <eId5, family>, …,
                       <eId8, clinical_document_header>, <eId9, patient>, <eId10,
                       name>}
```

Figure 3.9: Sample Sensitivity Levels and Elements to be Secured with LBAC.

```
<ClinicalDocument sl="u">
 <patient sl="s">
 <name sl="s">
 <given sl="s">Carol</given>
 <family sl="ts">Smith</family>
 <suffix sl="s" />
 </name>
 <administrativeGenderCode
            code="F" codeSystem="2.16.840.1.113883.5.1" sl="s" />
 <birthTime value="19830413" sl="ts" />
 </patient>
    <component sl="ts">
     <title sl="ts">History of Present Illness</title>
     <text sl="ts">
     <content styleCode="Bold">Carol Smith</content>
     is a 31 year old female referred for further asthma management.
     Onset of asthma in her <content revised="delete">twenties</content>
     <content revised="insert">teens</content>.
     She was hospitalized twice last year, and already twice this year.
     She has not been able to be weaned off steroids for the past several
     months.
     </text>
     </component>
    </component>
</ClinicalDocument>
```

Figure 3.10: Result of an LBAC Decoration of Carol's CDA Instance.

```
<ContinuityOfCareRecord sl="c">
 <Body sl="c">
   <Problems sl="c">
   <Problem sl="c">
   <Description sl="c">
   <Text>Asthma</Text>
   </Description>
   <Status sl="s">
   <Text>Active</Text>
   </Status>
   </Problem>
   </Problems>
   <Medications sl="c">
   <Medication sl="c">
   <Product sl="c">
   <ProductName sl="c">
   <Text>Zithromax</Text>
   <Code>
   <Value></Value>
   <CodingSystem>RxNorm</CodingSystem>
   </Code>
   </ProductName>
   </Product>
   </Medication>
   </Medications>
  </Body>
</ContinuityOfCareRecord>
```

Figure 3.11: Result of an LBAC Decoration of Carol's CCR Instance.

The next portion of the LBAC model is the need to define the conditions under which sensitivity levels are compared. Specifically, when a user with a clearance wants to access an element with a classification, LBAC and MAC both provide different ways to compare the allowable read and/or write from CLR to CLR. Specifically, as given in Section 2.2, there are a number of different read and write permissions that are defined and enforced for a user against an application that are called *domination properties* that define the way to evaluate a user's CLR. Simple security (SS), or read-down, no read-up, is the permission to read at equal or lower levels so that means CLR > CLS for read. That is, a user with CLR is allowed to read elements with a CLS lower than their clearance level, but not those elements with a higher CLS. Simple integrity (SI), or write-down, no write-up, is the permission to write to equal or lower levels, meaning CLR > CLS for write. That is, a user can write elements with a lower CLS when compared to their clearance level, but not to those elements with a CLS. Liberal star (LS), or write-up, no write-down, is the permission to write to equal or greater levels (the opposite of SI) meaning CLR < CLS for write. Finally, Strict Star Write (SSW) and Strict Star Read (SSR), or write (read) equal, is the permission to write (read) only to equal levels of CLR as compared to CLS, meaning CLR = CLS for either read or write. Overall, the domination properties SS, SI, LS, SSR and SSW allow a security administrator to carefully control the usage of information, and most importantly, by setting the appropriate the read and write permissions of users it can be insured that a user will never violate the requirements when reading and writing to CLSs.

These features are comparisons performed over the nature of the operations that act in the schemas and instances. Building on this assertion, we define access modes and their

relationship to the operations as follows:

Defn. 33: $AM = \{AM - READ, AM - WRITE\}$ is the set of access modes that are used to categorize the multiple read oriented operations into the *AM-READ* category and multiple write operations in the *AM-WRITE* category that act against the secured tree nodes.

Defn. 34: Each $op \in O$ has an access mode assigned based on the operation. For non-destructive operations such as $\{read, aggregate\}$ have $am = AM - READ$, while destructive operations such as $\{insert, update, delete\}$ have $am = AM - WRITE$.

The operations from Section 3.3 have five values: read, aggregate, insert, update, and delete. When enforcing the read and write levels to check the domination properties SS, SI, LS, SSR and SSW, the comparison of a user's CLR against an element's CLS must first translate the operation into either AM-READ or AM-WRITE. When that has occurred, then the comparison of the CLR against the CLR can be performed based on the domination properties assigned to a user; that is, each user is assigned one read domination property and one write property. This translation is performed using the relation described in **Defn. 30**, which determines that non-destructive operations have an access mode of AM-READ, while destructive operations have an access mode of AM-WRITE.

This section is completed by utilizing all of the previous definitions, **Defn. 6** can be redefined for a user to capture the definition of CLR of a user along with the R-W properties (SS, SI, LS, etc.) to access elements. The prior two-tuple becomes a five tuple as below:

Defn. 6 (LBAC Version): A user that has been authorized to LBAC (but not RBAC) redefines the *user* u as a tuple $< u_{ID}, u_{NAME}, u_{CLR}, u_{LBAC-R}, u_{LBAC-W} >$, where $u_{LBAC-R/W}$ are either SS, SI, LS, SSR, or SSW.

Example: Building on the healthcare scenario of Section 2.6, the users of `GetWellMobile` would look as follows:

```
{<uId1, Brock Ketchum, TS, SS, SI>, <uId5, Elisa Fakington, S, SS,
SSW>,
 <uId10, Leroy, C, SS, LS>, <uId11, Jenkins, S, SS, SI>,
 <uId20, Gail, U, SS, SI>}.
```

This user definition is utilized in the situation where the security for the information application is limited to LBAC to define permissions on schemas that can be enforced on instances. For example, consider an insert operation which has an access mode of AM-WRITE. If the user Brock Ketchum, as described in **Defn. 6 (LBAC Version)** has a u_{LBAC-W} of SI (simple integrity), this means that can only insert new data in those elements of the schema that have a lower classification than his clearance.

3.5 Model: DAC Delegations

The next set of definitions involves DAC and delegation of authority that allows a role and its permissions to be passed among users. These definitions tackle the delegation of roles for an administrator-directed delegation where the security administrator, rather than the user, decides the delegation of roles and their validity based on constraints set by the model. Note that the concepts used in this work is based on the DAC/delegation concepts in prior research (Liebrand et al., 2003).

There exists a set of original users that have assigned roles and the ability to delegate those roles.
There is a set of delegable users that have the ability to receive roles from original users.
Pass-on delegation is an ability given to original users that allows an individual the authority to delegate the role even further. This means that an original user can delegate a role to a delegable user with pass-on delegation, resulting in the permission of that delegable user to pass the role on one step further to another user.
Delegation of roles is only allowed to delegable users. This means that original users can delegate the role to a user in the delegable users set, and in the case that pass-on delegation is authorized, the receiving delegable user can delegate the role to another user in the delegable users set.

Table 3.5: DAC Delegations Assertions.

Given the assertions in Table 3.5, we now proceed to discuss DAC and delegation aspects of the security model, where the security administrator models the allowed delegations that can be defined (**Defn. 32**) and the allowed delegations that can be received (**Defn. 33**) by each user.

Defn. 35: Let $DelRoles \subseteq R$, where $DelRoles = \{r_1, r_2, ..., r_y\}$ and $r_y \in R$, be defined as the subset of all roles that are eligible for delegation.

Defn. 36: The set of *Original Users*, $OU \subseteq U \times DelRoles$, $OU \subseteq U \times DelRoles$, where $DU = \{< u_1, r_1 >, ..., < u_x, ry >\}$, is the set of original users with their assigned roles. That is, the members of *OU* have the form $< u_x, r_y > \in OU$, which means that the user with identifier u_{ID} is allowed to delegate the role with identifier r_y.

Example: Consider Figure 3.8 where the original users of the application have been defined. The *OU* set would look as follows:

```
{<uId1, rId1>, <uId5, rId2>, <uId10, rId3>,
 <uId11, rId3>, <uId20, rId4>}.
```

Defn. 37: The set of *Delegable Users*, $DU \subseteq U \times DelRoles$, where $DU = \{< u_1, r_1 >, ..., < u_x, ry >\}$, is the set of delegable users with the roles they are

allowed to receive as part of a delegation. The members of *DU* have the form of $<u_x, r_y> \in DU$, which means that the user u_x can receive the role with identifier r_y when delegated by another user.

Example: For example, let us extend the scenario of Section 2.6. User Samantha, a fourth year medical student with the role `PhysicianIntern` with an identifier `uId30`; and, user Emily, a first year medical student also with the role `PhysicianIntern` with identifier `uId40`. The acting Physician, Dr. Elisa Fakington, has decided that Samantha should be delegated the role of `Physician` whenever she is not available. Elisa wishes to do this with pass-on delegation (see **Defn. 34**) so that Samantha can delegate the role to Emily should she not be available. Therefore, the *DU* set would look as follows:

```
{<uId30, rId2>, <uId40, rId2>}.
```

Defn. 38: Pass-On Delegation, or *PoD*, is a Boolean {0,1} that indicates whether a user and role pair $<u_x, r_x> \in DU$ can delegate the role r_x, originally received by a user and role pair $<u_y, r_x> \in OU$, one further step to another user u_z that is set as $<u_z, r_x> \in DU$. In this case, a *PoD* value of 0 means that the delegation has no pass-on authority. A value of 1 means that the delegation has a pass-on authority.

Example: Following the example from **Defn. 33**, the user Elisa might grant Samantha pass-on delegation so that in the case that Samantha is not available, Emily would take over the role of `Physician`. In this case, *PoD* would have a value of 1 (true).

Defn. 39: Delegable, or *Del*, is a Boolean {0,1} that indicate whether a user and role

pair $< u_y, r_y > \in OU$ or a user and role pair $< u_x, r_x > \in DU$ can execute the permission of delegation to another user and role pair $< u_z, r_z > \in DU$, where $r_z = r_y$ or $r_z = r_x$.

Defn. 6 (RBAC with Delegation): A user that has been authorized to RBAC (but not LBAC) redefines the *user* u as a tuple $< u_{ID}, u_{NAME}, r_{ID}, Del, PoD >$, where *Del* true means that the role can be delegated and when *PoD* true means that the authority to delegate that role can be passed on when the role is delegated. If *Del* is false, *PoD* must be false; if *Del* is true, *PoD* can be either true or false.

Example: Following the scenario in Section 2.6, users of Figure 3.11 are the members of the *OU* set. Their role delegation capabilities would look as follows:

```
{<uId1, Brock Ketchum, rId1, false, false>,
 <uId5, Elisa Fakington, rId2, true, false>,
 <uId10, Leroy, rId3, true, false>,
 <uId11, Jenkins, rId3, true, false>,
 <uId20, Gail, rId4, true, true>}.
```

What this means is that the security administrator has several users defined (e.g. Brock Ketchu, Elisa Fakington, etc.) with their roles (e.g. rId1, rId2, etc.). The security administrator then takes the user-role assignment and defines the delegation capabilities by leveraging the set of original users (see **Defn. 32**) and the delegable users (see **Defn. 33**). At the model level we then capture delegation as the user-role permission of delegation to a secondary user (e.g. Samantha or Emily from **Defn. 33**) with the pass-on delegation permission (see **Defn. 34**). All the information needed for the delegation is

available in the model in the form of the user object (see **Defn. 6 (RBAC with Delegation)**) and the user sets

The example of **Defn. 6 (RBAC with Delegation))** shows that Brock Ketchum cannot delegate his role of `Psychiatrist`. Elisa Fakington can delegate her role of `Physician`, but without Pass-on Delegation (i.e. Samantha or Emily, which would be in the set of Delegable Users, *DU*, from the example of **Defn. 33** would not have PoD). The nurses Leroy and Jenkins follow the same role delegation rules as Elisa Fakington. Last, Gail can delegate her role with PoD (the user who is assigned the role can delegate it one step further). The users that act as recipients of these delegations, if triggered, must be members of the Delegable Users (DU) set. Those users of the application that are not part of the DU set cannot receive any role delegation.

3.6 Model: User Authorizations

In this section, we complete the model with a discussion of user authorizations and the authentication process. Recall that we define authority as a capability obtained by a *user/role* pair that allows him/her to perform operations over a schema's instances. In other words, if user *U* with role *R* has authorization for instance $I_k^{S_i}$, then *s/he* can perform the permissions tied to *R* over the instantiated elements in $I_k^{S_i}$. The following set of definitions describes the characterization of authorization with respect to instances and schemas.

Defn. 40: The set of *authorized schemas*, $AS = \{< u_{ID_1}, r_{ID_1}, s_{ID_1} >, ..., < u_{ID_x}, r_{ID_y}, s_{ID_z} >\}$, is defined as the schemas from the application A that have been assigned to a user/role combination. That is, the tuple

$< u_{ID}, r_{ID}, s_{ID} > \in AS$ represents an authorized schema to a user/role combination, where u_{ID} is the unique identifier for a user $u \in U$, r_{ID} is the unique identifier for a role $r \in R$, and s_{ID} is the unique identifier for a schema $S \in A$, where *S* is a role-secured schema or an LBAC decorated schema.

In **Defn. 35,** the end result is that,the set *AS* is a derived set from the users *U* set (see **Defn. 6 RBAC Version**) and the role-permission assignments *RPA* sets, in which the u_{ID} and r_{ID} are obtained from the *U* set, and the s_{ID} is obtained from the *RPA* using the r_{ID} as a seed.

Example: Recall the users from `GetWellMobile` in Figure 3.9. A sample schema authorization object for the user Brock Ketchum would look as follows: `<uId1, rId1, sId1>`. This means that Brock is authorized to the CCR schema under the role of Psychiatrist.

Defn. 41: The set of *authorized instances (AI*) is defined as the instances from the application A that have been assigned to a user/role combination. That is, the tuple $< u_{ID}, r_{ID}, i_{ID} > \in AI$ represents an authorized instance to a user/role combination, where u_{ID} is the unique identifier for a user $u \in U$, r_{ID} is the unique identifier for a role $r \in R$, and i_{ID} is the unique identifier for an instance $i \in A$.

Example: Following the example above, the user Brock Ketchum's instance authorization object would be: `<uId1, rId1, iId1>`. This means that Brock is authorized to the CCR instance of Carol Smith's record under the role of Psychiatrist.

The previous two definitions represent the authorization granted over schemas and instances. Since in our model the security is defined at the schema level and enforced at the instance level, schemas authorized to a user/role combination include those that have permissions defined over their elements (**Defn. 35** and **Defn. 6 RBAC Version**). Complete instances are authorized to the user/role combination, and the security is enforced with the permissions defined at their respective schemas. For example, if the instances of one schema S_1 is a patient record for inpatient visit (hospital) and another schema S_2 is a patient record for outpatient visit (MD office), we assign Carol Smith's instances of S_1 to user Elisa. Carol Smith's instances, following S_2, can be assigned to user Brock. The security over these instances is the result of a projection or decoration of the schemas with respect to the user/role (Elisa as a physician would use the security defined for her role in S_1 and Brock as a psychiatrist would use the security defined for his role in S_2, for example).

To complete the work, we bring together the three access control models (RBAC, LBAC, and DAC) in order to define the ability of users who play optional roles to access elements of tree-structure documents which may have optional security levels. In our proposed security model, DAC is considered a policy-level security component, which means that authorizations over schemas and instances are automatic due to the defined permissions (with regards to the schemas) and driven by a security-administrator (with regards to the authorized instances). In terms of RBAC and LBAC, no preset order is necessary in terms of which one is defined first for information applications where components from both models are required; order is irrelevant with the final result staying the same. The combined definition is:

Defn. 6 (LBAC + RBAC): A user that has been authorized to both RBAC and LBAC redefines the *user* u as a tuple $< u_{ID}, u_{NAME}, u_{CLR}, u_{LBAC-R}, u_{LBAC-W}, r_{ID}, SoD^{r_{ID}}, ME^{r_{ID}}, Del, PoD >$, where u_{ID} is the unique identifier for the user, u_{NAME} is the tag for the user u, u_{CLR} is the clearance level assigned to the user (per LBAC components), $u_{LBAC-R/W}$ are either SS, SI, LS, SSR, or SSW, r_{ID} is the unique identifier for the assigned role $r \in R$, $SoD^{r_{ID}}$ is the resulting set of roles that have a separation of duty for role r_{ID}, $ME^{r_{ID}}$ is the resulting set of roles that have a mutual exclusion with r_{ID}, *Del* means that the role with the identifier $u_{r_{ID}}$ can be delegated (Boolean value), and *PoD* means that pass-on delegation is allowed or not (Boolean value).

Example: Building on all the definitions and examples utilized in this chapter, the finalized user object for the `GetWellMobile` application would look as follows:

```
{<uId1, Brock Ketchum, TS, SS, SI, rId1, Ø, Ø, false, false>,
 <uId5, Elisa Fakington, S, SS, SSW, rId2, Ø, Ø, true, false>,
 <uId10, Leroy, C, SS, LS, rId3, Ø, Ø, true, false>,
 <uId11, Jenkins, S, SS, SI, rId3, Ø, Ø, true, false>,
 <uId20, Gail, U, SS, SI, rId4, Ø, Ø, true, true>}.
```

Figure 3.12 demonstrates the way that a tree-structured schema would look after RBAC project and projections are performed. In the figure notice that from the original schema to the LBAC decorated schema, there has been a prune and decoration. This is the execution of **Defn. 8c** and **Defn. 9**. When decorating the original S schema, there was first a projection operation over those nodes that needed a classification. The resulting $\tilde{S}$ was then further pruned via a projection operation over a role *R*, yielding the

schema $\approx\!\!\approx$. The end result of this process is a role-secured schema with classification labels on the nodes that would provide RBAC and LBAC security.

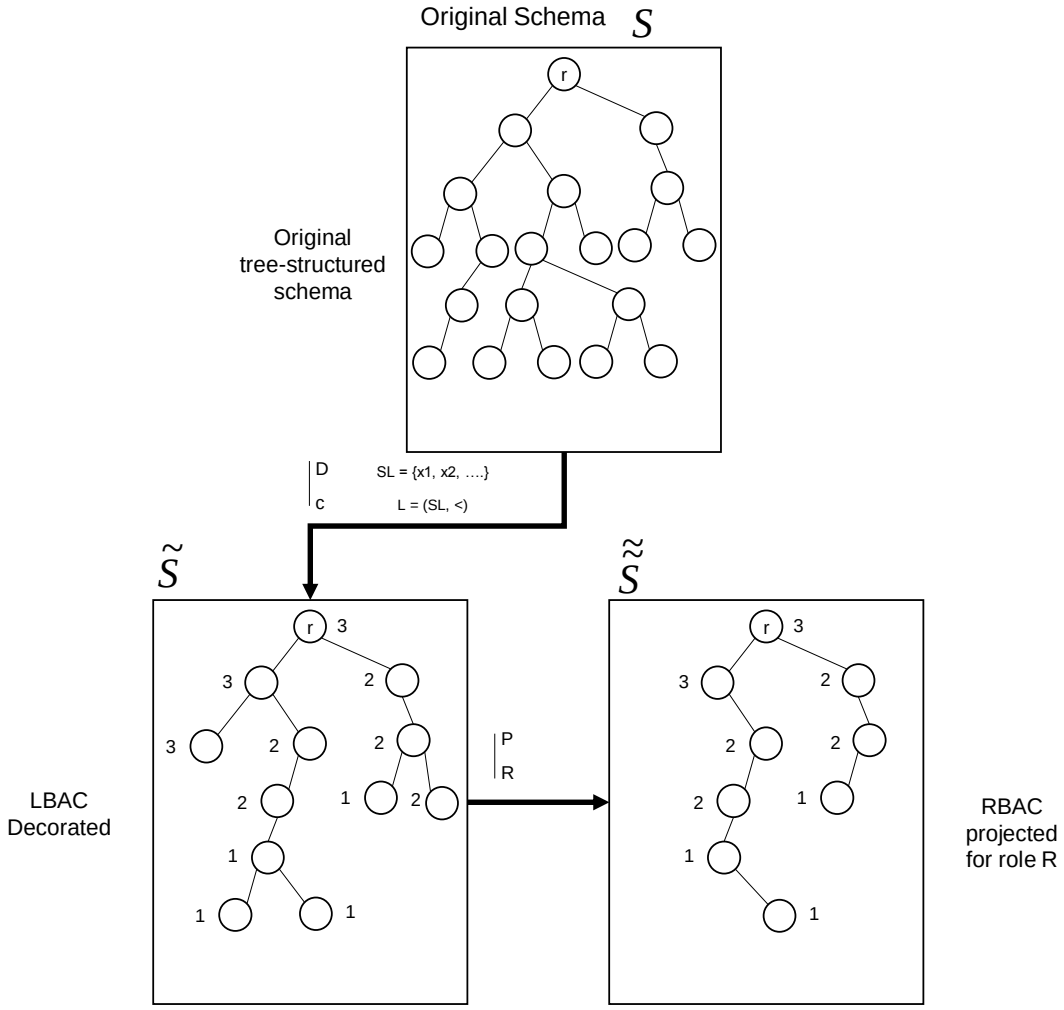

Figure 3.12: LBAC Decoration and RBAC Projection of a Tree-structured Schema.

3.7 Access Control Related Work

Access control enforcement in tree-structured documents, most commonly with 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 schema and documents to provide different views

based on the policies in place. This embedded enforcement can include either security policy definitions or cryptographic properties.

The work of (Damiani, De Capitani di Vimercati, S., Paraboschi, & Samarati, 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 also be embedded in the XML DTD (L. Sun & Li, 2006), 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 schemas in the newest specifications of the format. Second, the security policies are embedded into both the DTD and the instance. When policies are changed, the cost of updating the XML instances is huge.

Another effort by (Damiani, Fansi, Gabillon, & Marrara, 2008) 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. For example, XPath (Clark & DeRose, 1999) design allows it only to perform reading queries to XML instances.

The encryption of different sections of an XML document with different encryption keys is presented in (Bertino & Ferrari, 2002). 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 (Bertino, Castano, Ferrari, & Mesiti, 2002) 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 (Cao, Sun, & Wang, 2005; L. Sun & Li, 2006) 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 (Rahaman, Roudier, & Schaad, 2008) 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 destructive permissions such as insert, delete and update.

The work of (Leonardi, Bhowmick, & Iwaihara, 2010) considers the scenario of a federated access control model, in which the data provider and policy enforcement are handled by different organizations. This approach relates to ours with regards to the separation of the security policies from the data to be handled, but differs in the specifics of where the policies' details are stored. (Kuper, Massacci, & Rassadko, 2005) has presented a model consisting of access control policies over DTD's (again, outmoded in XML) with XPath expressions in order to achieve XML security. The purpose of their model is similar to ours, as it aims to provide different authorized views of an XML document based on the user's credentials. However, the significant difference is that this approach combines query rewriting and authentication methods, whereas our approach can be applied to any non-normative XACML architecture (having a policy enforcement point) for both reading and updating, as well as XPath or XQuery queries.

Last, the work of (Müldner, Leighton, & Miziołek, 2009) presents an approach of supporting RBAC to handle the special case of role proliferation, which is an administrative issue that happens in RBAC when roles are changed, added, and evolve over time, making security of an organization difficult to manage. This approach supports the encryption of segments of the XML document. Our approach doesn't address role proliferation; however, by separating our security into an XACML policy, we do insulate role proliferation from impacting an application's XML schemas and instances.

Chapter 4
UML Security Extensions for Tree-Structured Documents

In this chapter, the second major component of our security framework for tree-structured documents involves the realization of the security model in Chapter 3 as a series of new unified modeling language (UML) diagrams (Fowler, 2004) that capture the characteristics of the security model and allow us to augment the software engineering process of UML with an information engineering process for tree-structure documents. Recall from Section 2.7 that UML provides a large variety of diagrams for the visualization of different software requirements: class, component, deployment, activity, use-case, state-machine, communication, sequence, etc. UML provide the benefit of reducing misinterpretation and promoting simple communication of domain requirements with its visual notation (Lange & Chaudron, 2005). However, while UML can be utilized to define security requirements, what is lacking in UML is actual diagrams that are dedicated to, in our interest, access control models (RBAC, LBAC, and DAC) that allow the definition of security requirements using new security UML diagrams that seamlessly integrate with the UML model and unified design process. This is particularly true for domains such as healthcare where the information to be utilized is private and often governed by legal constructs that assure its proper use and dissemination, as we have described in Section 2.6 and illustrated with a detailed example for our security model of Chapter 3.

The work presented in this chapter leverages off of early work that has extended UML with new diagrams for RBAC, mandatory access control (MAC) (Bell & La

Padula, 1976), and DAC capabilities (Pavlich-Mariscal, 2008; Pavlich-Mariscal, Demurjian, & Michel, 2010) from an object-oriented perspective. This prior work defines role slices that contain the methods of an object-oriented application that are both assigned and prohibited on a role-by-role basis, along with the ability to tag roles with classifications in support of MAC and support delegation of author. The work also generates aspect-oriented AspectJ enforcement code for a UML design augmented with security. The work in this chapter also meshes with work on extending NIST with collaboration of duty and adaptive workflow capabilities (Berhe, Demurjian, & Agresta, 2009; Berhe et al., 2010; Berhe, Demurjian, Gokhale, Pavlich-Mariscal, & Saripalle, 2011) that also added new UML diagrams to represent users, roles, collaborations, and the required interaction of users towards a common goal. To support enforcement in an object-oriented RBAC context, the Java Meta Language is leveraged to define policies that can be directly embedded in code. The work in this chapter augments the overall UML design process by providing new UML diagrams for the information engineering of tree structure documents (see the security model of Chapter 3) that is integrated with an overall application process that includes object-oriented and collaborative capabilities. These three approaches have been integrated into a secure software engineering approach for functional, collaborative, and information concerns for complex applications like the one given in Section 2.6 (Pavlich-Mariscal, Berhe, De la Rosa Algarín, A., & Demurjian, 2014).

The formal definition of the UML metamodel by OMG with the Meta-Object Facility (MOF) allows the extension of the modeling language with several degrees of formality, as reviewed in Section 2.7. MOF facilitates Model Driven Architecture (MDA) (Kleppe,

Warmer, Bast, & Explained, 2003; Soley, 2000), another standard that aims to platform independent models (PIM) (Burmester, Giese, & Schäfer, 2005) to platform specific models (PSM) (Kelly & Tolvanen, 2008). UML can be extended via the use of the MOF, which consists of four layers: 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). The other major benefit of UML that the security framework presented in this dissertation makes use of is the automatic generation of code from the diagrams (Montrieux et al., 2010), which has also been done in our prior work on RBAC, MAC, and DAC (Pavlich-Mariscal et al., 2010) and NIST extensions for collaboration of duty and adaptive workflow (Berhe et al., 2010). The security framework presented herein further extends and supplements that capability by mapping the new UML extensions for the security model in Chapter 3 with XACML policy elements. This results in the automatic generation of proper enforcement ready for deployment in information systems. Towards this purpose, we advance the information security problem from a software engineering perspective, elevating information security to a first-class citizen of the software design and development process.

In support of this work, this chapter presents a set of UML diagrams to realize the security model in Chapter 3 into a format that can be utilized to design an application that has functional (object-oriented), collaborative, and information security concerns. To accomplish this, we introduce new UML diagrams that correspond to the key security model characteristics as given in Chapter 3. First, there exists a necessity to represent a tree-structured schema (see Definitions 1 and 2 from Section 3.1) for which we propose

the creation of a new UML diagram to represent such a schema as a tree structured collection of elements, namely, the *Document Schema Class Diagram (DSCD*). Second, to identify those portions of the tree-structure document schema as captured in DSCD (see Definitions 8a, 8b, 8c, 9, and 10 in Section 3.2) , we propose the creation of a new UML diagram that identifies the subset of DSCD that needs to be protected, namely, the new *Secure Information Diagram (SID*). Third, to represent RBAC security capabilities (see Definitions 11 and 12 from Section 3.3), we propose a new UML diagram to represent the roles and provide the ability to define. For elements of a schema, permissions for RBAC (read, write, etc.) and classifications for LBAC, we propose the new UML *Document Role-Slice Diagram (DRSD).* Note that DSCD, SID, and DSRC are generalizations of our prior work that specified these diagrams for XML (De la Rosa Algarín, A. et al., 2012; De la Rosa Algarín, A. et al., 2013); these new document based diagrams can apply to different document standards (e.g., XML, JSON, OWL, etc.). Fourth, to support LBAC features (see Definitions 26, 27, and 28 in Section 3.4), we propose the new UML *LBAC Secure Information Diagram (LSID) that* builds off of the SID and serves as a means to define which elements in the tree-structured document need classifications. Fourth, to tie together RBAC, LBAC and DAC is the *User Diagram (UD)*, created to consider the orthogonal nature between RBAC and LBAC, as well as the role-delegation and authorization assignments from DAC. Fifth, to support the ability to have privileges delegated from one user to another (see Definitions 31 to 35 in Section 3.5), the new UML *Delegation Diagram (DD)* provides the ability to define which users and role pairs can delegate or be delegated. Lastly, to support the definition of users and their authorization (see Definition 6 in Section 3.1 and its augmented versions in Sections

3.3 and 3.4), we propose the new UML *Authorization Diagram (AD)* that provides the ability to determine which schemas and instances are tied to a user/role pair (see Definitions 35 and 36 in Section 3.6)

The remainder of this chapter is organized into four subsections. Section 4.1 introduces and motivates the usage of UML for secure information engineering for tree-structured documents and places our work into a context with previous work (Berhe et al., 2011; Pavlich-Mariscal et al., 2010) that includes functional, collaborative, and information security concerns. Using this as basis, Section 4.2 presents the new UML diagrams for our security model, namely, DSCD, DRSD, SID, LFD, DD, UD, and AD, and provides a detailed example of their usage using the healthcare scenario from Section 2.6 and by reformulating the example in Chapter 3 from a model to a UML context. Section 4.3 complements the formalization of the new UML diagrams by starting their definition via a utilization of the UML metamodel to define these new security diagrams. Section 4.4 brings all of these diagrams together and presents a *secure information engineering* process, and shows the way that this process it be placed into an appropriate context with the work presented in Section 4.1 for designing an healthcare application for functional, collaborative, and information concerns. This chapter concludes with detailed discussion on the related work in Section 4.5 as related to security modeling via UML.

4.1 Motivation and Usage of UML for Secure Information Engineering

The software development process has evolved over 30+ years from the *waterfall model* (Royce, 1970) to the *iterative model* (Larman & Basili, 2003) to the *spiral model* (Boehm, 1988) to the *unified process model* (Jacobson, 1999) to *agile development lifecycle* (Cohen, Lindvall, & Costa, 2003). Despite this long history, these processes

have yet to address challenges of large-scale applications that have varied concerns (users' interfaces, server functionality, database support, tracking and logging, security, etc.) that are often tangled, e.g., an object-oriented application, code to read/write the database can be spread across multiple classes even if the database is abstracted via Hibernate (Bauer & King, 2005). All of these different concerns end up intertwined and spread out across the application's varied components. As a result, the traceability of security through different aspects, which in our case involves functional, collaborative, and information, capabilities, cannot be easily isolated. This chapter presents UML extensions and a *secure information engineering* process that elevates information security to a primary step of the software development process as depicted in Figure 4.1 where the prior work on functional (Pavlich-Mariscal et al., 2010) and collaborative (Berhe et al., 2010) security with extensions with UML yields a secure engineering approach that encompasses the different aspects of a system's security. In our focus on information security, representing the tree-structured schemas with new UML diagrams allows 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.

To provide a motivation for the usage of UML for security design, consider Figure 4.1 more closely, where a secure software engineering approach for functional, collaborative, and information concerns (focused on tree-structured documents) is intended to visually model RBAC, LBAC, and DAC. From a functional perspective, the work of (Pavlich-Mariscal et al., 2010), extended UML with new diagrams for RBAC, MAC, and DAC to

identify a subset of the object-oriented classes and their methods that need to be securely controlled in terms of permissions (methods) assigned to users by roles, and to allow these roles to be assigned permissions via a new role-oriented diagram. From a collaborative perspective, the work of (Berhe et al., 2010) extended NIST RBAC and UML with new diagrams to support the specification of collaboration among users that need to coordinate their activities to achieve a particular tasks in a series of steps over time. Our work in this chapter adds an information perspective, to allow tree-structured documents to be securely handled in support of RBAC, LBAC, and DAC by allowing different portions of the instances of a document to be delivered to different users based on roles, permissions, and classifications. Individually, all three approaches can generate appropriate enforcement mechanisms (AspectJ code for functional, Java Meta Language code for collaboration, and XACML policies for information). The combination of all three concerns promotes security as an integral part of a secure software engineering approach, while tracking software quality assurance in terms of the consistency of the security and non-security requirements.

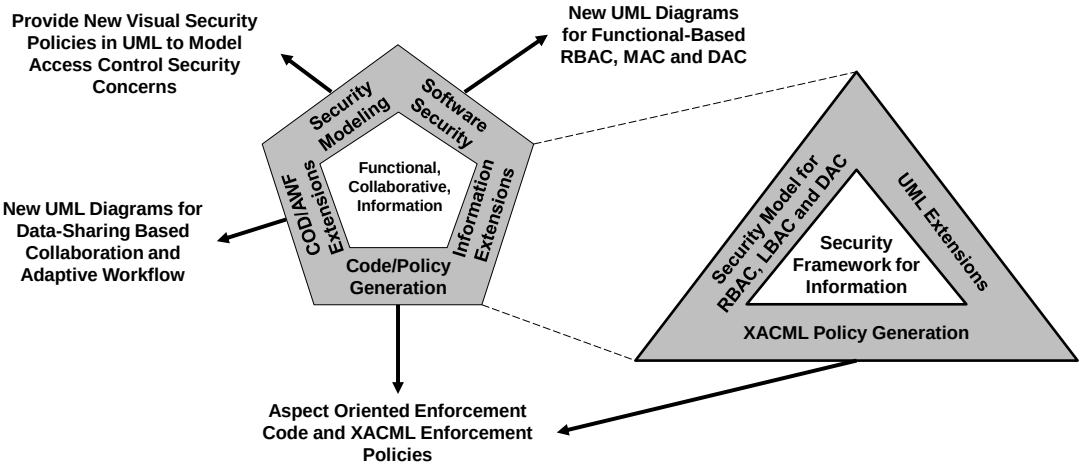

Figure 4.1: An Abstraction Process for Concerns.

4.2 UML Security Extensions for Tree-Structured Documents

The basis of the work presented herein leverages the work of (Pavlich-Mariscal et al., 2010) for functional security concerns (RBAC, MAC, and DAC) supported in an object-oriented context within UML with new diagrams in order to support the definition of informational security concerns for tree-structure documents via new UML diagrams for RBAC, LBAC, and DAC. Our work transitions the security model of Chapter 3 to support security of tree-structured document schemas and instances in a granular approach that focuses on the elements, rather than the work of (Pavlich-Mariscal et al., 2010) that has focused on providing secure access to methods. However, our work on a security framework for tree structured documents has been influenced by some concepts from (Pavlich-Mariscal et al., 2010), namely: the secure subsystem diagram (SSD), which denotes all of the classes and methods in the system that requires protection; the role-slice diagram (RSD), which provides a means to define permissions by role to the classes and methods that comprise the SSD; and, the user diagram (UD), which focuses in denoting the roles assigned to each user in the system, with the role retaining the actual permissions. The concept of identifying a secure portion via SSD corresponds to our identification of the portion of a document schema that needs to be protected, the ability to define a RSD for the methods allowed against SSD corresponds to our using the new document role-slice diagram to define permissions against a portion of the document schema, and, our user diagram corresponds to the actual document instances and portions thereof that are available to each user with a role from a permission perspective (e.g., read, write, etc.).

This section covers the new diagrams that extends UML with RBAC, LBAC, and DAC, with a specific target of tree-structured documents with elements that have defined schemas and instances (see Definitions 1 and 2 from Section 3.1). The UML extensions it this section generalize our prior work (De la Rosa Algarín, A. et al., 2013) that defined a set of new UML diagrams that were specialized for XML. By generalizing, we are able to provide a design approach in UML that can work for applications that will be built with XML, JSON, OWL, etc., all which can be aligned into a document-tree structure format. Towards that end, the remainder of this section reviews six new UML diagrams: DSCD, DRSD, SID, LFD, DD, and AD. More specifically, Section 4.2.1 introduces the *Document Schema Class Diagram (DSCD)* that can handle any tree-structured schema to model the document and realize the instance (see Definitions 8a, 8b, 8c, 9, and 10 in Section 3.2). Note that in showing examples for DSCD and other new UML diagrams, we utilize both the HL7 CDA and CCR schemas, which are specializations of a tree-structure document whose structure can be represented with the UML DSCD modeling construct called the UML Profile. The new UML *Secure Information Diagram (SID)* is presented in section 4.2.2, and is a UML extension that allows security administrators determine which subtree of the original document schema tree requires a level of security some sort of security (e.g. role filtering, LBAC sensitivity, etc.). Given DSCD and SID, the next new diagram in Section 4.2.3 is the *Document Role Slice Diagram (DRSD)* to define the role and the associated permissions to access elements (see Definitions 11 and 12 from Section 3.3). Next, in Section 4.2.4, LBAC is supported via the definition of a new UML *LBAC Secure Information (LSID)* to capture the capabilities of security model (see Definitions 26, 27, and 28 in Section 3.4) that are used to define access modes and

classifications for schema elements; this decorates the originally created SID. Then, Section 4.2.5 contains the *User Diagram (UD)* for the definition of users (see Definition 6 in Section 3.1 and its augmented versions in Sections 3.3 and 3.4). To handle delegation of roles between users, in Section 4.2.6, the new UML *Delegation Diagram (DD)* is proposed (see Definitions 31 to 35 in Section 3.5) to capture which users and roles are allowed to delegate authority in an application. Finally, Section 4.2.7 contains the *Authorization Diagram (AD)* that ties schemas and instances with the user/role combination to arrive at a culminating new diagram (see Definitions 35 and 36 in Section 3.6 and Definition 6 in Section 3.6).

4.2.1 The Document Schema Class Diagram (DSCD)

The new UML Document Schema Class Diagram (DSCD), shown in Figure 4.2 for HL7 CDA's clinical_document_header and section subtrees and in Figure 4.3 for the CCR segment of the healthcare example of Chapter 2, is an artifact that holds all of the characteristics of the schema, including structure, data type, and value constraints. The DSCD graphically represents the schemas utilized by an information system as described by Definitions 1 and 2 from Section 3.1. Recall that the assumption of the work presented in this dissertation is that schemas are characterized by a tree-structure, possibly complimented with data type constraints. To achieve this, we utilize a UML profile for tree-structured documents. There has been research in UML profiles for tree-structured documents, mainly utilizing XML (Bernauer, Kappel, & Kramler, 2003; Bernauer, Kappel, & Kramler, 2004), which range from information modeling (Carlson, 2008; Combi & Oliboni, 2006; Conrad, Scheffner, & Christoph Freytag, 2000; Routledge, Bird, & Goodchild, 2002) to systems modeling represented in XML (Bray et al., 1998). This

work also considers round-trip engineering, a concept that denotes the ability of producing a UML diagram from XML and vice-versa, without the loss of information. For the scope of the work in this dissertation, we generate a DSCD a UML diagram from a source tree-document schema, which for the purposes or demonstrating the concepts, is actually an XML schema. To facilitate this process, we utilize the UML Profile concept that allows new diagrams to be defined using the various UML concepts (stereotypes, tags, constraints applied to classes, attributes, operations, etc.) that allow a tree structured document to be transitioned into DSCD, and for the particular purposes of this section, to demonstrate the way that an XML schema (a tree structured format) can be transitioned to the new UML DSCD diagram; this is shown in Table 4.1.

While it is possible to utilize the UML profile to represent an entire schema as a UML package, we instead have chosen to represent each schema as a tree of stereotyped classes. This approach was chosen in order to capture the hierarchical structure of a schema as a series of related classes. Table 4.1 has three columns: the first column represents the features of tree structured document, the second column defines the corresponding XML equivalents of these features, and the third column transitions the second column into the equivalent UML profile concept. In the first row of Table 4.1, a general element in the tree-structured document is equivalent to an XML element (`xsd:element`) and is realized as a UML class; the second row maps the element name to a UML class name. In the third row of Table 4.1, an element attribute in the tree-structured document is equivalent to a generic attribute in XML which can be mapped to a «stereotyped» attribute in UML. The fourth row corresponds to a patient – child relationship at the schema level to identify a tree and its subtrees, which in XML is

observed as nested elements, and is represented as a UML dependency relationship in the DSCD. The fifth row of Table 4.1 describes complex elements (those that are built out of many sub-elements), which in XML are denoted as `xsd:complexType` and in the DSCD are denoted as a UML class with the «complexType» stereotype. The sixth row covers a similar case, considering sequences or lists of elements, which in XML are denoted as `xsd:sequence` and in the DSCD are denoted as a UML class with the «sequence» stereotype. Aggregation of attributes are handled with the seventh row of Table 4.1 and is represented as `xsd:attributeGroup` in XML and as a UML class with the «attributeGroup» stereotype in the DSCD. In the eighth row of Table 4.1, groups of elements in a tree-structured document are equivalent to an XML `xsd:group` node and is represented as a UML class with the «group» stereotype in DSCD. The ninth row of Table 4.1 handles acceptable or allowable values for elements, which in XML are usually `maxOccurs` and `minOccurs` attributes to an XML element constraints, realized as a «constraint» stereotyped class member in DSCD. In the tenth row of Table 4.1, indirect references allow elements of a tree-structure document to be associated with one another, which in XML is a ref attribute on an element that are represented as a «ref» class member from UML profile in the DSCD. Lastly, in the eleventh row of Table 4.1, for tree-structured document, the parent-child relationship between non-named elements corresponds to non-named elements in XML (e.g., `xsd:complexType`, `xsd:attributeGroup`, etc.) and is represented with a UML directed association relationship between classes in the DSCD. Note that by using these mappings in Table 4.1 it is possible to develop an algorithm that operate over an XML schema to generate a DSCD equivalent in UML. Note also that there would need to be other versions of Table

4.1 for other data formats (e.g., JSON, RDF, etc.) where the second column of the table would be replaced with the relevant model constructs from the other formats.

Tree-Structured Document Component	XML Analog	DSCD Component
General Element	Element	UML Class
General Element Name	Element Name	UML Class Name
General Element Attribute	Generic Attribute	Stereotyped Attribute
Parent – child relationship of a schema (tree-subtree)	Tree - Subtree	UML Dependency Relationship
Complex Type of Elements and/or Attributes	XML xs:complexType	Stereotyped «complexType» UML Class
Sequential Element Order	XML xs:sequence	Stereotyped «sequence» UML Class
Aggregation of Attributes	XML xs:attributeGroup	Stereotyped «attributeGroup» UML Class
Grouping of Elements to form a complex type	XML xs:group	Stereotyped «group» UML Class
Acceptable Values for Elements	XML constraints via minOccurs, maxOccurs	Stereotyped «constraint» class member
Indirect Reference of Elements	XML ref	Stereotyped «ref» name class member
Parent – child relationship of non-named Elements	XML Element – non-named child element	UML Directed Association Relationship

Table 4.1: Specialized UML Profile for Tree-Structured Document to DSCD with XML Cases.

To illustrate the process of creating a DSCD that transitions from an XML diagram, consider the DSCD shown in Figure 4.3 has been created from the HL7 CDA schema segment from Figure 2.4, which we repeat in Figure 4.2. To create Figure 4.3, we utilize the equivalences from the tree-document structure and XML that then aligns the various XML concepts (column two of Figure 4.1) in order to map them to the appropriate UML profile concepts (column three of Figure 4.1). To begin, consider lines 1, 2, and 28 of the

XML in Figure 4.2 which correspond to a set of UML constructs (A/B, C, and D in Figure 4.3) that will be generated from the XML source. Specifically, these three lines of XML cause the creation of three UML classes corresponding to the three labeled boxes A/B, C and D in Figure 4.3 that are inside the dashed box at the top of the figure. First, from line 1 in Figure 4.2, we have `<xsd:element name="clinical_document_header">` where the element being defined in the XML schema maps to a UML class with the same element name that acts as the root of the DSCD (see UML class next to A in Figure 4.3). This mapping uses rows 1 and 2 of Table 4.1 to generate a class and its class name `clinical_document_header`. Second, notice that in the XML, the `xsd:complexType` in lines 2 to 33 (open/close) in Figure 4.2 is composed of a `xsd:sequence` (lines 3 to 27 in Figure 4.2), an `xsd:attributeGroup` (line 28 in Figure 4.2), and three `xsd:attribute` (lines 29-32 in Figure 4.2. The `xsd:complexType` of XML CDA in lines 2 to 33 (open/close) causes the generation of a UML «complexType» stereotyped generic class (see B in Figure 4.3), connected with a direct association to the root UML class (rows 4, 5, 7 and 11 of Table 4.1, shown in A/B of Figure 4.3). Once the «complexType» has been generated, it is then possible to create its components. The `xsd:sequence` for lines 3 to 27 causes the generation of the UML class next to C in Figure 4.3 that creates a sequence (list) that contains all of the refs given on lines 4 to 26 generated as UML «elements» as captured within the dotted box at the bottom of Figure 4.3 (see F). These UML «elements» are stereotyped with respect to the attributes where the XML ref is represented as a «ref» stereotype and constraints are represented as a «constraint» stereotype, as shown in row 10 of Table 4.1. Similarly, the `xsd:attributeGroup` in line 28 causes the generation of the UML construct next to D in

Figure 4.3 that has the name of "`common_atts`" from line 28 of Figure 4.2. Finally, the last portion of the «complexType» is generated corresponding to the three attributes: `HL7-Name`, `T`, and `RIM-Version` (lines 29, 31, and 32, respectively) resulting in the generation of E in the top right portion of Figure 4.3.

```
1.  <xsd:element name="clinical_document_header">
2.   <xsd:complexType>
3.   <xsd:sequence>
4.   <xsd:element ref="id"/>
5.   <xsd:element ref="set_id" minOccurs="0"/>
6.   <xsd:element ref="version_nbr" minOccurs="0"/>
7.   <xsd:element ref="document_type_cd"/>
8.   <xsd:element ref="service_tmr" minOccurs="0"/>
9.   <xsd:element ref="origination_dttm"/>
10.  <xsd:element ref="copy_dttm" minOccurs="0"/>
11.  <xsd:element ref="confidentiality_cd" minOccurs="0" maxOccurs="unbounded"/>
12.  <xsd:element ref="document_relationship" minOccurs="0"
    maxOccurs="unbounded"/>
13.  <xsd:element ref="fulfills_order" minOccurs="0"/>
14.  <xsd:element ref="patient_encounter" minOccurs="0"/>
15.  <xsd:element ref="authenticator" minOccurs="0" maxOccurs="unbounded"/>
16.  <xsd:element ref="legal_authenticator" minOccurs="0"/>
17.  <xsd:element ref="intended_recipient" minOccurs="0" maxOccurs="unbounded"/>
18.  <xsd:element ref="originator" minOccurs="0" maxOccurs="unbounded"/>
19.  <xsd:element ref="originating_organization" minOccurs="0"/>
20.  <xsd:element ref="transcriptionist" minOccurs="0"/>
21.  <xsd:element ref="provider" maxOccurs="unbounded"/>
22.  <xsd:element ref="service_actor" minOccurs="0" maxOccurs="unbounded"/>
23.  <xsd:element ref="patient"/>
24.  <xsd:element ref="originating_device" minOccurs="0" maxOccurs="unbounded"/>
25.  <xsd:element ref="service_target" minOccurs="0" maxOccurs="unbounded"/>
26.  <xsd:element ref="local_header" minOccurs="0" maxOccurs="unbounded"/>
27.  </xsd:sequence>
28.  <xsd:attributeGroup ref="common_atts"/>
29.  <xsd:attribute name="HL7-NAME" type="xsd:string"
30.  fixed="doc_serv_as_clin_doc_header"/>
31.  <xsd:attribute name="T" type="xsd:string" fixed="service"/>
32.  <xsd:attribute name="RIM-VERSION" type="xsd:string" fixed="0.98"/>
33.  </xsd:complexType>
```

Figure 4.2: HL7 CDA 'clinical_document_header' Schema Segment.

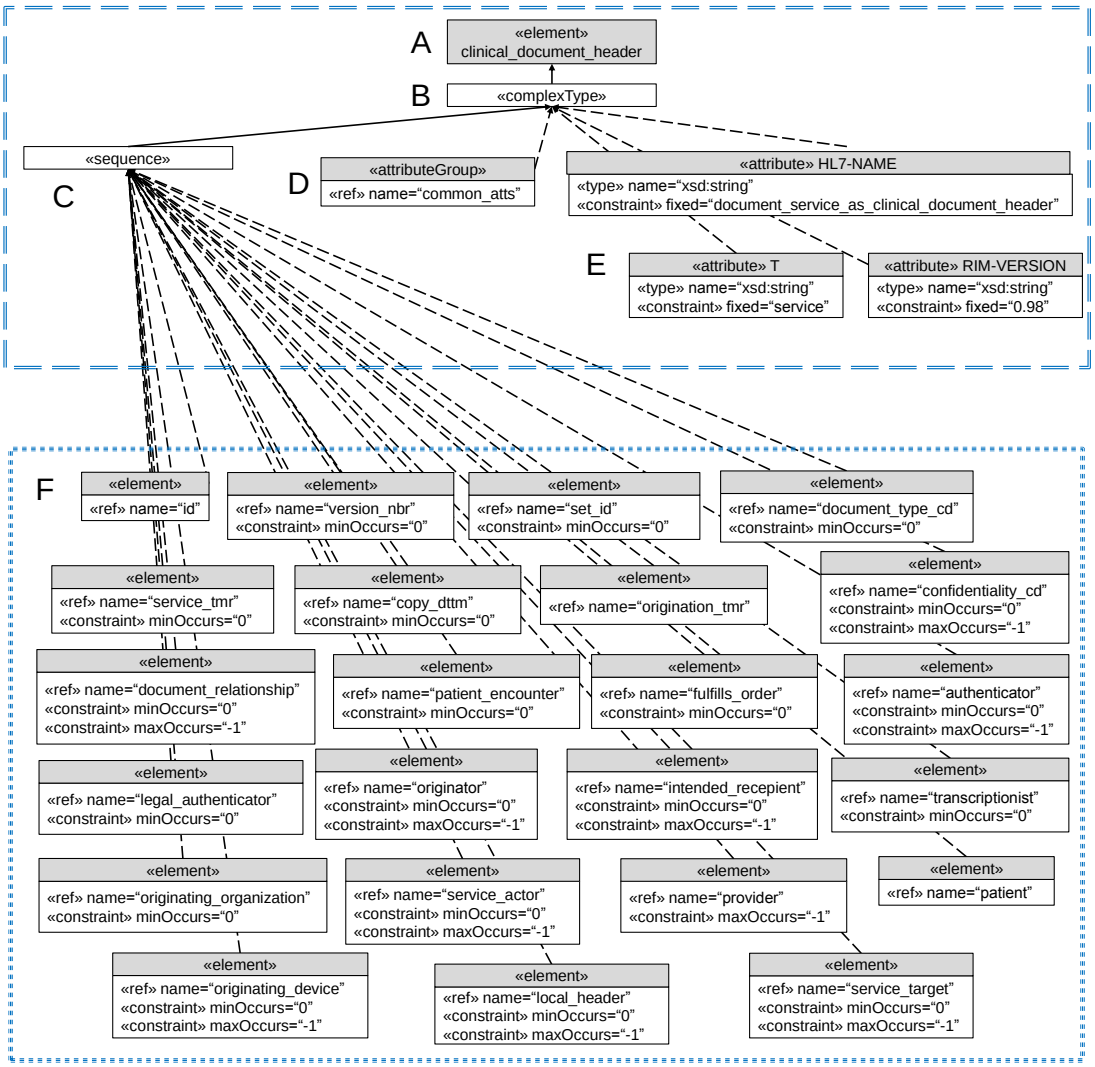

Figure 4.3: A DSCD for the CDA Segment.

The conversion of the CCR schema from Figure 2.5 as repeated in Figure 4.4 follows a similar process in conversion to the CDA example, with Figure 4.5 showing the first portion of the CCR schema as a DSCD after applying the mapping process from Table 4.1. As with the prior example, the `ContinuityOfCareRecord` root (see row 3 of Table 4.1 and line 1 of Figure 4.4) contains a `xsd:complexType` that contains a sequence which in turn contains a set of elements `CCRDocumentID` (line 4), `Language` (line 5), `Version` (line 6), `DataTime` (line 7), `Patient` (lines 8 to 14), and `Body` (lines 16 to 64 and following – no shown). In converting these to UML as shown in Figure 4.5, the

`ContinuityOfCareRecord` as given in the dashed top portion of the figures is represented as a «complexType» that contains a «sequence» as given in A, which in turn contains four simple attributes that are represented as «element» stereotyped UML classes as given in B, a `Patient` that has in turn a «complexType», «sequence», and «element» with an `ActorID` as given in C, and finally an «element» `Body` that is a «complexType», and a «sequence» as given in D of multiple «element»s as given in E. The `Body` subtree of the XML document is composed of subtrees for: `Payers` (line 19 of Figure 4.4), `AdvancedDirectives` (line 26 of Figure 4.4), `Support` (line 34 of Figure 4.4), `FunctionalStatus` (line 42 of Figure 4.4), `Problems` (line 49 of Figure 4.4), and `FamilyHistory` (line 56 of Figure 4.4); these are shown in the bottom dotted area of Figure 4.5). These XML subtrees are mapped to the corresponding UML diagrams in E of Figure 4.5 in the bottom dashed box, where each of the «element» classes are translated to UML diagrams that contain a «element» that is composed of a «complexType» with a «sequence», that has an «element». As an example, the XML element that has a `AdvancedDirectives` (line 26 of Figure 4.4) acts as the root of the subtree right after the «sequence» stereotyped UML class (shown in the F in the middle of Figure 4.5 following row 6 of Table 4.1). The constraint of `AdvancedDirectives`, which is an XML `minOccurs` constraint (line 26 of Figure 4.4), is represented as a stereotyped member «constraint» `minOccurs="0"` of the UML class. The direct children, the `complexType` (line 27 of Figure 4.4) and `sequence` (line 28 of Figure 4.4) elements in the XML schema are represented as directed association UML classes. The leaf node, the `AdvancedDirective` element (line 29 of Figure 4.4), is represented as an «element»

```
1.  <xs:element name="ContinuityOfCareRecord">
2.   <xs:complexType>
3.   <xs:sequence>
4.   <xs:element name="CCRDocumentObjectID" type="xs:string"/>
5.   <xs:element name="Language" type="CodedDescriptionType"/>
6.   <xs:element name="Version" type="xs:string"/>
7.   <xs:element name="DateTime" type="DateTimeType"/>
8.   <xs:element name="Patient" maxOccurs="2">
9.   <xs:complexType>
10.  <xs:sequence>
11.  <xs:element name="ActorID" type="xs:string"/>
12.  </xs:sequence>
13.  </xs:complexType>
14.  </xs:element>
15.  …
16.  <xs:element name="Body">
17.  <xs:complexType>
18.  <xs:sequence>
19.  <xs:element name="Payers" minOccurs="0">
20.  <xs:complexType>
21.  <xs:sequence>
22.  <xs:element name="Payer" type="InsuranceType" maxOccurs="unbounded"/>
23.  </xs:sequence>
24.  </xs:complexType>
25.  </xs:element>
26.  <xs:element name="AdvanceDirectives" minOccurs="0">
27.  <xs:complexType>
28.  <xs:sequence>
29.  <xs:element name="AdvanceDirective" type="CCRCodedDataObjectType"
30.  maxOccurs="unbounded"/>
31.  </xs:sequence>
32.  </xs:complexType>
33.  </xs:element>
34.  <xs:element name="Support" minOccurs="0">
35.  <xs:complexType>
36.  <xs:sequence>
37.  <xs:element name="SupportProvider" type="ActorReferenceType"
38.  maxOccurs="unbounded"/>
39.  </xs:sequence>
40.  </xs:complexType>
41.  </xs:element>
42.  <xs:element name="FunctionalStatus" minOccurs="0">
43.  <xs:complexType>
44.  <xs:sequence>
45.  <xs:element name="Function" type="FunctionType" maxOccurs="unbounded"/>
46.  </xs:sequence>
47.  </xs:complexType>
48.  </xs:element>
49.  <xs:element name="Problems" minOccurs="0">
50.  <xs:complexType>
51.  <xs:sequence>
52.  <xs:element name="Problem" type="ProblemType" maxOccurs="unbounded"/>
53.  </xs:sequence>
54.  </xs:complexType>
55.  </xs:element>
56.  <xs:element name="FamilyHistory" minOccurs="0">
57.  <xs:complexType>
58.  <xs:sequence>
59.  <xs:element name="FamilyProblemHistory" type="FamilyHistoryType"
60.  maxOccurs="unbounded"/>
61.  </xs:sequence>
62.  </xs:complexType>
63.  </xs:element>
64.  …
```

Figure 4.4: CCR – Continuity of Care Record Schema Segment.

stereotyped UML class with the `minOccurs` constraint (line 30 of Figure 4.4) as a stereotyped «constraint» member (row 9 of Table 4.1).

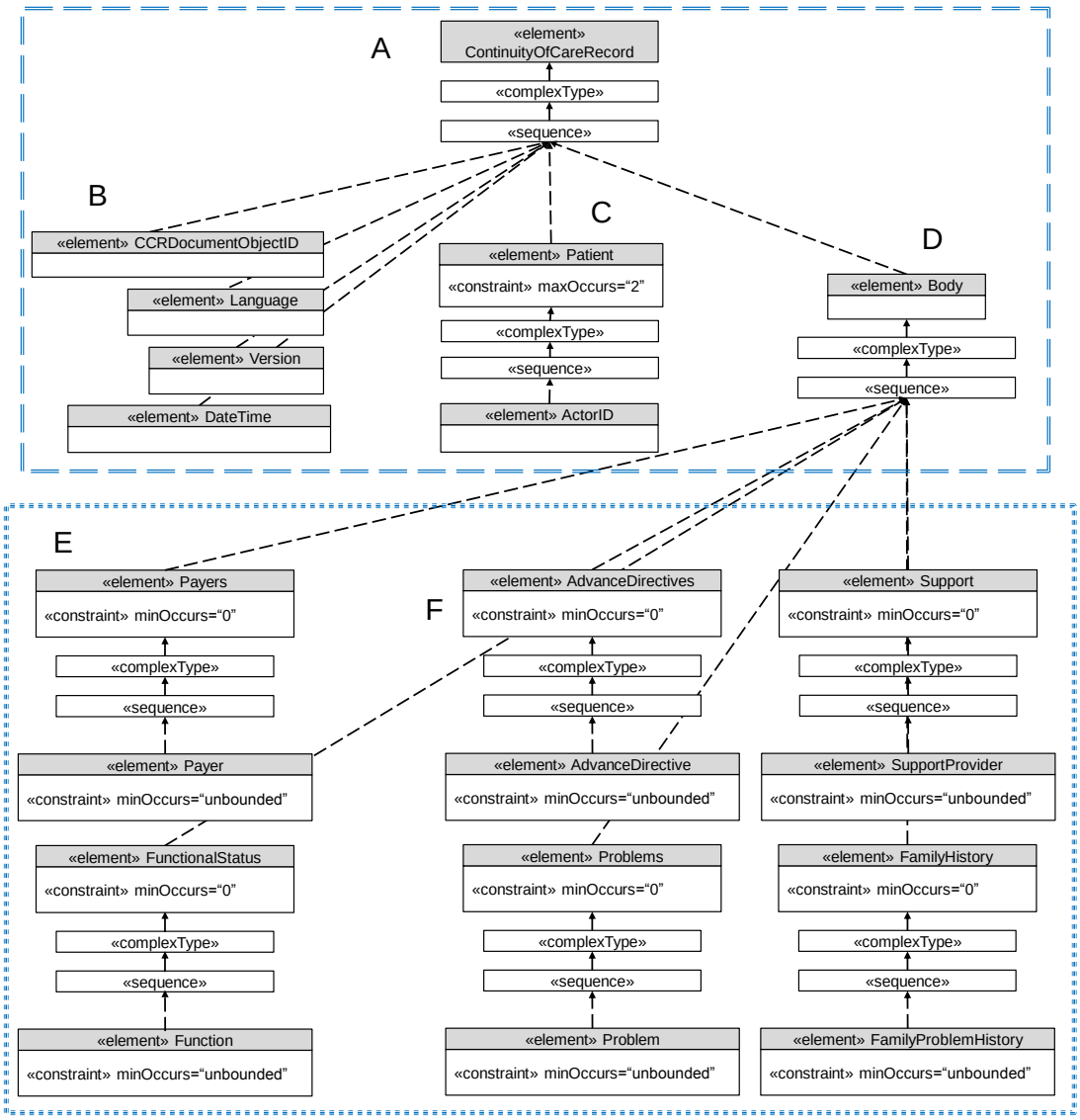

Figure 4.5: A DSCD for the CCR Segment.

4.2.2 The Secure Information Diagram (SID)

The DSCD is utilized to provide input for the definition and construction of the *Secure Information Diagram (SID),* which represents: the different portions of the DSCD that need to be secured based on the overall security requirements for an application. Essentially, the intent is to identify a subtree of the DSCD that must be secured for access

by users. In the case of our sample DSCDs in Figures 4.3 and 4.5, a subtree and/or a set of elements will be identified that must be securely controlled. The SID utilizes extensions to the UML metamodel (M2-layer) in order to represent those portions of the DSCD that need to be securely controlled. Graphically, at the model layer of the MOF (M1), the SID is an UML package with the stereotype «SecureInformation» that contains all of the respective classes of elements from the schema to be secured. The SID acts as the visual representation of the projection operation (see Definition 8c in Section 3.2) over the schema and elements (see Defs. 1 and 2 of Section 3.1). The effective subset of elements from the schema that result from the projection over the original schema's elements is visually represented with the SID. For example, consider the left side of Figure 4.6, which shows of a SID with CDA's elements (e.g., `clinical_document_header` and `section` in Figure 4.3) as classes while the right side of Figure 4.6 has an example of a SID with the CCR elements (e.g., `Body`, `Patient`, `Problems`, `FamilyHistory` in Figure 4.5) with some of the elements organized as a subtree of the original schema (`Body`, `Problems`, `FamilyHistory`).

The security administrator that is in charge of designing the enforcement policies of the clinic in the healthcare scenario of Section 2.6 has the responsibility to identify the elements of a schema that need security control in terms of projecting their access (read, aggregate, insert, update, and delete). This can include from Figure 4.2 and the CDA elements such as `patient`, `patient_encounter`, and `originating_organization`, and from Figure 4.4 and the CCR elements such as the `Patient`, `Body`, `Problems` and `FamilyHistory` with the last three a subtree The security administrator will interact with all of the application's stakeholders in order to identify those portions of the application

A's schemas (see Definition 5 in Section 3.1) that need to be securely controlled. Once these have been identified, the security administrator can perform a projection operation (Definition 8c from Section 3.2) over the elements that need some level of security, to specifically select the set of elements from the DSCD that will comprise the SID package. As a result, the SID elements are taken from across all of the schemas of the application A. In fact, elements may be structured is packages that form a subtree as shown in the right side of Figure 4.6 for CCR. These elements might need RBAC permission defined over them in order to control destructive (insert, update, or delete) or non-destructive (read or aggregate) operations (Definitions 15 to 17 from Section 3.3), or, decorated with classification levels from LBAC (Definitions 9 and 24 to 28 from Section 3.4) to provide a more complete security policy, which will be further discussed in Section 4.2.4 with the LBAC SID. Note that the process of creating SID is iterative in nature, and the security administrator needs to arrive a preliminary version and constantly revisit the content of an SID as work on defining roles, permissions, privileges, and users are being defined and modified over time. There is also a need to revisit SID whenever there are any substantial changes (additions, deletions, or modifications) to the schemas for an application.

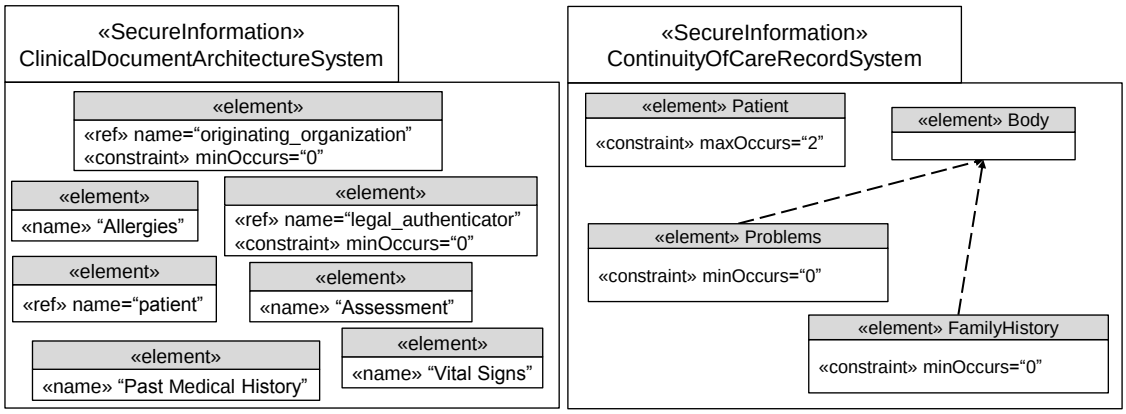

Figure 4.6: SID with CDA Elements (left) and CCR Elements (right).

4.2.3 The Document Role Slice Diagram (DRSD)

To support RBAC of operations that target schemas and their instances and to enable granular LBAC labeling of elements, it is necessary to provide a construct that security administrators can utilize to describe policies and segments of policies. Towards this purpose, we present the *Document Role Slice Diagram (DRSD)* that organizes the roles (see Definitions 11 and 12 from Section 3.3) into a hierarchy Graphically, the DRSD is an UML package with the stereotype «DRSD» with class diagrams as members that represent each of the elements of the schema (e.g., `xs:complexType`, `xs:element`, `xs:sequence`, etc. for XML) that need to be secured. These elements appear in the DSCD for the CDA schema as given in Figure 4.3. These diagram segments, which are organized hierarchically depending on their position with respect to the schema tree, contain stereotypes that represent the operations permitted by the role being described. For example, the middle-left of Figure 4.7 shows the role slice for a Nurse with respect to classes of the SID (see Figure 4.6) obtained by projecting the original XML schema of the CDA (see Figure 4.2). The diagram in the DRSD package contains the stereotype «read» to dictate that the Nurse role can read the elements `legal_authenticator`, `patient` and `originating_organization` from the `clinical_document_header` subtree. The Nurse role can also read captions with the name values `Allergies`, `Assessment` and `Past Medical History`, while the same role can insert new `Vital Signs` and update `Vital Signs`, and aggregate `Vital Signs`.

Once the security administrator in charge of designing the security for the healthcare scenario of Section of 2.6 has identified the elements that require security in creating the SID (see Figure 4.6 in Section 2.2), the next step in the process is to define which roles in

the application can perform which actions (read, aggregate, insert, update, and delete), establishing the permissions (see Definitions 16, 17, 18 from Section 3.3). Towards this objective, the security administrator designs the DRSDs on a role-by-role basis, defining in a granular document level which operations can be performed on which elements and by which role against the DSCD for CDA in Figure 4.3. These roles can also be organized into a hierarchy with MedicalProvider as the parent role of Nurse, Physician, and Psychiatrist and Staff as a separate role; this will allow permissions shared by all roles to be defined at the parent MedicalProvider as is shown in Figure 4.7 so that they are not unnecessarily replicated which improves the maintainability of the security policy. For example, the permissions for the MedicalProvider role (top of Figure 4.7) includes read access to the `patient`, `Allergies`, `Assessment`, `Past Medical History`, `Vital Signs`, `originating_organization`, and `legal_authenticator` elements of the CDA. At the same time, the security administrator might define for the Nurse role, permissions to read the current `prescriptions` and `laboratorytests` (Definition 16 of Section 3.3), while both the Physician and Psychiatrist can write both of those items, with the Psychiatrist having the additional capability to write `psychhistory`. Repeating this process over all of the roles in the system will result in the creation of several DRSDs, one per role, for the application that defines the RBAC requirements for the application as a whole. Like the process of creating the SID as from Section 4.2.2, the process of creating the DRSD is iterative. Therefore, refinement over the permissions assigned to a role is possible by performing the design process as many times as desired. This allows the possible changes over policy that affects one role without incurring in a high cost of modification. This also allows for new roles to be defined as requirements change across

the stakeholders of an application. For example, after meeting to discuss the security permissions defined over the roles of the healthcare scenario of Section 2.6, the decision of adding the permission for Nurses to delete `Vital Signs` in case of input error is agreed upon. This change could only require a refinement process over the Nurse DRSD (Figure 4.7) that would include the «delete» stereotype over the access() method of the `Vital Signs` element class. Note that while we have not shown a DRDS for the CCR schema, it is easy to imagine having a similar set of roles with the elements now referring to the DSCD in Figure 4.5.

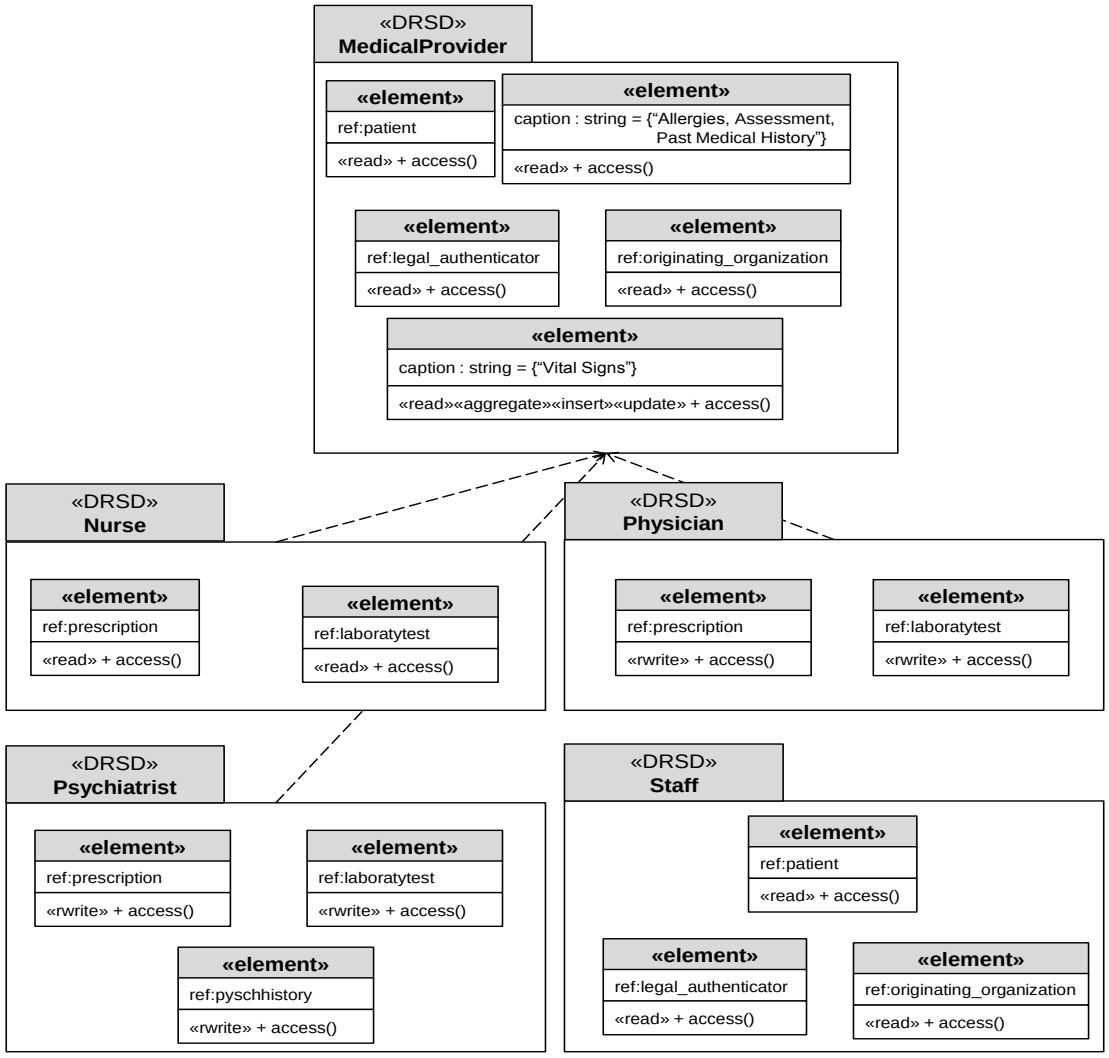

Figure 4.7: DRSD Role Hierarchy for CDA.

4.2.4 The LBAC Secure Information Diagram (LSID)

The *LBAC Secure Information Diagram* (*LSID*) is a UML package with the stereotype «SecureInformation» that decorates the SID (see Section 4.2.2) and contains all of the respective classes of elements from the schema to be secured per access modes (ams) and classifications (cls). In the left of Figure 4.8 for the CDA segment's example, is an LSID where each element would have several attributes that indicate the access mode (denoted inside the element class with *am*) and the classification level (denoted with security level or *cls*); in this case, for each element, there is a *cls* associated with respect to the access mode. For example, the `patient` element has a *cls* of unclassified when considering operations that have an access mode of read, while the same element has a classification of classified when considering operations that have an access mode of write. The right of Figure 4.8 shows the example of an LSID for the CCR schema segment that is a subset of the DSCD in Figure 4.5. This LSID denotes a classified classification to the `Patient`, `Problems` and `FamilyHistory` elements of the DSCD. Note that in practice, the LSID works off an initial SID that has been established by the security administrator during the initial design process. While the SID is a project of the original schemas, the LSID decorates this projection thereby supporting decoration as detailed in Definitions 27 and 28 in Section 3.4. Like SID, LSID may have packages corresponding to subtrees of schemas.

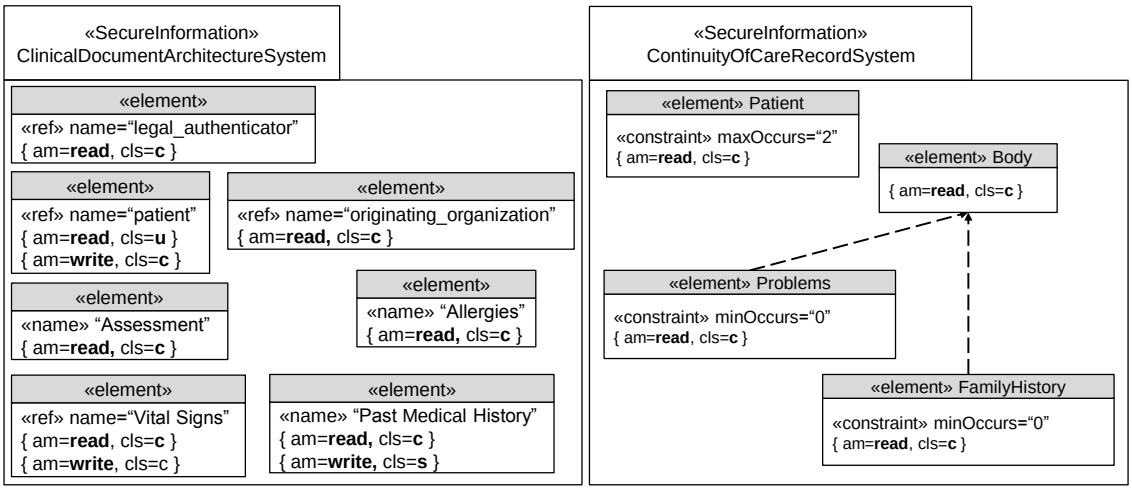

Figure 4.8: LSID for CDA (left) and CCR (right).

The LSID permissions via decoration can be established before or after RBAC has been, or in fact, can be in lieu of RBAC if the application only has LBAC. In either case, whether RBAC has been defined or not, the security administrator is in charge of designing the enforcement policies to utilize the capabilities of LBAC by assigning clearance levels to users (Definition 6 from Section 3.4) and classification levels to elements (Definition 27 from Section 3.4) of the CDA or CCR schemas. More importantly, the security administrator must decorate the defined SID to create an LSID with access modes and sensitivity levels. Following the example of Figure 4.8, this can include for CDA elements in the DSCD of Figure 4.3 that includes `patient`, `Vital Signs`, and `originating_organization` or for CCR elements in the DSCD of Figure 4.5 that includes `Patient` and a subtree of `Body`, `Problems` and `FamilyHistory`; both access modes and sensitivity levels have been defined in Figure 4.8. In the same manner as with the SID, the security administrator will interact with the relevant stakeholders to determine which access modes and classification levels are assigned to which elements and which clearance levels will be assigned to which users (this will be later defined in Section 4.2.7) in arriving at the LSID. The security administrator may also further refine

the SID to an SID' if for some reason there is a wish to further restrict the information to control for LBAC. Once the classification levels for elements have been defined, the security administrator can then perform a decoration operation (Definition 9 from Section 3.2) over the elements defined in order to extend them with classification (sensitivity) levels. As a result, the SID elements are classified with LBAC levels and thereby create the LSID. In the same manner as with the SID and the DRSD, the process of creating the LSID is iterative and open to refinements in case the security requirements constantly change. Like the earlier diagrams (SID and DRSD), the security administrator will likely pursue an iterative design process in order to arrive at a final LSID and this may involve making changes to the original SID.

4.2.5 The User Diagram (UD)

Another major component of information security to support RBAC, LBAC, and DAC is to quantify different users of the system, their requirements, and their constraints in order define the users of the system whose information is to be secured. The interplay of users, their roles and delegation permissions (for RBAC), their clearance levels (for LBAC), and their authorization permissions (DAC) require the proper definition of a user concept. The work in secure software engineering (Pavlich-Mariscal et al., 2010) proposed a UML extension for users via a User Diagram. In this dissertation, we build upon this first iteration of the User Diagram to extend to include both LBAC and RBAC user features directly to the metamodel. Graphically, the *User Diagram (UD)* is a UML package with the stereotype «User». User-role assignments, which are part of RBAC (see Definition 6 and its revisions throughout Chapter 3) are a directional arrow tagged with the «RoleAssignment» stereotype. Clearance levels, in support of LBAC, are represented

with a directional arrow tagged with the «CLRAssignment». Another capability of the UD is the representation of separation of duty constraints (SoD) with an n-ary association tagged with the «SOD» stereotype. These separations of duties connect all of the roles that have a separation of duty relation. Lastly, the UD can also represent the mutual exclusion relations between assigned roles via the use of the «ME» stereotype. Figure 4.9 follows the healthcare scenario of Section 2.6 for UD with the user named Elisa assigned a role of Physician and the clearance level of secret (s). That role has a separation of duty with the roles Nurse that is assigned to the users Leroy and Jenkins, in which Leroy has a clearance level of classified and Jenkins has a clearance level of secret, and Psychiatrist, which is assigned to the user Brock with clearance level of top-secret.

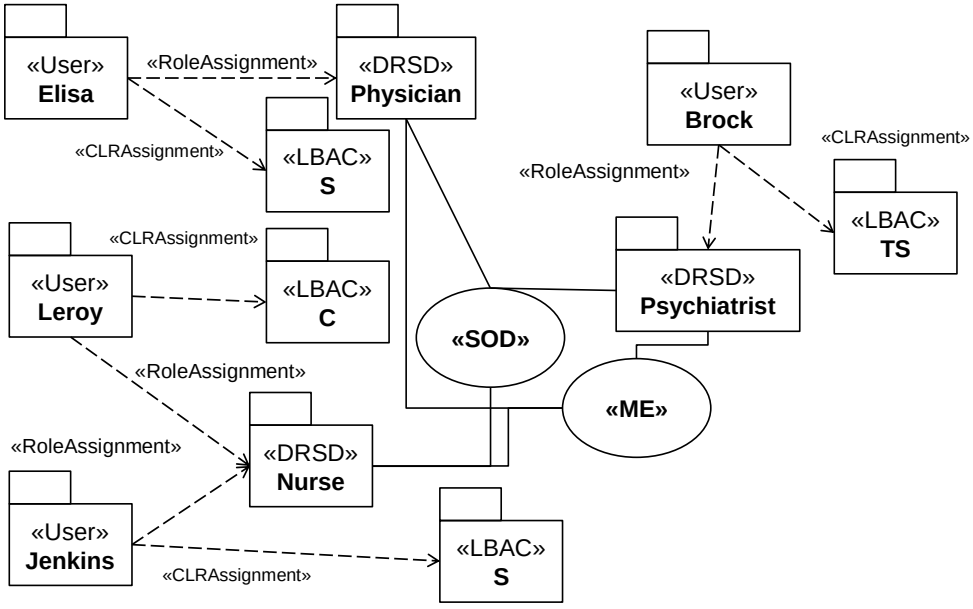

Figure 4.9: UD for the Healthcare Scenario.

The security administrator can bring together all the previous security capabilities that cover RBAC, LBAC and DAC that include DSCD, SID, DRSD, LSID, and DD via the creation of the UD as shown in Figure 4.9. The security administrator has the responsibility of ensuring that clearance levels are assigned to users (Definition 6 of

Section 3.4), their roles (Definition 6 of Section 3.3), and the separation of duty and mutual exclusion between roles (Definition 6 of Section 3.3). As a result, the UD of an application would graphically denote the interplay of users, their roles, and their clearances (if needed) while also showing how separation of duty and mutual exclusion is organized. This diagram is also iterative in its creation, and after deployment will be managed by a security office who handles day-to-day updates and adjustments to this diagram. This allows the security office to add, update or delete users and their credentials as many times as needed. If there is a need to revisit the UD due to changes in which roles are assigned to who, or which clearance level is now the one assigned to the user, the security office can refine the end-requirements without obstacles particularly in a rapidly changing environment like healthcare.

4.2.6 The Delegation Diagram (DD)

The Delegation Diagram (DD) captures the information of the security model's delegation mechanisms as a new UML diagram extension and is meant to capture the concepts of original user, role assigned, delegated users, and role delegation per Definitions 31 to 35 in Section 3.5. This includes the delegable users that can receive a role delegation from the original user/role pair. Figure 4.10 illustrates a delegation diagram for our continuing example with the healthcare scenario from Section 2.6 where the user Elisa who has the role Physician assigned (left side of the figure) is interested in delegation that role to one or more other users. The role Delegation Diagram on the right side of Figure 4.10 is a UML package tagged with the «DelegationDiagram» stereotype. The User Diagram (left side of Figure 4.10 and within the Delegations class) contains information on the user and role and are part of both the delegating user and the delegable

user(s) who are capable of receiving the role as part of a delegation. The connection between a user and the delegations is done with a directional connection tagged with the «Delegation» stereotype. In Figure 4.10, the user Elisa with the role of Physician can delegate the Physician role to Samantha or Emily.

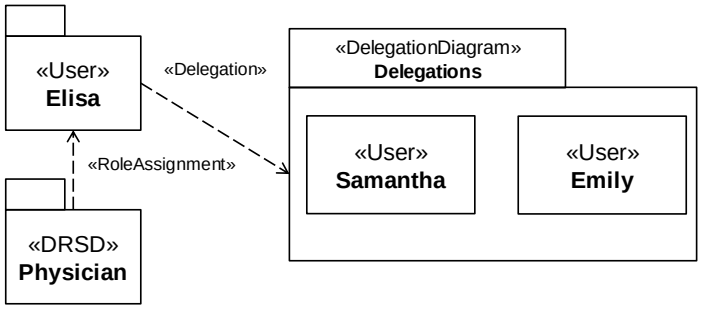

Figure 4.10: DD for User Elisa in Physician Role in a Healthcare Scenario.

The security administrator that is in charge of the healthcare scenario of Section 2.6 also has the responsibility of defining permissions over user-role delegation to other users which includes the delegation from an original to a delegated user. What is not directly supported in the DD is the pass-on-delegation capability that allows Elisa to pass on the authority to delegate Physcian to a user (say Samantha) who in turn would be able to further delegate Physician to another user; this is supported in Definition 34 in Section 3.5. In Figure 4.10 Elisa defines the possibility to delegate her Physician role to potential recipients Samantha and Emily. The security administrator would define the DelegationDiagram package that would have the delegable users (Definition 33 of Section 3.5) inside, while the original user (Definition 32 of Section 3.5) would have the respective role assigned with a Delegation tagged relationship with the package. While the security administrator will define the delegation, the user Elisa has the ability to initiate the delegation. For example, Elisa might delegate to Emily on a Friday night since Emily is the covering physician and will be answering calls from Elisa's patients. Like all

the constructs discussed in the previous sections, the process of creating the DD is iterative and can be refined as many times as needed by the security requirements of the application. Note also that if Elisa has multiple roles assigned, then there will be separate DDs for each user/role combination to establish the delegations that are allowed for all roles Elisa can delegate.

4.2.7 Authorization Diagram (AD)

The *Authorization Diagram (AD)* is a new UML diagram that illustrates a particular user/role combination and the way that it is connected to authorizations to particular schemas and/or their instances for a given application. Authorizations are used to augment security by providing another layer of verification. For example, if a user has permissions defined over a specific schema, but is not authorized to it, then that user cannot perform any of the permissions. Further, a user may have permission to access a particular schema but have no assigned instances as yet; e.g., a Elisa with the Physician role is a new doctor that doesn't have any patients. Figure 4.11 illustrates the structure of an AD which is a UML package tagged with the «AuthorizationDiagram» stereotype for user Elisa under the role of Physician (left hand side of the figure). Classes on the right hand side of Figure 4.11 represent those specific schemas and instances that have been authorized. In the case of the schemas, a simplified version of the DSCD is used. For instances of the schema, placeholder classes that serve as identifiers are utilized tagged with the «Instance» stereotype. In Figure 4.11, Elisa has access to the CCR schemas as a DSCD for CCR as given in Figure 4.5, as well as two CCR instances «has» for Carol Smith and John Jones. The connection between a user and the authorizations is done with

a directional connection tagged with the «Authorization» stereotype. Note that while not shown, in practice, there can be multiple DSCDs with associated instances in each AD.

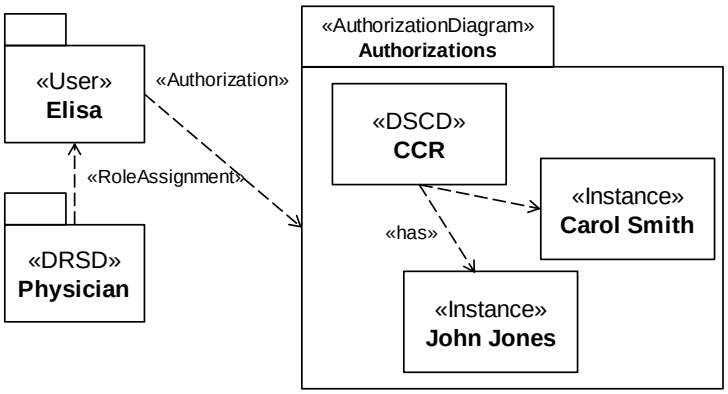

Figure 4.11: AD for User Elisa in a Healthcare Scenario.

Like the case with the DD, the security administrator also has the responsibility of defining and maintaining authorizations over user/role combinations and the schemas and/or instances which they have been authorized to use, e.g., the patients each physician is treating. This includes the capabilities of, as shown in Figure 4.11, a user such as Elisa being authorized to specific schemas (CCR) and/or instances (Carol Smith and John Jones) while performing a specific role, such as that of Physician. The security administrator would define the AuthorizationDiagram package that would have the users/role and the authorized schemas (Definition 35 of Section 3.6) and instances (Definition 36 of Section 3.6) inside the package. And like the DD, the AD creation process is iterative and capable of handling refinements over new security requirements and definitions. Moreover, as with UD, the day-to-day usage of the AD is taken over by a security officer that would be in charge of adjusting user/role schema/instance permissions on a continuous basis. This may actually be a person that is in charge of a unit in a medical establishment (e.g., a nurse manager) that would be in charge of authorizing which patients have been assigned to each user with the nurse role.

4.3 The UML Metamodel for Security Extensions

The underlying model from Chapter 3 that is realized in UML extensions on Section 4.2 sets requirements that are engineered at the MOF M2 metamodel layer (see Figure 2.8 from Section 2.7). That is, all the new UML constructs presented in Section 4.2, which are defined at the MOF M1 layer, require a corresponding metamodel extension at the M2 layer. Towards this purpose, this section covers the UML metamodel extensions that support those extensions presented in Section 4.2. To preface the discussion, we provide in Figure 4.12 a high-level view of our MOF M2 metamodel for the new UML diagrams, demonstrating the way that all of the new constructs interact to allow the modeling and security definition for a set of schemas composed of tree-structure documents as we have illustrated with XML. As shown in Figure 4.12, the meta-classes SecureInformation, Element, AccessMode, Permission, SensitivityLevel, User, UserRole, RoleSlice, OriginalUser, DelegableUser, Authorization, Instance, Schema, ME, and SOD extend the MOF M2 layer to support RBAC, LBAC, and DAC capabilities as defined in Chapter 3 and as realized with the new UML diagrams in Section 4.2. Extending the MOF M2 layer metamodel with these new constructs makes it possible to define the diagrams at a model layer, and then instantiate their definitions to generate security policies that will be further discussed in Chapter 5. The top-left of Figure 4.12 graphically represents the section of the metamodel that pertains to the SID diagram from Section 4.2.2. The top of Figure 4.12 contains the meta-classes the build the metamodel extensions for the SID with LBAC as discussed in Section 4.2.4. In turn, the middle-right of Figure 4.12 shows the meta-classes that make the Delegation Diagram from 4.2.6 possible. Moving to the bottom of Figure 4.12, Mutual Exclusion and Separation of Duty meta-classes are shared

by the Role Slice for the purposes of the User Diagram from Section 4.2.5. Lastly, the Authorization Diagram from Section 4.2.7 has its meta-classes shown in the bottom-left of Figure 4.12. The remainder of this section presents the MOF M2 metamodel for all of the new UML diagrams in Section 4.2 in greater detail.

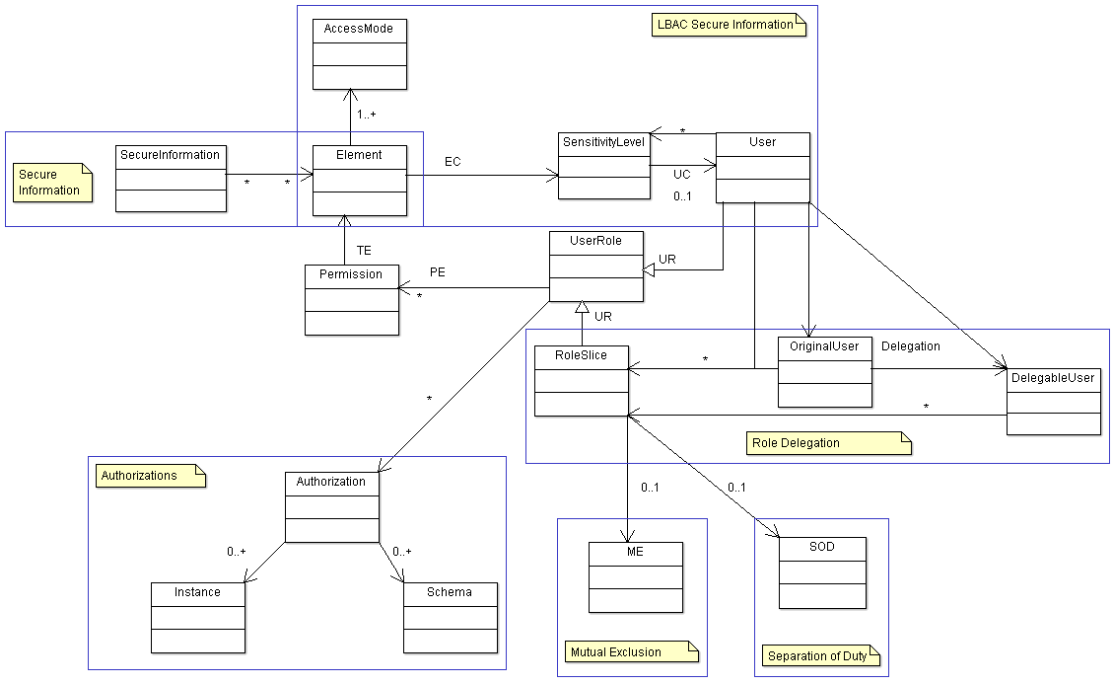

Figure 4.12: Tree-Structured Document Security UML Metamodel at MOF M2.

The DSCDs (schemas) for an application must be constrained to identify those portions of the schema that require security control. This was accomplished in as discussed in Section 4.2.2 with the Secure Information Diagram (SID) in Figure 4.6 that identified those portions (elements and subtrees) of an application's schema on which both RBAC permissions and LBAC classifications will be defined. For SID, the M2 metamodel is shown in Figure 4.13 where each class that is part of the SID is represented as meta-class (SecureInformation) associated with many possible instances of any given schema element as represented with the Element meta-class.

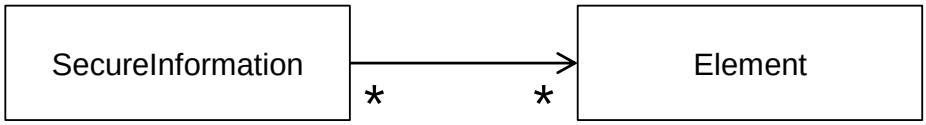

Figure 4.13: Secure Information Diagram M2 metamodel.

Next, in Figure 4.14, the metamodel representation of the document role-slice diagram (DRSD) is shown as discussed in Section 4.2.3 and shown in Figure 4.7, where the RoleSlice meta-class represents the role slices that will be defined with permissions against the SID with respect to the schema(s) of the application to be secured. The Permission meta-class represents the permissions allowed over the instances validated against the secured schema (read, aggregate, insert, update, delete) that define what a role can and can't do for the elements in a schema. In order to create a relation between the RoleSlice meta-class (which contains all of the DRSD instances) and the Permission meta-class (which contains all of the schema targeting permissions), it is necessary to create a relation between the users and their roles. In Figure 4.14, the UserRole meta-class is a parent-class of the RoleSlice meta-class and a sibling class of the Permission meta-class. The connections between the UserRole and Permission meta-classes are given by the permitted permission (PP) relation. The Element meta-class in turn represents all of the instances of elements (from the schema) that are targeted by the different permissions. This connection is tagged with the targeted element (TE) label in Figure 4.14.

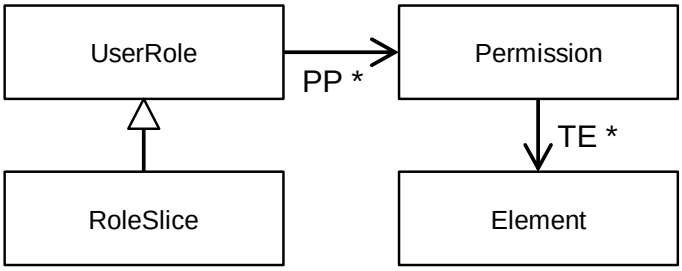

Figure 4.14: Document Role Slice Diagram M2 metamodel.

Next, the meta-model for the LBAC Secure Information Diagram (LSID) as discussed in Section 4.2.4 and shown in Figure 4.8 is shown in Figure 4.15. The four meta-classes User, AccessMode, Element, and SensitivityLevel are interconnected to represent the relations between the user (User), its clearance level (Sensitivity), and access modes (read, aggregate, insert, update, delete; AccessMode) for each of the elements (xs:element, xs:complexTyp, etc.; Element) from the SID that need to be protected. To represent the relation between User and SensitivityLevel, an arrow with a UC (user clearance) tag is used. This relation indicates that the user could either have a clearance level or is without a clearance level, therefore the utilization of the 0..1 cardinality constraint. Element and Sensitivity are similarly related, represented with the arrow tagged EC (element classification). The relationship between Element and AccessMode is represented with the 1..+ cardinality constraint to cover the case of an element with different possible access modes (e.g. patient element from Figure 4.8).

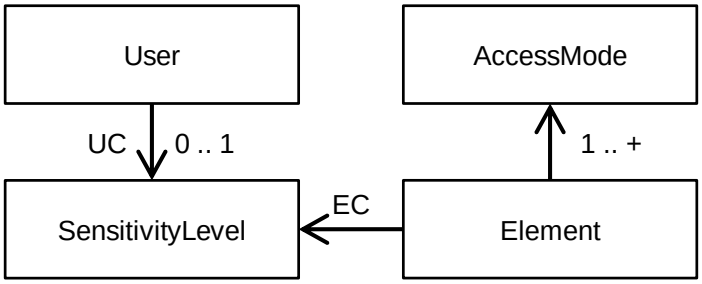

Figure 4.15: LSID M2 metamodel.

Next, at the meta-model layer (M2) of the MOF, the User Diagram as discussed in Section 4.2.5 and shown in Figure 4.9 is composed of six major meta-classes as given in Figure 4.16: User, SensitivityLevel, UserRole, RoleSlice, SOD, and ME. The User meta-class represents all of the possible instances of users in a particular application. Both the User meta-class and the RoleSlice meta-class, is a subtype of the UserRole meta-class.

The UR tag represents user-role assignments (RBAC), the separation of duty (SOD) meta-class represents the separation of duty relations, and, the mutual exclusion (ME) meta-class represents the mutual exclusion relations between roles. The SensitivityLevel meta-class, which represents the sensitivity as related to LBAC is a clearance level for a user and is tied to the User meta-class. This distinction shows an important feature of the security framework presented in this dissertation, namely, that RBAC and LBAC capabilities are orthogonal.

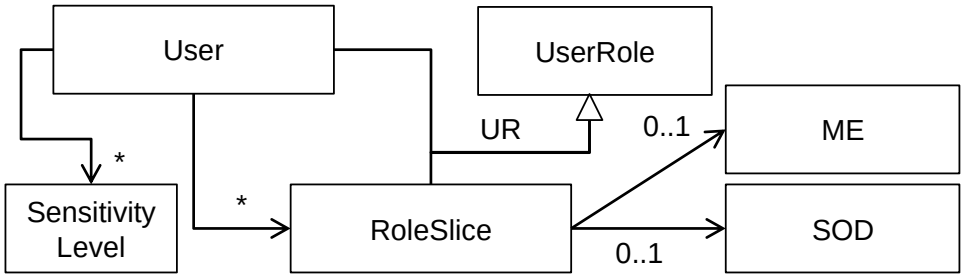

Figure 4.16: User Diagram M2 metamodel.

The metamodel of the Delegation Diagram (DD) as presented in Section 4.2.6 and shown in Figure 4.10 is given in Figure 4.17. The metamodel consists of three meta-classes: OriginalUser, DelegableUser, and RoleSlice. The OriginalUser meta-class, along with the RoleSlice meta-class represents the original users of the application and their assigned roles. The DelegableUser, connected to the RoleSlice meta-class, represents the user/role pairs of authorized delegations. In turn, the Delegation tag in the connection between OringalUsers and DelegableUsers represents the ability to perform the delegation operation.

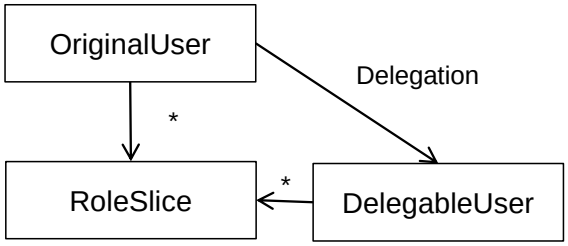

Figure 4.17: Role Delegation Diagram M2 metamodel.

The final metamodel of the Authorization Diagram (AD) as presented in Section 4.2.7 and shown in Figure 4.11 is given in Figure 4.18 and consists of four meta-classes: UserRole, Authorization, Instance, and Schema. The UserRole meta-class represents the specific user/role pair in a similar fashion as the case of the UD in Figure 4.16 The Authorization meta-class is connected to the Instance and Schema meta-classes to represent whether an authorization to an instance or schema exists and is represented with the 0..+ tag on the directional connection. This metamodel definition allows scenarios in which a user might not be authorized to any schema/instance, or any combination of the two (e.g. all schemas and all instances).

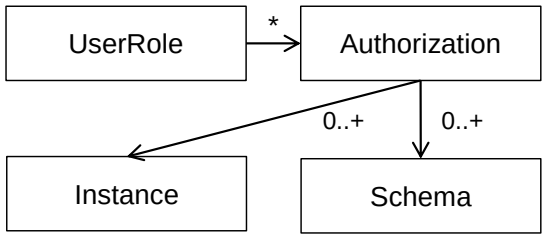

Figure 4.18: Authorization Diagram M2 metamodel.

4.4 Related Work in Security Modeling with UML

This section provides related work on secure software engineering and security for multiple concerns for a system to be secured: functional, collaborative and information. In the functional security area, there have been many research efforts that involve UML. SecureUML (Basin, Doser, & Lodderstedt, 2006) is a modeling language to design security in distributed systems. This language is based in UML, extending its semantics

and notation to support RBAC, and certain authorization constraints. The focus of this approach is to utilize UML to specify access control as part of the main design of the application and then automatically generate access control infrastructures based on the design models. The approach defines a meta-model for SecureUML and details a methodology to integrate it into several design modeling languages. UMLSec, described in (Jürjens & Juerjens, 2005) and improved by (Zisman, 2007) and ((Popp, Jurjens, Wimmel, & Breu, 2003), is an extension to UML that defines several new stereotypes towards formal security verification of elements such as: fair exchange to avoid cheating for any party in a 2-party transaction; secrecy and confidentiality of information (accessible only to the intended people); secure information flow to avoid partial leaking of sensitive information; and, secure communication links like encryption. AuthUML (Alghathbar, 2007) models RBAC policies using use cases and Horn clauses to represent the access control information and to check its consistency. The approach of (Pavlich-Mariscal et al., 2010) includes the definition of several new UML diagrams to represent different access control concerns (RBAC, MAC, and DAC), and a set of features that represent small subsets of an access control model. These features can then be composed to create custom access control policies.

In the collaborative security area, research has occurred in many areas. In terms of access control and collaboration, in (Tolone, Ahn, Pai, & Hong, 2005), a set of eight criteria (complexity, understandability, ease of use, applicability, groups of users, policy specification, policy enforcement, and granularity) critical for a collaborative environment are presented. The eight characteristics and their support are evaluated against seven access control models (Matrix, RBAC, TBAC, TMAC, C-TMAC, SAC,

and Context-AW). This work demonstrates that collaboration capabilities are not primary goals in access control models. As given, the seven access control models do not support an integrated model for coordinated, obligated, secure, and team-based collaboration. Another related area is a distributed secure interoperability framework for collaboration environments (Sachpazidis, Rizou, & Menary, 2008). This framework presents a multi-system architecture in which different stakeholders with different privileges collaborate with one another towards optimizing monitoring prescription intake. With regard to collaboration, workflow, and security, the work of (Shehab, Bertino, & Ghafoor, 2005) addressed security services that support inter-organizational collaborative enterprises, which may span multiple organizations. This work presents a framework for mediator-free collaboration. Similarly as in (Sachpazidis et al., 2008), this work focuses on inter-system collaboration, while our work focuses on early integration of collaboration requirements into the software engineering process. The work of (Kang, Park, & Froscher, 2001) concentrates on workflows that are addressed from an access control perspective. This is an important aspect of our effort (Berhe et al., 2010), where workflow is also represented as collaboration steps with access control addressed at each step and for the overall workflow. The work in (Y. Sun & Pan, 2005) proposes a model that integrates RBAC into workflows. In their approach, permissions, roles, cardinality, ancestors (pre-obligations), and a status value are assigned to activities (collaboration steps).

Finally, in the informational security area, a later effort (Mouelhi, Fleurey, Baudry, & Le Traon, 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. (Basin et al., 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. In the approach presented in (De la Rosa Algarín, A. et al., 2013), UML is leveraged to provide a secure information engineering approach for XML schemas and documents. By extending UML with new XML diagrams, an enforcement security policy in XACML can be generated and deployed for access control purposes (not unlike automatic code generation from UML class diagrams). By doing this, the approach scales to scenarios that involve a high volume of XML documents validated against a common schema.

Chapter 5
Security Policy Generation Process

To this point, this dissertation has defined a security framework that includes a model for access control for RBAC, LDAC, and MAC for the definition of permissions against tree-structured documents (see Chapter 3 again) coupled with the definition of new UML diagrams for the security modeling of schemas (see Chapter 4 again). This chapter presents the third component of our framework that supports the automatic generation of a security enforcement policy when given a security design for a set of schemas as captured in our new UML diagrams (see Chapter 4). UML has a long history for the automatic generation of code (Vogel-Heuser, Witsch, & Katzke, 2005) in varied languages; our usage of our new UML diagrams to generate a security policy is consistent with this usage. In this chapter, we present a process for the generation of enforcement policies that transitions a UML design containing a Document Schema Class Diagram (DSCD), a Secure Information Diagram (SID), a Document Role Slice Diagram (DRSD), lattice-based access control SID (LSID), a User Diagram (UD), a Delegation Diagram (DD), and an Authorization Diagram (AD); see respectively Sections 4.2.1 to 4.2.7.

To support the automatic generation of a security enforcement policy, we define a set of *mapping statements (MSs)* that are utilized to define the conditions under which the combination of the various diagrams (DSCD, SID, DRSD, LSID, UD, DD, and AD) can be utilized to support the creation of respective policies for RBAC, LBAC, DAC, and authorization. A mapping rule (MR) is defined to take the security model concepts and capabilities to Chapter 3 that both underlie correspond to different portions of the new the

UML diagrams of Chapter 4 and use this combination to yield a portion of the security policy. To illustrate, in support for RBAC: a *role mapping statement (R-MS)* takes a specific role such, as Physician from the healthcare scenario of Section 2.6, and maps it to the policy's subject construct, such as the <Subject> element of XACML's specification; an e*lement mapping statement (E-MS)* takes an attribute, such as a child node of 'clinical_document_header' or 'section' segments of the HL7 CDA from the healthcare scenario of Section 2.6, and maps it to the policy's resource construct, such as the <Resource> element of XACML's specification; and, a *permission mapping statement (P-MS)* establishes permissions for the element (read, aggregate, insert, update, delete) as actions against elements of the document to be secured. These three mapping statements support the transition of information from DSCD, SID, and DRSD into a security policy segment that supports RBAC, specifically restricted to one user role combination.

In support of LBAC, mapping statements are need with respect to: the user's clearance level, the element's classification (sensitivity), and the access mode of the operation being performed. Specifically: a *subject (user) mapping statement (SU-MS)* that takes a specific user, such as Elisa from the healthcare scenario of Section 2.6, and maps it to the policy's subject construct; an *action (operation) mapping statement (AO-MS)* that takes a specific operation (e.g. read, aggregate, insert, update, delete from Section 3.3) and its access mode and maps it to the action construct of the policy; and, a *resource (element) mapping statement (RE-MS)* that takes an element that needs classification levels to be added and maps it to the resource construct of the policy. In support of DAC: a *delegation user (original user) mapping statement (DUOU-MS)* takes

a user from the Original Users set as defined in the Delegation Diagram of Section 4.2.6 and Section 3.5 to a user construct in the policy, such as the <Subject> element introduced for XACML; a *delegation resources (roles) mapping statement (DRR-MS)* takes the role to be delegated, such as Physician from Section 4.2.6, and maps it to the resource construct of the policy; and, a *delegation targets (delegable users) mapping statement (DTDU-MS)* corresponds to the users in the delegable set as set in the DD from Section 4.2.6 and Section 3.6 and maps it to the delegation target construct of the policy. The authorizations over schemas and instances are in turned assigned to users. We utilize a *subject (user) mapping* that takes the user from the User Diagram (UD) of Section 4.2.5 and the Authorization Diagram (UD) of Section 4.2.7; and a *resources (schemas and instances) mapping* that takes the schemas and instance constructs (e.g. CDA and Carol Smith's record from the healthcare scenario of Section 2.6). This characteristic allows the definition of authorizations in a policy similar to assigning a given user an LBAC clearance. Overall, the work in this chapter presents a high-level view of the policy generation process that is accompanied by a detailed examination of the policy generation using the eXtensible access control modeling language (XACML) (Godik et al., 2002).

The remainder of this chapter is organized into six major sections. Section 5.1 provides an overview of the mapping process from an architecture perspective to take the new UML diagrams through a process that results in the generation of a security policy. Section 5.2 introduces the key XACML concepts needed for the discussion in this chapter, including the normal components of a security architecture that enforces XACLM policies, and policy/rule combination algorithms. Section 5.3 presents the generation process for RBAC, LBAC, DAC delegations, and authorizations that provide

the capability to convert from the new UML security diagrams (see Section 4.2 again) to an XACML security policy via a series of mapping statements. Section 5.4 defines an algorithm that automates the policy generation process by organizing the mapping statements as presented in Section 5.3 into a structured process. Lastly, Section 5.5 reviews related work on policy generation and integration.

5.1 An Architecture for Security Policy Generation

In this section, we provide a high-level view of the security policy generation process that combines the access control concepts and capabilities of our security model (see Chapter 3 again) with the new UML diagrams into an architecture. As shown in Figure 5.1, the seven new UML diagrams in the first column (DSCD, SID, DRSD, LBAC, UD, DD, and AD) are used in various combinations (see four different arrow types) in order to start a process that can map them through access control models RBAC, LBAC, DAC delegations, and authorizations (see column two) in order to identify the key policy components (see column three) that are then utilized to automatically generate a security policy (fourth column). First, DSCD, SID, DRSD and UD are combined to produce an RBAC oriented policy for each user/role combination as shown with the solid arrows in Figure 5.1; as a result, multiple security policies are generated on a user/role basis. Second, DSCD, SID, LSID, and UD are combined to produce an LBAC oriented policy for each user as shown with the dashed arrows in Figure 5.1; again, specific security policies are generate for each user. Third, DRSD, UD, and DD are combined to produce a security policy that defines the delegable users and the role that can be delegated as shown with the long dash dot arrows in Figure 5.1; again, this results in a separate policy each user/role combination. Finally, DSCD, DRSD, UD, and AD are combined to

generate the policy that identifies the schemas and instances are authorized for a specific user as shown with the long dash dot-dot arrow in Figure 5.1. For each of these combinations, there is a transition to the policy components that form the basis of the generated policies (third column of Figure 5.1). The last step in the process (fourth column of Figure 5.1), illustrates the alternative policies that can be generated, including XACML (the focus of this dissertation), SQL DDL code for a relational database system, and aspect-oriented programing (AOP) for an object-oriented application (Pavlich-Mariscal et al., 2010).

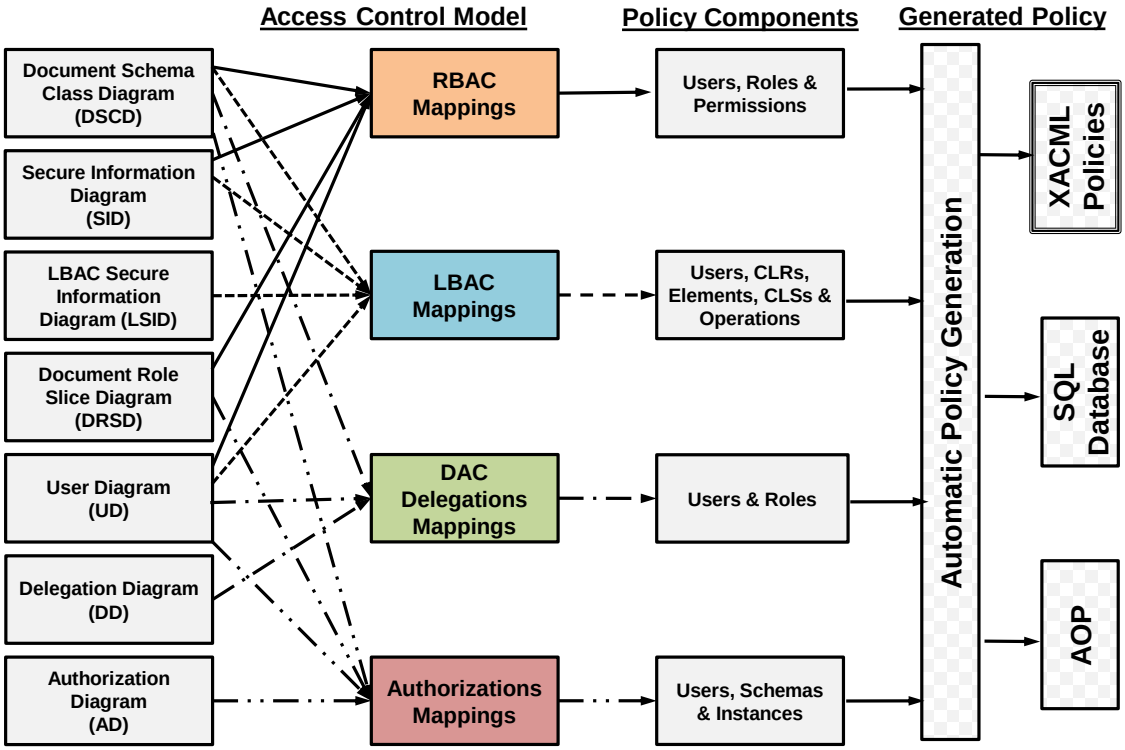

Figure 5.1: An Architecture for Generating Policies from UML Diagrams.

To illustrate the usage of the architecture, we continue with the healthcare scenario of Section 2.6. Suppose that user Elisa's role of Physician has a no read permission over a psychiatric element of the CCR schema. In this case, the policy would involves Elisa, Physician, and psychiatric element would filter the CCR schema to hide this information

for the RBAC mapping as well as controlling and the ‘Carol Smith’ instance for the Authorizations mapping. The generality of the policies created via the architecture presented in Figure 5.1 could be readily applied to an eXtensible Stylesheet Language (XSLT) (Clark, 1999), to other query tools such as XPath (Clark & DeRose, 1999) and XQuery (Boag et al., 2002) for XML documents, or to a relational database schema and tuples. The generated policies must also be able to target the software methods at the system’s level in order to support destructive operations such as insert, update, and delete. In all of these cases, the validation of the consistency of the altered instance is left as a task of the system. In other words, this security model does not validate proper alterations to an instance against its schema; it just assures that the security requirements that control the destructive operations are properly enforced. For example, in the case of a relational database, a system such as Oracle (Harrison, 2000) would be responsible for enforcing operations against the database tuples. From the perspective of a security administrator, the ability to automatically generate enforcement policies from the UML diagrams as shown in Figure 5.1 would reduce the costs associated when security requirements are frequently changed and modified over time. UML has a long history of the automatic generation of object-oriented code (OOC) (Vogel-Heuser et al., 2005) from UML class diagrams, and, this automated process reduces the complexity to the click of one button with modeling tools that have that capability, e.g., CodeSmith Tools (CodeSmith tools.2014), Acceleo (Mtsweni, 2012), etc. The benefit of automatic policy generation, combined with the security framework’s target of defining security definitions at the schema level, would further reduce the effort and cost of updating security requirements that can potentially affect thousands or millions of documents (instances).

This benefit is particularly useful in the case of large institutions in the healthcare domain such as hospitals and clinics.

The architecture given in Figure 5.1 can be reformulated as given in Figure 5.2 to provide a view of this approach from the perspective of UML metamodel changes, UML diagrams, and the support for policy generation at the schema and instance level that results in the creation of a set of enforcement policies. Specifically, starting from the top of Figure 5.2 downward, the generalized mapping approach utilizes the meta-classes of the UML Meta-Object Facility (MOF) (Poernomo, 2006) M2 layer presented in Section 4.3 (e.g., User, UserRole, Permission, RoleSlice, etc.) to create two new sets of authorizations (middle of Figure 5.2). These meta-classes are combined to form the seven new UML diagrams, which in turn supports both *policy level authorizations* and *document level authorizations*. Policy level authorizations include delegations and authorizations, or those operations that have an effect on a user and not on a document. Document level authorizations include permissions (operations over elements) on the tree-structured document, or those operations that have an effect on the documents being secured and not on the users. This includes support for delegation, where there is a set of original users that can delegate their roles, and a set of delegable users that can receive specific roles. This means that roles are resources that can be interchanged between users of an application and have properties (e.g., the permissions are defined as part of a policy) such as pass-on authority for second-level delegations. To support all of these capabilities, it will be necessary to have a policy language that has a non-normative construct that serves to identify the delegable users (delegation targets). This construct

would act as a holder for all of the delegable users that can receive the role being treated as a resource.

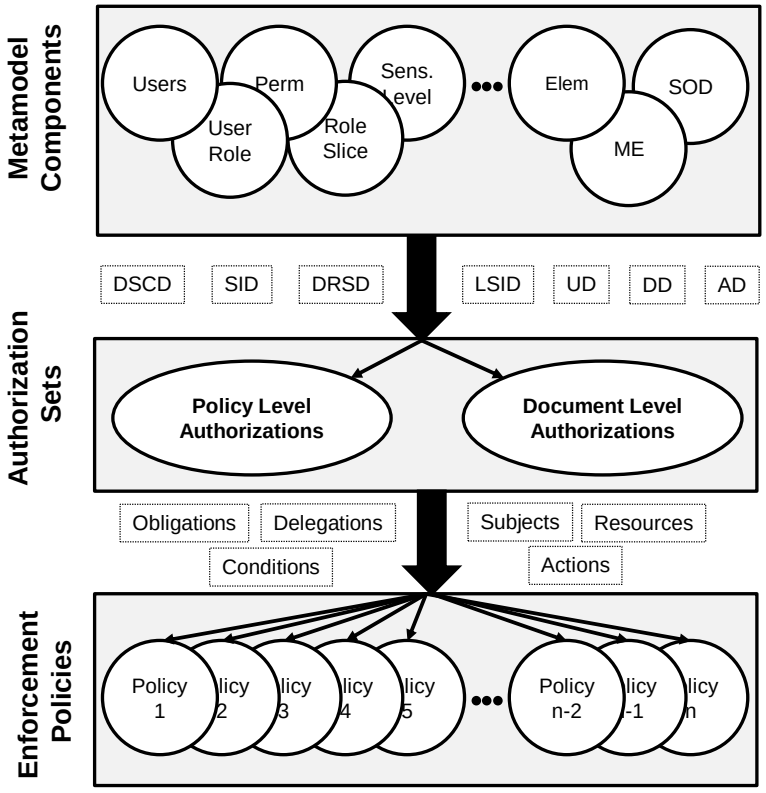

Figure 5.2: Mapping Process to Generate Enforcement Policies.

5.2 XACML Concepts and Rule Combining Algorithms

In the same way that XML (Bray et al., 1998) provides a common, structured language for information exchange among heterogeneous systems, the eXtensible access control modeling language (XACML) (Godik et al., 2002) defines 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 5.3) include: the *PolicySet*, the *Policy*, and the *Rule*. In XACML, every policy has: a rule

combining algorithm to interconnect the result of rules in order to reach a proper authorization; a description to textually illustrate the purpose of the policy; and, an identifier for indexing. The body of a policy consists of a target <Target> element and one or more rules <Rule>. The <Target> element of the policy or the <Rule> element is utilized by the policy evaluation point to determine whether the policy is relevant to the request received by the application. If no policy or rule is relevant to the request, the request is dropped, likely indicating an invalid access. The XACML specification defines four standard combining algorithms: *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.

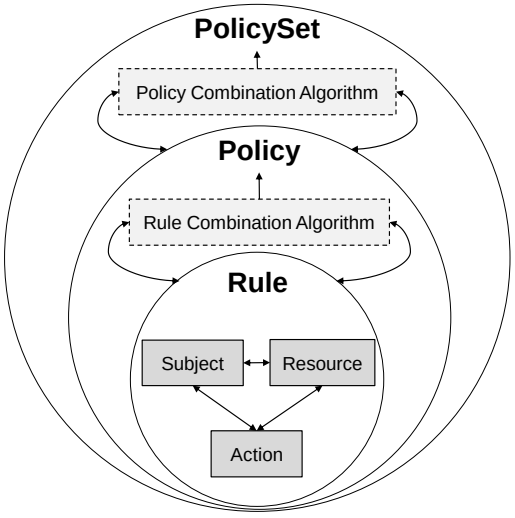

Figure 5.3: Layered Representation of XACML's PolicySet, Policy and Rule Constructs.

The architecture of a typical security system that utilizes XACML for enforcement is revisited with an example using our healthcare scenario from Section 2.6 in Figure 5.4. The *Policy Enforcement Point (PEP)* allows a request to be made on a resource such as

Dr. Ketchum playing a Physician role to access a CCR instance of Carol Smith. The *Policy Decision Point (PDP)* evaluates the request and provides a response according to the policies in place. That would define, in the case of a Physician role played by Dr. Ketchum, as to whether a portion of a CCR schema can be accessed (read/written). The *Policy Administration Point (PAP)* is utilized to write and manage policies, which for the CCR schema and its associated instances would be utilized to deliver the appropriate subset of information to a Nurse or Physician role and the individual who is playing that role. Last, the *Policy Information Point (PIP)* can be utilized to arbitrate very fine grained security issues, which could be employed to control access to mental health data of Carol Smith, allowing Ketchum, while denying Fakington.

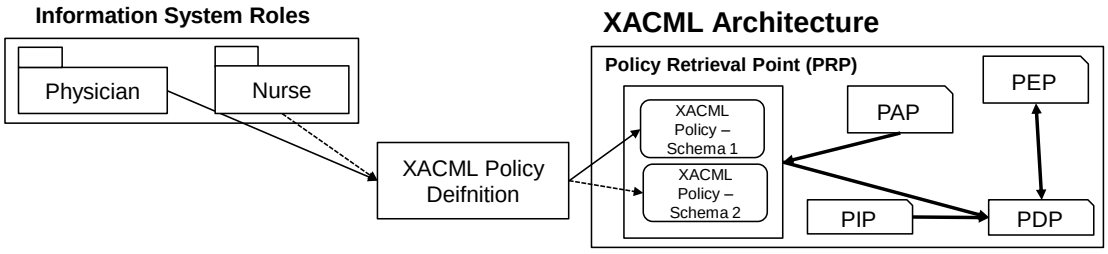

Figure 5.4: Typical Security Architecture for XACML Policies.

The layered representation (Figure 5.3) and architecture (Figure 5.4) provide the means to support a process that can utilize the seven new UML diagrams as a starting point for security policy generation. Specifically, to create a XACML policy, we can utilize the policies' language structure and processing model where a policy consist of a *PolicySet*, a *Policy*, and a *Rule*. Based on the capabilities of XACML, we are taking an approach that each of the application' role(s) as represented, in say, a DRSD, must be mapped into a XACML *Policy* structure with its own set of rules that represent the appropriate enforcement for roles against a schema, the DSCD. Note that multiple

XACML *Policy* structures may be generated, resulting in a *PolicySet* for a specific set of XML schemas that have an associated set of user/roles that comprise a given application.

The collection of Policy structures, one for each DRSD, is contained in a *PolicySet*, combined via an algorithm specified by the *PolicySet's PolicyCombiningAlgId* attribute that targets the particular tree-structured schema. In order to support instance-level security for tree structured documents (schemas), the DRSD is mapped into an XACML Policy with the combining algorithm *deny-overrides* chosen. With the *deny-overrides* 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 evaluated as *Deny*. We note that while this assumption works when focusing on access control for instances in the document-level, as in the work of this dissertation, other higher-level systems (e.g., software applications that utilize the instance, etc.) can very deploy security policies with different combining algorithms. The pseudo-code implementation of all of the policy-rule combining algorithms can be found in Appendix A. An XACML *PolicySet* is utilized to make the authorization decision via a set of rules in order to allow for access control decisions such as granting access to a resource, allowing an operation to continue, etc. 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 tree-structured schema.

5.3 Security Enforcement Policy Generation in XACML

This section concentrates on the third component of our framework in this dissertation that supports the automatic generation of a security enforcement policy in XACML from the security design as captured by the new seven UML diagrams (DSCD, SID, DRSD, LBAC, UD, DD, and AD). To properly generate and XACML policy/rule pairs for the different UML diagrams and their components, there are a number of key correlations that can be established between the nomenclature of UML and the terminology of XACML. These correlations represent meta-mapping statements that defines the matching between a security design in UML to XACML in order to support a fully automated process. Specifically, for our purposes, we correlate XACML's Policy and Rules to our security model (see Chapter 3) as follows:

- Policy's *PolicyId* attribute value is the string *AccessControlPolicy{Model}*, where {Model} serves as a placeholder for LBAC, RBAC or DAC, concatenated to a unique identifier such as an integer. PolicyId represents a unique identifier that is used to index and identify any given policy from a group of policies.
- Rules' *RuleId* attribute value is the string *ProductRule{Model}* concatenated to a unique identifier such as an integer. RuleID, in the same fashion as the PolicyId, is utilized to index and identify any given rule inside a policy from the rest of the rules that can be part of said policy.
- Rules' *Description* value is the string *AccessControlPolicyRule{Model}* concatenated to a unique identifier such as an integer. Description is used to textually describe the purpose of the rule inside a policy.

The intent of this section is to transition the seven new UML diagrams to its realization as an XACML policy through the definition of *mapping statements* that represent the actions needed to generate policies for RBAC, LBAC, and DAC delegations/authorizations, respectively, in Sections 5.3.1, 5.3.2, and 5.3.3. We note that there is no predefined order to apply the mapping statements at a higher-level (between RBAC, LBAC and DAC) or at a lower level (each mapping statement). These mapping statements work in a similar fashion to the UML profile presented in Section 4.2.1, which means that each mapping statement create a relationship between a component of the UML diagram and metamodel to a component of the XACML schema. This notion is further discussed in Section 5.4 with the automatic algorithm for XACML policy generation.

Each of the first three subsections (5.3.1, 5.3.2, and 5.3.3) have a unified format of presentation, structured in three parts: a definition of the mapping statements that transition the UML diagrams and their components to XACML equivalents; a demonstration of this mapping process through the use of example UML diagrams; and, a detailed explanation that illustrates the mapping from UML to equivalent XACML code. To complete the discussion, Section 5.3.4 explores the case when an application has a combination of RBAC, LBAC, and DAC in terms of its security capabilities and its mapping to XACML.

5.3.1 RBAC Capabilities

The generation of RBAC policies with XACML is facilitated due to the fact that RBAC's combination of a role, element, and permission (see Defns. 11 to 29 in Section 3.3) seamlessly aligns to XACML's paradigm of <Subjects>, <Resource>, and

<Actions>. In our previous work (De la Rosa Algarín, A. et al., 2013; De la Rosa Algarín, A. et al., 2013), we presented an automated method of achieving this specific goal with an emphasis on XML. This dissertation extends that effort by generalizing away from XML and considering a more varied set of operations (e.g., read, aggregate, insert, update, delete) that can be performed on general-purpose tree-structured documents and their schemas and instances. Note that while this section and dissertation is based on XACML policy generation, the work presented herein can easily be reused to generate policy code in another format. The mapping statements as presented are generalizable to generate "code" for other target domains; for example, in RBAC, the mapping of roles, elements, and permissions has to still occur regardless of whether the target is XACML or SQL DDL and a relational database system.

The mapping statements for RBAC are shown in Table 5.1, and include: a *role mapping statement (R-MS)* that takes a role such as Physician and a user such as Elisa to a <Subject> in XACML; an e*lement mapping statement (E-MS)* that takes an attribute such as `FamilyHistory` in CCR to a <Resource> in XACML; and, a *permission mapping statement (P-MS)* that takes a permission such as update to an <Action> in XACML. As discussed in Section 5.1 and shown using the solid arrows in Figure 5.1, the mappings for RBAC utilize the DSCD, SID, DRSD, and UD diagrams from Chapter 4. For the *R-MS*, the DRSD and UD are utilized to set the XACML policy's <Target> Subject element as the role and to set the role identifier (using R-MS1) along with the user and user identifier (using R-MS2). The SubjectMatch's MatchId attribute utilizes the function 'string-equal' to evaluate the user's role as modeled in the DRSD (using R-MS3). The XACML policy for a specific role will have as many *Rule* constructs as permission combinations shown

in the DRSD (using R-MS6). Finally, the *Subject* element in the *Rule*'s <Target> subtree is the role from the DRSD (using R-MS7). For the *P-MS*, the DRSD and SID are utilized to represent the permissions (operations) that are tied to the respective role. The policy's <Target> element's *Action* child is set to <AnyAction/> in order for the policy to apply to the role when any request is done (using P-MS1). The *ActionMatch's* MatchId attribute utilizes the 'string-equal' function in a similar fashion to the *SubjectMatch's* MatchId (using P-MS2), while the *ActionAttributeDesignator's* AttributeId is set to the permission's operation (e.g., read, aggregate, insert, update, or delete) (using P-MS3). In the case of the *P-MR*, the policy's rules <Action> children are <Operation> elements with the <operationName> subchild and <opAccessMode> values which are the DRSD's stereotypes for the element classes (using P-MS4). Last, for the *E-MS*, the DRSD, DSCD, and SID are utilized in conjunction to represent those elements that a role can operate over by virtue of the tied operations in XACML form. In a similar fashion to the policy's <Target> Subject element, the Resource element is set to <AnyResource/> to ensure that the higher-level policies apply to the role (using E-MS1). The *Rule's* <Resource> children are <element> nodes with element data that corresponds to the element classes in the DRSD, DSCD, and SID diagrams (using E-MS3).

To provide a realistic example of the usage and application of the mapping statements and associated process, consider the healthcare scenario of Section 2.6, where the user Elisa has the role of Physician. Consider the DSCD in Figure 4.3 of Section 4.2.1 and the SID in Figure 4.6 of Section 4.2.2, which provide the basis for defining the DRSD of Figure 5.6. The UD that corresponds to user Elisa is shown in Figure 5.5, the DRSD that corresponds to the Physician role is shown in Figure 5.6, while the corresponding

generated XACML code is shown in Figure 5.7. Notice that the figures are all labeled with capital letters. In the UML diagram figures, we will refer to Section A, Section B, etc., while in the XACML code figures, we will refer to part A, part B, etc.; To begin, section A of Figure 5.5 denotes the user that will become part of the user/role combination for a given role's policy with the role shown in section B. Figure 5.6 shows the DRSD's role in section A, with section B referring to those elements (resources in XACML) that a role has some a permission over, and section C indicating those operations allowed to be performed by the role. At a conceptual level, section A of the DRSD in Figure 5.6 corresponds to the <Subject> subtree of the XACML policy in Figure 5.7 (oarts A and B, respectively), while at a higher-level policy <Target>, the user from Figure 5.5 (section A) and the role (section B) are used in conjunction with the DSCD in Figure 5.6's role (section A).

The R-MS1 and R-MS2 of Table 5.1 are used for Figure 5.7's part A, while the R-MS7 is used for Figure 5.7's part B. In this case, section B in Figure 5.6 denotes the components of the DRSD that corresponds to the *P-MS* of Table 5.1. These sections of the DRSD are used by P-MS4 to result in Figure 5.7's part D, which contains the possible actions the user Elisa under the role of Physician can perform. Lastly, section B of Figure 5.6 denotes the components of the DRSD that corresponds to the *E-Ms* of Table 5.1, which map to the <Resource> subtree of the policy in Figure 5.7 (part C). The resulting policy for RBAC that is generated would permit the role Physician to write the past medical history element of the CDA (Alschuler et al., 2002) instance that is shown in Figure 5.7. At the policy level (Figure 5.7), the subject is the role and user (part A lines 7-13 of Figure 5.7, corresponding to the A and B parts of Figure 5.5), and a match is done

Mapping Statement Type	Involved UML Diagram(s)	Mapping Statements
R-MS **(Role Mapping Statements)**	DRSD, UD	**R-MS1.** XACML Policy's <Target> *Subject* is the role and role identifier set as a <role> subtree with <roleName> and <roleID> children that corresponds to the DRSD package name. **R-MS2.** XACML Policy's <Target> *Subject* is the user name and user identifier set as a <user> subtree with <name> and <id> children that corresponds to the UD. **R-MS3.** SubjectMatch's *MatchId* uses the function "string-equal" to evaluate the user's role as modeled in the DRSD. **R-MS4.** *AttributeValue* of the Subject is a string, and the value is the «DRSD» *Role*. **R-MS5.** *SubjectAttributeDesignator's AttributeId* is the role attribute. **R-MS6.** As many *Rule* per role Policy as permission combinations. **R-MS7.** *Subject* in *Rule* is set as the *Subject* in the higher-level <Target> (role).
P-MS **(Permission Mapping Statements)**	DRSD, SID	**P-MS1.** XACML Policy's <Target> *Action* set to <AnyAction /> to ensure that the higher-level policy applies to the role. **P-MS2.** *ActionMatch's MatchId* uses the function "string-equal". **P-MS3.** *ActionAttributeDesignator's AttributeId* set to the operation's tag (e.g. read, aggregate, insert, update, delete). **P-MS4.** *Rule's* <Actions> children are <Operation> with the operation name (<operationName>) and access mode (<opAccessMode>) that correspond to the stereotypes of the +access() method in the «element» classes in the DRSD package.
E-MS **(Element Mapping Statements)**	DRSD, DSCD, SID	**E-MS1.** XACML Policy's <Target> *Resource* set to <AnyResource /> to ensure that the higher-level policies applies to the role. **E-MS2.** Each Resource's *ResourceMatch* has a *MatchId* that determines the usage of the function "string-equal". **E-MS3.** *Rule's* <Resources> children are <element> with the element identifier (<elementID>) and element name (<elementName>) that correspond to the «element» classes in the DRSD, DSCD and SID packages.

Table 5.1: RBAC Mapping Statements.

based on the attributes of the role, where a unique ID could exist that would be denoted in part A by the <ruleID> element in line 9 and <id> element in line 12 of Figure 5.7. Once this is verified, the rule 'simple-RBAC-rule' (lines 21-42 of Figure 5.7) would be evaluated to determine if the triple of role, element, and operation match. This is represented by the role in part B, lines 23-28 of Figure 5.7 corresponding to the section B of Figure 5.5 and section A of Figure 5.6, by the element in part C, lines 29-34 of Figure 5.7, corresponding to the classes of Figure 5.6 with section B denoting their names, and by the operation in part D, lines 35-40 of Figure 5.7, corresponding to the C parts of Figure 5.6. As result, the user/role is allowed to perform the insert operation on the past medical history element as denoted by the Effect="Permit" of the XACML rule in line 21 of Figure 5.7.

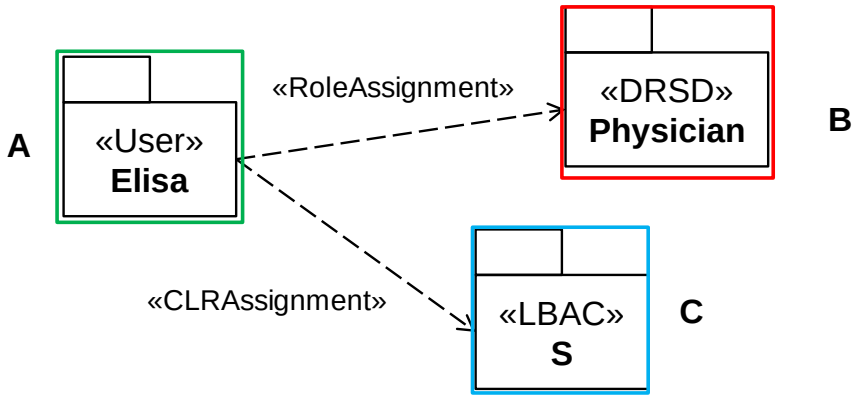

Figure 5.5: A User Diagram of User Elisa under Role Physician with a Clearance of Secret (S).

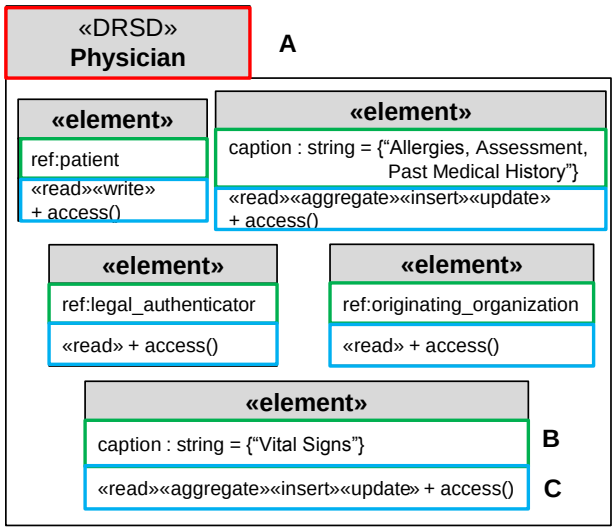

Figure 5.6: A DRSD for the Role of Physician in the Healthcare Scenario of Section 2.6.

```
1.  <Policy PolicyId="ada-example-rbac-policy"
2.   RuleCombiningAlgId="deny-overrides">
3.    <Description>This is a pseudocode example of an
4.     XACML policy with RBAC capabilities for role
5.     Physician and user Elisa</Description>
6.     <Target>
7.       <Subjects>
8.        <role>
9.         <roleID>5</roleID>
10.        <roleName>Physician</roleName>
11.       </role>
12.       <user><id>6</id><name>Elisa</name></user>
13.      </Subjects>
14.      <Resources>
15.        <AnyResource/>
16.      </Resources>
17.      <Actions>
18.        <AnyAction/>
19.      </Actions>
20.    </Target>
21.    <Rule RuleId="simple-RBAC-rule" Effect="Permit">
22.    <Target>
23.      <Subjects>
24.        <role>
25.          <roleID>5</roleID>
26.          <roleName>Physician</roleName>
27.        </role>
28.      </Subjects>
29.      <Resources>
30.        <element>
31.          <elementID>el-3</elementID>
32.          <elementName>Past Medical History</elementName>
33.        </element>
34.      </Resources>
35.      <Actions>
36.        <operation>
37.          <operationName>insert</operationName>
38.          <opAccessMode>write</opAccessMode>
39.        </operation>
40.      </Actions>
41.    </Target>
42.   </Rule>
43. </Policy>
```

A

B

C

D

Figure 5.7: Pseudo-code for XACML Policy with RBAC Capabilities.

5.3.2 LBAC Features

LBAC capabilities involve three major components: the user's clearance level, the element's classification (sensitivity), and the access modes of the operation being performed against the target element. In Section 4.2.4, we introduced the LSID to graphically represent the LBAC attributes of elements to be secured. To properly generate an XACML policy that supports LBAC, we leverage DSCD, SID, LSID, and UD in combination with constructs of the XACML specification. Recall that the UD contains the LBAC characteristics tied to a specific user, providing us the information necessary to complete the user's clearance level component of an LBAC policy. In turn, the LSID provides us with the element's classification (sensitivity) with respect to the access modes permitted (read and/or write) against the elements of the LSID. To represent this in an XACML policy, we designate the target of a policy, <Target>, with a <Subject> value equal to the name of the user. This produces a policy-level target that is specific for a user. The LBAC rule of the policy leverages the LBAC user object from Definition 6 in Section 3.4, including the clearance level, mapped to the rule's <Target> <Subject> subtree. Then, the resource's LBAC characteristics denoted in the LSID, mapped to the rule's <Target> <Resources> and <Actions> subtrees.

The mapping statements for LBAC are shown in Table 5.2, and include: a *subject-user mapping statement (SU-MS)* that takes a user such as Elisa to a <Subject> in XACML; an *action-operation mapping statement (AO-MS)* that takes a permission such as insert or delete to an <Action> in XACML; and, a *resource-element mapping statement (RE-MS)* that takes an element such as such as `FamilyHistory` in CCR to a <Resource> in XACML. As discussed in Section 5.1 and shown using the dashed arrows

in Figure 5.1, the mappings for LBAC utilize the DSCD, SID, LSID, and UD from Chapter 4. For the *SU-MS*, the UD is utilized to set the XACML policy's <Target> Subject element as the user and user identifier (using SU-MS1). The SubjectMatch's MatchId attribute uses the function 'string-equal' to evaluate the user's name as modeled in the UD (using SU-MS2). Finally, the *Subject* element in the *Rule*'s <Target> subtree is the user from the UD (using SU-MS3) along with the user's clearance (using SU-MS4). For the *AO-MS*, the UD and LSID are utilized. The policy's <Target> element's *Action* child is set to <AnyAction/> in order for the policy to apply to the user when any request is performed (using AO-MS1). The *ActionMatch's* MatchId attribute utilizes the 'string-equal' function in a similar fashion to the *SubjectMatch's* MatchId (using AO-MS2), while the *ActionAttributeDesignator's* AttributeId is set to the permission's operation (e.g., read, aggregate, insert, update, or delete) (using AO-MS3). In the last case for the *AO-MR*, the policy's rules <Action> children are <Operation> elements with the <operationName> subchild and <opAccessMode> values equal to the LSID's member definitions (using AO-MS4). Lastly, for the *RE-MS*, the DSCD, SID and LSID are utilized in conjunction. In a similar fashion to the policy's <Target> Subject element, the Resource element is set to <AnyResource/> to ensure that the higher-level policies apply to the user (using RE-MS1). The *Rule's* <Resource> children are <element> nodes with element data that correspond to the element classes in the DSCD, SID and LSID diagrams (using RE-MS3).

Mapping Statement Type	Involved UML Diagram(s)	Mapping Statements
SU-MS **(Subject User Mapping Statements)**	UD	**SU-MS1.** XACML Policy's <Target> Subject is the user and user identifier set as a <user> subtree with <name> and <id> children that corresponds to the «User» package of the UD. **SU-MS2.** SubjectMatch's MatchId uses the function "string-equal" to evaluate the user's name and id as modeled in the UD and the security model. **SU-MS3.** Subject in Rule is set as the Subject in the higher-level <Target> (user). **SU-MS4.** Subject in Rule is extended with a <clearance> element that corresponds to the LBAC clearance from the UD.
AO-MS **(Action Operation Mapping Statements)**	UD, SID, LSID	**AO-MS1.** XACML Policy's <Target> Action set to <AnyAction /> to ensure that the higher-level policy applies to the user. **AO-MS2.** ActionMatch's MatchId uses the function "string-equal". **AO-MS3.** ActionAttributeDesignator's AttributeId set to the operation's tag (e.g. read, aggregate, insert, update, delete). **AO-MS4.** Rule's <Actions> children are <Operation> with the operation name (<operationName>) and access mode (<opAccessMode>) that corresponds to the members of the stereotyped «element» classes of the LSID.
RE-MS **(Resource Element Mapping Statements)**	DSCD, SID, LSID	**RE-MS1.** XACML Policy's <Target> Resource set to <AnyResource /> to ensure that the higher-level policies applies to the user. **RE-MS2.** Each Resource's ResourceMatch has a MatchId that determines the usage of the function "string-equal". **RE-MS3.** Rule's <Resources> children are <element> with the element identifier (<elementID>), element name (<elementName>), LBAC classification (<classification>), and access mode (<accessMode>) that corresponds to the stereotyped «element» class of the LSID.

Table 5.2: LBAC Mapping Statements.

To continue the realistic example of the usage and application of the mapping statements and associated process from the prior section, we reuse the UD in Figure 5.5, introduce an LSID in Figure 5.8 that corresponds to the SID of the prior section, with the

corresponding generated XACML code from DSCD, SID, LSID, and UD is shown in Figure 5.9. The UD and section A in Figure 5.5 allows for the generation of the policy's target, specifically the user Elisa part A, line 8 in Figure 5.9. When a request by user Elisa is made on the system that involves one of the elements in the LSID (Figure 5.8), this policy is matched against the name in lines 6-16 of Figure 5.9, and then the rule in lines 17-41 of Figure 5.9 is checked for validity. Elisa is assigned a clearance level of secret (S), denoted by <clearance> element inside the rule in line 23 in part B of Figure 5.9, corresponding to the section C of Figure 5.5. When Elisa tries to perform an insert with an access mode of write, denoted by <opAccessMode> in line 37 in part C of Figure 5.9, the effect of the policy is *Permit* in line 17 of Figure 5.9, and she will be allowed to continue. The element that is targeted in lines 29-33 in part C of Figure 5.9) corresponds to the 'Past Medical History' element class in the LSID of Figure 5.8. Note that the members of the «element» class in the LSID package in Figure 5.8 for the 'Past Medical History' element class is *classified* (c) for when the access mode is *read*, and, *secret* (s) for when the access mode is *write*. This property is matched in lines 30 and 31 in part C of Figure 5.9.

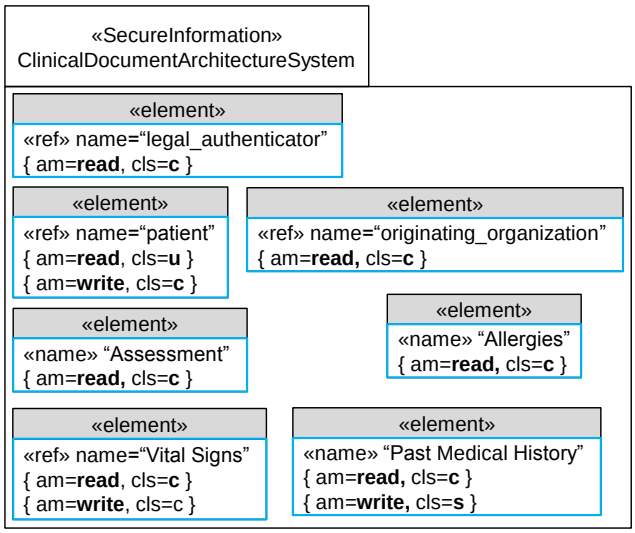

Figure 5.8: An LSID for HL7 CDA Elements from the Scenario of Section 2.6.

```
1.  <Policy PolicyId="ada-example-lbac-policy"
2.   RuleCombiningAlgId="deny-overrides">
3.   <Description>This is a pseudocode example of an
4.    XACML policy with LBAC capabilities for user
5.    Elisa</Description>
6.    <Target>
7.      <Subjects>                                          A
8.        <user><id>6</id><name>Elisa</name></user>
9.      </Subjects>
10.     <Resources>
11.       <AnyResource/>
12.     </Resources>
13.     <Actions>
14.       <AnyAction/>
15.     </Actions>
16.   </Target>
17.   <Rule RuleId="simple-LBAC-rule" Effect="Permit">
18.     <Target>
19.       <Subjects>
20.         <user>
21.           <id>6</id>                                    B
22.           <name>Elisa</name>
23.           <clearance>S</clearance>
24.         </user>
25.       </Subjects>
26.       <Resources>
27.         <element>
28.           <elementID>el-3</elementID>
29.           <elementName>Past Medical History</elementName>
30.           <classification>S</classification>
31.           <accessMode>write</accessMode>
32.         </element>                                      C
33.       </Resources>
34.       <Actions>
35.         <operation>
36.           <operationName>insert</operationName>
37.           <opAccessMode>write</opAccessMode>
38.         </operation>
39.       </Actions>
40.     </Target>
41.   </Rule>
42. </Policy>
```

Figure 5.9: Pseudo-code for XACML Policy with LBAC Capabilities.

5.3.3 DAC Delegations and Authorizations

This section focuses on both delegation (DD) and authorization (AD) from our model as presented in Sections 4.2.6 and 4.2.7, respectively. For delegation, XACML has built-in support that allows the delegator to delegate partial or complete authority to another user in the system. This mechanism avoids the need to modify the original security policy since it effectively decouples the delegation rights and the access rights, thereby simplifying the policy generation process. However, the XACML's delegation component does not have a way to enforce which user can receive what roles, which is crucial to our approach. This effectively turns delegation into a discretionary non-checked operation. In order to support the delegation of roles to predetermined users, namely, delegable users as defined in Section 3.5 with Definition 33, we extend the XACML schema with non-normative constructs that support the creation of specific rules regarding role delegation with the purpose of integrating the delegation permissions on a user basis. Recall from Section 3.5, that in order to have proper role delegation, there is a set of original users (Definition 32 of Section 3.5) that can delegate their roles and a set of delegable users (Definition 33 of Section 3.5) that can receive specific roles. This means that roles can be treated as resources that can be interchanged between users of an application, albeit limited in only one direction and only when an original user delegates it to a delegable user. The roles also have properties, namely, the permissions that are defined as part of a policy, such as pass-on authority for second-level delegations. To support all of these capabilities, we extend the XACML schema with a <DelegationTargets>. The <DelegationTargets> element that acts as a holder of all of the delegable users that can receive the role being treated as a resource. The delegation targets are users from the

delegable users set (see Definition 33 of Section 3.5) that can receive the role that is matched from the policy. To generate the delegation portion of the XACML policy, we define mapping rules between the Delegation Diagram (DD) of Section 4.2.6 and the extended concepts of XACML.

The mapping statements for DAC delegations are shown in Table 5.3, and include: a *delegation subject-original user mapping statement (DSOU-MS)* that takes a user such as Elisa to a <Subject> in XACML; a *delegation resources-role mapping statement (DRR-MS)* that takes a role such as Physician to a <Resource> in XACML; and, a *delegation targets-delegable user mapping statement (DTDU-MS)* that takes a delegable user such as Samantha or Emliy to a <DelegationTargets> extended element in XACML. As discussed in Section 5.1 and shown using the long dash dot arrows in Figure 5.1, the mappings for DAC delegations utilize the UD, DD and DRSD from Chapter 4. For the *DSOU-MS*, the UD and DD are utilized to set the XACML policy's <Target> Subject element as the user and user identifier (using DSOU-MS1). The SubjectMatch's MatchId attribute uses the function 'string-equal' to evaluate the user's role as modeled in the DRSD (using DSOU-MS2). Finally, the *Subject* element in the *Rule*'s <Target> subtree is the user from the UD (using DSOU-MS3). For the *DRR-MS*, the DRSD and DD are utilized in conjunction. In a similar fashion to the policy's <Target> Subject element, the Resource element is set to <AnyResource/> to ensure that the higher-level policies apply to the user (using DRR-MS1). The *Rule's* <Resource> children are <role> nodes with role data that correspond to the role packages in the DRSD and DD (using DRR-MS2). Last, for the *DTDU-MS*, the UD, DRSD and DD are utilized to represent those users that are allowed to receive the role from a delegation operation in XACML form. The policy

rule's newly introduced <DelegationTargets> element is set to the user of those classes insider the DelegationDiagram package of the DD (using DTDU-MS1).

Mapping Statement Type	Involved UML Diagram(s)	Mapping Statements
DSOU-MS **(Delegation Subject Original User Mapping Statements)**	UD, DD	**DSOU-MS1.** XACML Policy's <Target> Subject is the user and user identifier set as a <user> subtree with <name> and <id> children that corresponds to the «User» package of the DD. **DSOU-MS2.** SubjectMatch's MatchId uses the function "string-equal" to evaluate the user's name and id as modeled in the DD and the security model. **DSOU-MS3.** Subject in the Delegation Rule is set as the Subject in the higher-level <Target> (user), the «User» package from the DD.
DRR-MS **(Delegation Resources Role Mapping Statements)**	DRSD, DD	**DRR-MS1.** XACML Policy's <Target> Resource set to <AnyResource /> to ensure that the higher-level policies applies to the user. **DRR-MS2.** Rule's <Resources> children are <role> elements with the role identifier (<roleID>) and role name (<roleName>) that corresponds to the «DRSD» of the DD.
DTDU-MS **(Delegation Targets Delegable User Element Mapping Statements)**	UD, DRSD, DD	**DTDU-MS1.** Rule's <DelegationTargets> children are <User> with the user identifier (<id>) and user name (<name>) that corresponds to the «User» classes of the «DelegationDiagram» package in the DD.

Table 5.3: DAC Delegation Mapping Statements.

Continuing the example, the DD from Section 4.2.6 is reintroduced in Figure 5.10 to explain the mapping relations between the DD and XACML, with the corresponding generated XACML code from DRCD, UD, and DD is shown in Figure 5.11. Figure 5.10 shows a sample of the role delegation from the user Elisa in section A which corresponds in Figure 5.11 to parts A and B, lines 7-9 and 19-24, respectively. In addition, in Figure

5.10, the role of Physician in section B corresponds to part C, lines 26-31 of Figure 5.11, which is part of the 'simple-DAC-delegation-rule'. Note that the high-level target of the policy in lines 6-16 of Figure 5.11 that has the user Elisa with identifier 6 in line 8 of Figure 5.11, establishes the linkage to all for a delegation rule that states she can delegate the role Physician from section B of Figure 5.10 in lines 26-31 of Figure 5.11 to either Samantha or Emily (section C of Figure 5.10), with the user Samantha chosen with identifier 30 in part D, line 34 of Figure 5.11, denoting that she is a user part of the <DelegationTargets> element. Note that, as mentioned in the introduction of Section 5.3.3, the <DelegationTargets> element is a non-normative component of XACML. We chose to extend the schema with this element to make the logic of delegating roles more streamlined.

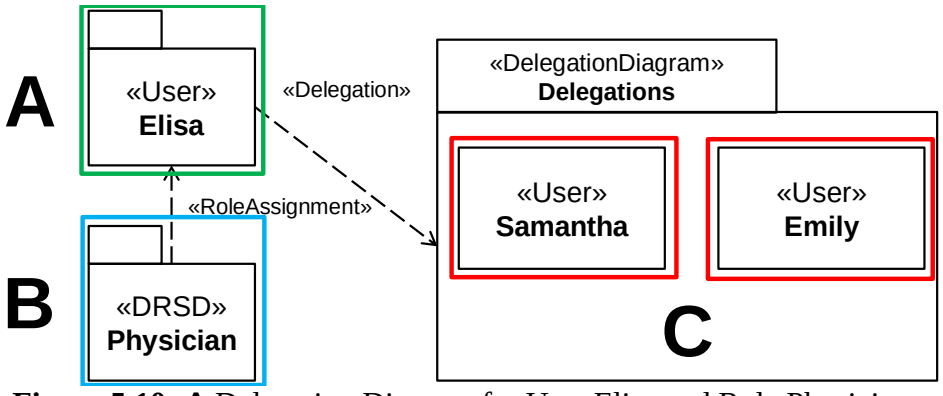

Figure 5.10: A Delegation Diagram for User Elisa and Role Physician.

```
1.  <Policy PolicyId="ada-example-dac-policy"
2.   RuleCombiningAlgId="deny-overrides">
3.   <Description>This is a pseudocode example of an
4.    XACML policy with DAC capabilities for user
5.    Elisa</Description>
6.    <Target>
7.      <Subjects>
8.       <user><id>6</id><name>Elisa</name></user>
9.      </Subjects>
10.     <Resources>
11.       <AnyResource/>
12.     </Resources>
13.     <Actions>
14.       <AnyAction/>
15.     </Actions>
16.   </Target>
17.   <Rule RuleId="simple-DAC-delegation-rule" Effect="Permit">
18.     <Target>
19.       <Subjects>
20.        <user>
21.          <id>6</id>
22.          <name>Elisa</name>
23.        </user>
24.       </Subjects>
25.       <Resources>
26.         <Roles>
27.           <role>
28.             <roleID>2</roleID>
29.             <roleName>Physician</roleName>
30.           </role>
31.         </Roles>
32.       </Resources>
33.       <DelegationTargets>
34.         <user><id>30</id><name>Samantha</name></user>
35.       </DelegationTargets>
36.     </Target>
37.   </Rule>
38. </Policy>
```

A

B

C

D

Figure 5.11: Pseudo-code for XACML Policy with Delegation Capabilities.

Authorization follows the delegation process, discussed in Section 3.6 with Definitions 35 and 36, to be defined over schemas and instances assigned to users. This requires the definition of authorizations in an XACML policy similar to assigning a user a clearance. To generate authorization related code for an XACML policy, we leverage AD from Section 4.2.7 and the <Resources> subtree of the <Rule> structure of XACML. In this case, the <Subject> subtree in the higher-level *Policy* <Target> is the user with his/her identifier and name. The resources under the rule would include the schema and instance identifiers as discussed in Section 3.1 with Definition 3, and represented as the new UML DD in Section 4.2.7.

The mapping statements for authorizations are shown in Table 5.4, and include: a *subject-user authorization mapping statement (SUA-MS)* that takes a user such as Elisa and a role such as Physician to a <Subject> in XACML; and, a *resources schemas-instances mapping statement (RSI-MS)* that takes a schema and/or instance such as CDA, CCR, or Carol Smith's record from Section 2.6 to a <Resource> in XACML. As discussed in Section 5.1 and shown using the long dash dot-dot arrows in Figure 5.1, the mappings for authorizations utilize the DSCD, DRSD, UD, and AD from Chapter 4. For the *SUA-MS*, the UD, DRSD, and AD are utilized to set the XACML policy's <Target> Subject element as the user and user identifier (using SUA-MS1) and the respective role and role identifier (using SUA-MS2). The SubjectMatch's MatchId attribute uses the function 'string-equal' to evaluate the user's role as modeled in the DRSD (using SUA-MS3). Finally, the *Subject* element in the *Rule*'s <Target> subtree is the user from the UD and AD (using SUA-MS4. For the *RSI-MS*, the AD and DSCD are utilized to set the schemas and instances that are to be authorized for the user/role combination in XACML form. The policy's <Target> Resource element is set to <AnyResource/> to ensure that the higher-level policies apply to the user (using RSI-MS1). The *Rule's* <Resource> children are <Schemas> or <Instance> nodes with element data that correspond to the element classes in the DSCD and AD diagrams (using RSI-MS2).

Mapping Statement Type	Involved UML Diagram(s)	Mapping Statements
SUA-MS **Subject (User) Mapping Statements**	UD, DRSD, AD	**SUA-MS1.** XACML Policy's <Target> Subject is the user and user identifier set as a <user> subtree with <name> and <id> children that corresponds to the «User» package of the AD. **SUA-MS2.** XACML Policy's <Target> *Subject* is the role and role identifier set as a <role> subtree with <roleName> and <roleID> children that corresponds to the DRSD package name. **SUA-MS3.** SubjectMatch's MatchId uses the function "string-equal" to evaluate the user's name and id as modeled in the AD and the security model. **SUA-MS4.** Subject in the Authorization Rule is set as the Subject in the higher-level <Target> (user), the «User» package from the AD.
RSI-MS **Resources (Schemas and Instances) Mapping Statements**	AD, DSCD	**RSI-MS1.** XACML Policy's <Target> Resource set to <AnyResource /> to ensure that the higher-level policies applies to the user. **RSI-MS2.** Rule's <Resources> children are <Schemas> and <Instances> elements that correspond to the with the «DSCD» and «Instance» components of the «AuthorizationDiagram» package.

Table 5.4: Authorization Mapping Statements.

In the final step for the example of XACML generation, we reintroduce the AD from Section 4.2.7 in Figure 5.12 for user Elisa under the role of Physician who is being authorized to Carol Smith's instance (and relevant schemas), with the corresponding generated XACML code from DSCD, DRSD, UD, and AD is shown in Figure 5.13. In this policy, the user Elisa from Figure 5.10 in section A corresponds to section A in Figure 5.12, has the identifier 6 that leads to the subject definition in part A, lines 7-10 of Figure 5.13 that correspond to parts A and B of Figure 5.12. In the example, Elisa is authorized by the rule with identifier "simple-DAC-rule" to the CCR schema in part C, lines 24-29 of Figure 5.13, that corresponds to the section C of Figure 5.12, as well as to Carol Smith

instance in part D, lines 30-35 of Figure 5.13, that corresponds to section D of Figure 5.12.

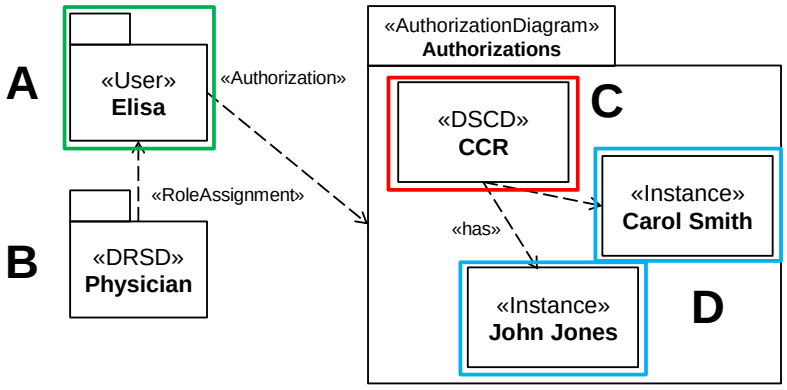

Figure 5.12: Authorization Diagram for User Elisa.

```
1.  <Policy PolicyId="ada-example-dac-policy"
2.   RuleCombiningAlgId="deny-overrides">
3.   <Description>This is a pseudocode example of an
4.    XACML policy with DAC capabilities for user
5.    Elisa</Description>                                    A
6.    <Target>
7.      <Subjects>
8.        <user><id>6</id><name>Elisa</name></user>
9.        <role><roleID>5</roleID><roleName>Physician</roleName></role>
10.     </Subjects>
11.     <Resources>
12.       <AnyResource/>
13.     </Resources>
14.     <Actions>
15.       <AnyAction/>
16.     </Actions>
17.   </Target>
18.   <Rule RuleId="simple-DAC-authorization-rule" Effect="Permit">
19.    <Target>
20.      <Subjects>
21.       <user><id>6</id><name>Elisa</name></user>          B
22.      </Subjects>
23.      <Resources>
24.       <Schemas>
25.         <schema>
26.           <schemaID>sId1</schemaID>
27.           <schemaName>CCR</schemaName>                   C
28.         </schema>
29.       </Schemas>
30.       <Instances>
31.         <instance>
32.           <instanceID>iId1</instaneID>
33.           <instanceName>Carol Smith</instanceName>       D
34.         </instance>
35.       </Instances>
36.      </Resources>
37.    </Target>
38.   </Rule>
39. </Policy>
```

Figure 5.13: Pseudo-code for XACML Policy with Authorization Capabilities.

5.3.4 Interplay of RBAC, LBAC and DAC

Sections 5.3.1, 5.3.2 and 5.3.3 detailed the mapping statements to support RBAC, LBAC, or DAC individually in XACML. When an application is to be comprised with capabilities from all three of these access control models, a comparison must be performed between users, clearances, roles, permissions, elements targeted by the permissions, the element's sensitivity levels, and authorizations in order to determine if the combination has any potential conflicts or inconsistencies. This interplay supports the authorization of a user as given in Definition 6 in Section 3.6 that involves a user with a role, clearance, and delegation. To allow the definition of policies that support multiple access control models, the security definitions for RBAC, LBAC, and DAC, must be compared with one another. From the perspective of XACML, one approach to accomplish this comparison would be to list all of the permitted rules for the user/role pair (including rules for RBAC and LBAC), and endeavor to determine if there are conflicts in terms of access among the permitted rules. The result of this would produce a set of security policies, one for each permission in the security definitions for the user. The problem with this approach occurs when there is a large amount of users which results in a large set of security policy instances that would be difficult to manage properly. While easies to implement, it is difficult to manage due to numerous rules across numerous policies which could result in poor performance when fetching the relevant policies affecting one user in one scenario. A second approach to support the integration of RBAC, LBAC, and DAC leverages the <Condition> element of the XACML 3.0 schema, which is used to further augment the form of the <Rule> element's <Target> by presenting the capability of comparing two or more attributes with a set of

normative functions (e.g. integer-greater-than-or-equal, etc.).. The intent is to pair a <Condition> element with each rule as needed in order to perform necessary consistency checks. Utilizing the <Condition> element for each rule could allow the LBAC requirements that govern operations over elements to be realized as an extra component of an already existing RBAC rule. Another potential benefit of leveraging the <Condition> element is to more easily support LBAC read and write features such as simple-integrity, simple-security, liberal-*, etc. as discussed in Section 3.4.

To illustrate this second approach, Figure 5.14 has an example of an XACML condition that would be part of a rule for the example of user Elisa and role Physician. The <Condition> XACML code segment results from the transformation of the mapping statements of Section 5.3.2 to support the enforcement of LBAC clearance vs. classification dominance. The first step of this transformation searches those permissions in RBAC that match the <Action> and <Resources> elements, and then generates the <Condition> logic to substitute for the LBAC logic of Section 5.3.2. This condition follows the logic of simple-integrity, or write-down and no write-up (see Section 3.4). This means that a user can write elements with a lower sensitivity level when compared to their clearance level, but would not be allowed to write those elements with a higher sensitivity. The first <Apply> in line 2 of Figure 5.14 utilizes the comparator integer-greater-than-or-equal, which takes two elements as parameters. The first is the attribute that denotes the user's clearance as shown in lines 4-6. The second attribute denotes the element's sensitivity in lines 8-10. If this condition were evaluated to be true, which means that the clearance is greater than or equal to the sensitivity by the function from line 2, then the condition would return true and the rule would be permitted.

The end result of utilizing the <Condition> element is the creation of only one policy-per-user as contrasted with the first approach discussed above. This is shown in Figure 5.16 where for the user Elisa, there are groups of each of the RBAC permissions under one rule each, and injects the necessary XACML logic in the form of <Condition> when there is a need to support LBAC as shown in line 18. The DAC delegations and authorizations in lines 23-34 in Figure 5.16, live at the policy-level and are separate rules of the policy. This procedure allows a security architecture to only fetch one policy when handling the security of one user. As a result, this one policy with respect to a user realizes Defn. 6 of Section 3.6 by including all of the necessary security requirements to only one object, resulting in all of the security requirements are included in the user object, which is translated in XACML to only one policy per user. Following the scenario of Section 2.6, for users Brock Ketchum, Elisa Fakington, Leroy, Jenkins, and Gail (5 users in total), the result of the policy mapping process would produce 5 XACML policy instances, one for Brock, one for Elisa, etc.

```
1.  <Condition>
2.   <Apply FunctionId="…:integer-greater-than-or-equal">
3.   <Apply FunctionId="…:integer-one-and-only">
4.   <AttributeValue DataType="…#integer">
5.   {userClearance}
6.   </AttributeValue>
7.   </Apply>
8.   <AttributeValue DataType="…#integer">
9.   {elementSensitivity}
10.  </AttributeValue>
11.  </Apply>
12. </Condition>
```

Figure 5.14: XACML Condition Pseudo-code for the LBAC Component of an RBAC + LBAC Rule.

5.4 Algorithm for Automatic Generation of XACML

The process of generating an XACML policy from the seven new UML diagrams (DSCD, SID, DRSD, LSID, UD, DD, and AD) of Chapter 4 can be automated with an

algorithm, as shown by Figure 5.15. The new UML diagrams along with the document schemas serve as the parameters, while the XACML schema is utilized as template for the resulting instances. While the mapping statements across the access control models (RBAC, LBAC, DAC) do not require any predetermined order to be applied (for example, DAC delegation and authorization mapping statements could be applied before RBAC mapping statements), for the purpose of automation, we prioritize some aspects of the process. From a high-level perspective, as shown in Figure 5.15, the first step is to iterate over every user of the information system that requires security. Once a user is selected (e.g., Elisa from the healthcare scenario of Section 2.6), the next step is to find that user's role and the respective DRSD that describes all of the permissions over every element. Then, by iterating over every permission in the relevant DRSD, the algorithm creates an XACML <Rule> object that would map the role to the <Subject>, the elements to the <Resources>, and the permissions (operations) to the <Actions>. Then, after that initial mapping is done, a check for LBAC features is done. If any LBAC features exist, such as simple-security, simple-integrity, etc., a <Condition> element is added to that rule. This process is repeated over every permission, resulting in one <Rule > with a <Condition> element if LBAC is needed (see Figure 5.14 again) for each permission in the DRSD. This iteration is repeated for every role the user might hold. After the mappings over RBAC and LBAC capabilities are complete, then delegations and authorizations are tackled. For each delegation and authorization, a <Rule> element is created that would map the schemas and instances to <Resources> inside the rule for authorization, or roles and delegable users to <Resources> and <DelegationTargets>

respectively, for delegation. The end result of this high-level process is the creation of one XACML policy instance per user, which could be readily deployed.

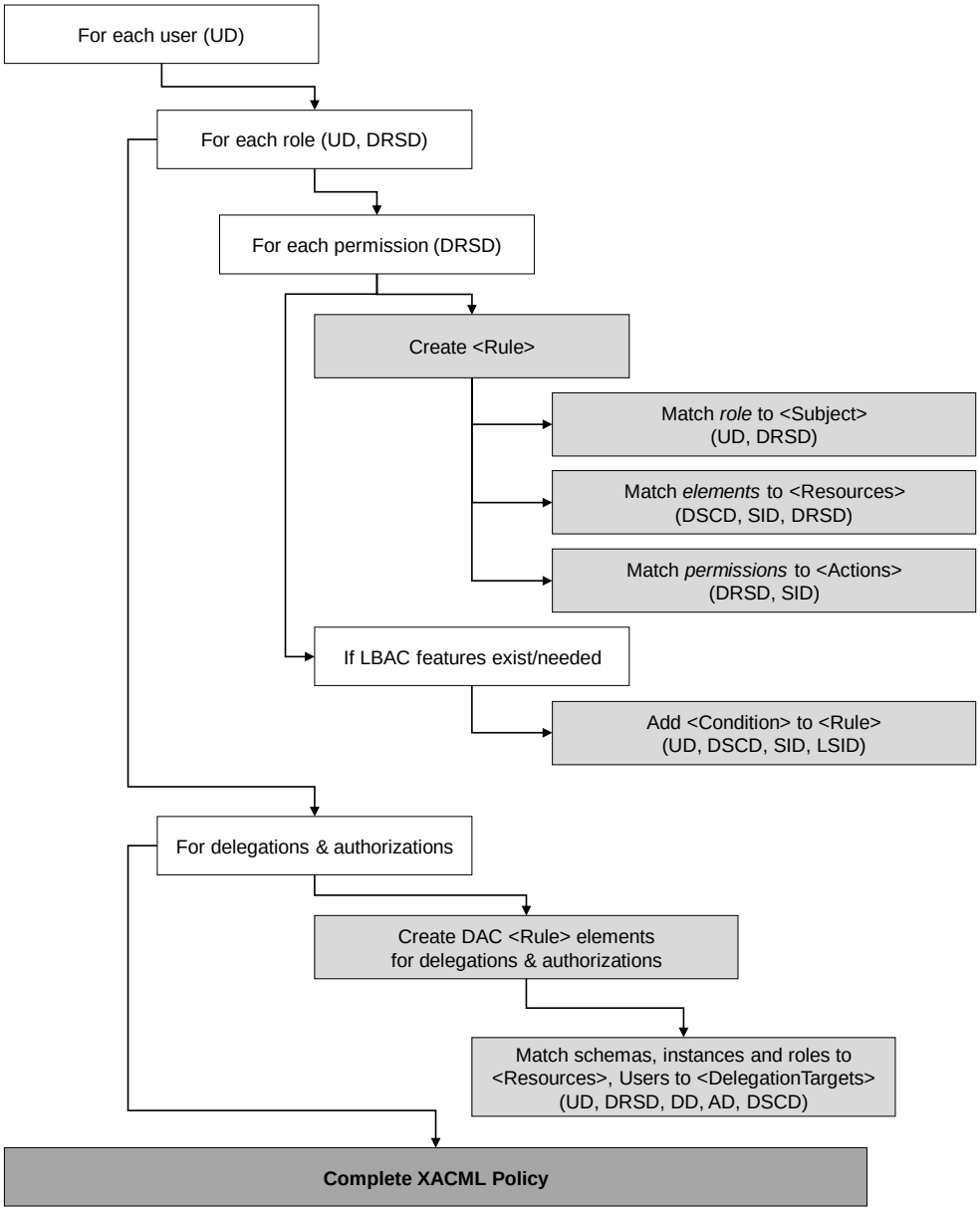

Figure 5.15: High-level Algorithm for XACML Generation from UML Extensions.

The high-level algorithm of Figure 5.15 can be transitioned to a more detailed pseudo code version as given in for the automatic generation of XACML that leverages the UML extensions of Chapter 4 and the mapping statements of Sections 5.3.1, 5.3.2, 5.3.3, and 5.3.4. The first step, as shown in line 3 of Figure 5.16 is to generate the XACML

description header as discussed in the introduction of Section 5.3. This process involves generating the PolicyID attribute of the <Policy> and the <Description> content of the <Policy>. The second step in line 4 of Figure 5.16 corresponding to the first step in Figure 5.15, involves a loop over each User in the system. For each user (line 4 of Figure 5.16), the roles that are tied to that user are identified (line 6 of Figure 5.16) utilizing the UD and DRSD. After this list of roles has been found, an iteration over the roles (line 7 of Figure 5.16) is performed. Following a similar procedure as before, the list of permissions tied to that role is fetched using the DRSD (line 9 of Figure 5.16). Following this, a loop over each of the permissions tied to the role denoted by DRSD (line 10 of Figure 5.16) is performed. The first step inside this loop is creating a <Rule> element (line 12 of Figure 5.16), followed by the three mappings. First, a map of the <Subject> is performed (line 13 of Figure 5.16) utilizing the R-MS from Section Table 5.1. Then, a map of the <Resources> is performed (line 14 of Figure 5.16) utilizing the E-MS from Table 5.1. The last step inside this loop maps the <Actions> (line 15 of Figure 5.16) utilizing the P-MS from Table 5.1. After the RBAC permission mapping segment of the algorithm is complete (lines 12-15 of Figure 5.16), a check is performed if LBAC support is desired (line 16 of Figure 5.16). If true, the <Condition> element is created (line 18 of Figure 5.16) by utilizing the UD, DSCD, SID, and LSID and following the SU-MS, AO-MS, and RE-MS of Table 5.2, followed by the transformation to the <Condition> element discussed in Section 5.3.4.

After the loops over permissions and roles are complete, a check for delegations in line 23 of Figure 5.16 is performed. For each of the delegation rules (line 23 of Figure 5.16), a <Rule> is created (line 25 of Figure 5.16), followed by a <Resource> map using

the DRSD and DD as discussed with the DRR-MS of Table 5.3. Then, the delegation targets are mapped (line 27 of Figure 5.16) utilizing the UD, DRSD, and DD with the DTDU-MS of Table 5.3. Once the loop over delegations is completed, a loop over authorizations starts (line 29 of Figure 5.16). The first step inside this loop creates a <Rule> (line 31 of Figure 5.16) followed by a <Subject> map utilizing the SUA-MS of Table 5.4 with the UD and AD, and ends with the <Resources> map utilizing the RSI-MS of Table 5.4 with the AD and DSCD.

The result of executing the algorithm in Figure 5.16 is an instance of an XACML <Policy> as shown in Figure 5.17 for user Elisa (line 5, 37 and 52 of Figure 5.17) and role Physician (line 13 and 41 of Figure 5.17) with the security requirements defined in the UML extensions of Chapter 4 that are built upon the model of Chapter 3 and created from the healthcare scenario of Section 2.6. In this policy, RBAC and LBAC features are found under the same rule in lines 10 to 32 of Figure 5.17. Note that this policy applies the function greater than or equal to figure out whether the condition under the insert/write rule is allowed as given in line 21 of Figure 5.17. Since the clearance of user Elisa (line 27 of Figure 5.17) and the classification of Past Medical History (line 29 of Figure 5.17) are equal, the condition is valid. The resulting enforcement would then depend on whether the schema and/or instance in which the operation is being tried on is authorized. Delegations and authorizations, from lines 34 to 63 in Figure 5.16, are translated as is from the mapping statements since they have no effect on the document-level operations. That is, all of the authorizations for the user Elisa (lines 49 to 63 of Figure 5.17) and her delegation of roles to users such as Samantha (lines 34 to 48 of Figure 5.16) have no impact on the decision of whether the insert operation (line 20 of

Figure 5.17) is permitted or not. While RBAC and LBAC are orthogonal, the policy follows the nature of DAC delegations and authorizations being in a different layer of the access control model. Note that not all of the permissions of the Physician DRSD are represented in this policy due to space reasons. As discussed earlier in this section, each permission would be represented as a <Rule> element in the XACML Policy instance. The complete policy for the user Elisa and role of Physician can be found in Appendix A.

```
1.  RBAC_LBAC_DAC_XACML_generation(DSCD, SID, DRSD, LSID, UD, DD, AD)
2.  {
3.    Generate_XACML_Description_Header() // lines 1-2 of Fig. 5.17
4.    foreach(User as currentUser)
5.    {
6.      role_list = Find_Role(UD, DRSD);
7.      foreach(role_list as currentRole)
8.      {
9.        permission_list = Find_permissions(DRSD);
10.       foreach(permission_list as currentPermission)
11.       {
12.         XACML.createRule(); // lines 10-36 of Fig. 5.17
13.         XACML.mapSubject(UD,DRSD); // lines 12-14 of Fig. 5.17
14.         XACML.mapResources(DSCD,SID,DRSD); // lines 15-18 of Fig. 5.17
15.         XACML.mapActions(DRSD,SID); // lines 19-22 of Fig. 5.17
16.         if(LBAC)
17.         {
18.           XACML.createCondition(UD,DSCD,SID,LSID); // lines 24-35 of Fig. 5.17
19.         }
20.       }
21.     }
22.   }
23.   foreach(Delegation)
24.   {
25.     XACML.createRule(); // lines 37-51 of Fig. 5.17
26.     XACML.mapResources(DRSD,DD); // lines 42-46 of Fig. 5.17
27.     XACML.mapTargets(UD,DRSD,DD); // lines 47-49 of Fig. 5.17
28.   }
29.   foreach(Authorization)
30.   {
31.     XACML.createRule(); // lines 52-66 of Fig. 5.17
32.     XACML.mapSubject(UD,AD); // lines 54-56 of Fig. 5.17
33.     XACML.mapResources(AD,DSCD); // lines 57-64 of Fig. 5.17
34.   }
35. }
```

Figure 5.16: Pseudo-code for XACML Policy Instance Generation Algorithm.

```
1.  <Policy PolicyId="ada-policy" RuleCombiningAlgId="deny-overrides">
2.  <Description>Omitted due to length.</Description>
3.  <Target>
4.   <Subjects>
5.   <user><id>6</id><name>Elisa</name></user>
6.   </Subjects>
7.   <Resources><AnyResource/></Resources>
8.   <Actions><AnyAction/></Actions>
9.  </Target>
10. <Rule RuleId="simple-RBAC+LBAC-rule" Effect="Permit">
11.  <Target>
12.  <Subjects>
13.  <role><roleID>5</roleID><roleName>Physician</roleName></role>
14.  </Subjects>
15.  <Resources><element>
16.  <elementID>el-3</elementID>
17.  <elementName>Past Medical History</elementName>
18.  </element></Resources>
19.  <Actions><operation>
20.  <operationName>insert</operationName>
21.  <opAccessMode>write</opAccessMode>
22.  </operation></Actions>
23.  </Target>
24.  <Condition>
25.  <Apply FunctionId="…:integer-greater-than-or-equal">
26.  <Apply FunctionId="…:integer-one-and-only">
27.  <AttributeValue DataType="…#integer">Secret</AttributeValue>
28.  </Apply>
29.  <AttributeValue DataType="…#integer">Secret</AttributeValue>
30.  </Apply>
31.  </Condition>
32. </Rule>
33. … // Remainder of permissions omitted due to space
34. <Rule RuleId="simple-delegation-rule" Effect="Permit">
35.  <Target>
36.  <Subjects>
37.  <user><id>6</id><name>Elisa</name></user>
38.  </Subjects>
39.  <Resources>
40.  <Roles><role>
41.  <roleID>2</roleID><roleName>Physician</roleName>
42.  </role></Roles>
43.  </Resources>
44.  <DelegationTargets>
45.  <user><id>30</id><name>Samantha</name></user>
46.  </DelegationTargets>
47.  </Target>
48. </Rule>
49. <Rule RuleId="simple-authorization-rule" Effect="Permit">
50.  <Target>
51.  <Subjects>
52.  <user><id>6</id><name>Elisa</name></user>
53.  </Subjects>
54.  <Resources><Schemas><schema>
55.  <schemaID>4</schemaID>
56.  <schemaName>Schema 4</schemaName>
57.  </schema></Schemas>
58.  <Instances><instance>
59.  <instanceID>4,2</instaneID>
60.  <instanceName>Carol Smith Health Record</instanceName>
61.  </instance></Instances></Resources>
62.  </Target>
63. </Rule>
64. </Policy>
```

Figure 5.17: Resulting XACML Policy for User Elisa and Role Physician.

5.5 Related Research in Policy Generation and Integration

In this chapter we presented a way of generating XACML that effectively integrated capabilities from LBAC, RBAC and DAC. Related work on this area of research usually focuses in integrating existing policies into one. 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 DTDs in order to provide customizable security. This provides a level of generalization for documents that share the same DTD, similar to our work where security policies act against XML schemas to control XML instances. Two differences are: their work targets outdated XML DTD's while ours utilizes schemas, and their polices are embedded into both DTD and instance, requiring changes to instances when policies change; our work allows changes with no impact on 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. While this work is similar to our work by targeting security in XML instances via policies, it differs by requiring changes to instance when the policy is modified and does not consider XML document writing (see Section 5.3). Efforts by (Bertino & Ferrari, 2002; Bertino, Carminati, & Ferrari, 2004) present Author-X, a Java-based system for DAC in XML documents that provides customizable protection to the documents with positive and negative authorizations. Author-X employs a policy-based DTD document that prunes an XML instance based on the security policies, which is similar to our approach, but focuses on discretionary access control where we focus on RBAC. The work of (Leonardi

et al., 2010) considers the scenario of a federated access control model, in which the data provider and policy enforcement are handled by different organizations. This approach relates to ours with regards to the separation of the security policies from the data to be handled, but differs in the specifics of where the policies' details are stored.

The work of (Kuper et al., 2005) has presented a model consisting of access control policies over outmoded DTD's with XPath expressions to achieve XML security. Their model is similar to ours, as it aims to provide different authorized views of an XML document based on the user's credentials. However, the significant difference is that this approach combines query rewriting and authentication methods, whereas our approach can be applied to any non-normative XACML architecture (having a policy enforcement point) for both reading and updating, as well as XPath or XQuery queries. The work of (Müldner et al., 2009) presents an approach of supporting RBAC to handle the special case of role proliferation, which is an administrative issue that happens in RBAC when roles are changed, added, and evolve over time, making security of an organization difficult to manage. Our approach doesn't address role proliferation; however, by separating our security into an XACML policy, we do insulate role proliferation from impacting an application's XML schemas and instances.

Policy integration approaches vary from similarity finding to specialized algebraic approaches. For example, (Mazzoleni, Bertino, Crispo, & Sivasubramanian, 2006; Mazzoleni, Crispo, Sivasubramanian, & Bertino, 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. Another approach (Rao, Lin, Bertino, Li, & Lobo, 2009) proposes an algebraic solution

to the problem to integrate complex security policies at a granular detail with an algebra that consists of five unary operations (three binary and two unary). This algebra is utilized as part of a framework (to achieve the generation of an instance policy automatically. Like (Mazzoleni et al., 2008), this approach also focuses on the instance level, creating a dependency on the OASIS XACML specification and policy structure to achieve proper results. The use of these methods would be impossible on security requirements defined in any other method (e.g., a database table with security rules, policies modeled with a different language, etc.).

Chapter 6
Secure Information Engineering Process and Enforcement with Mobile Apps

In this chapter, we discuss the *secure information engineering process (SIEP)* that leverages the security model presented in Chapter 3, the seven new UML diagram and respective metamodel extensions presented in Chapter 4, and the policy mapping process presented in Chapter 5. Over the past five years, one major focus of our research group has been on extending UML with new diagrams to realize a secure software engineering process for RBAC, MAC, and DAC, shown in Figure 6.1 (Pavlich-Mariscal et al., 2014) using separate concerns for functional, collaboration, and information application characteristics. First, from a *functional perspective* focused on object-oriented design, a framework of composable security features was defined (Pavlich-Mariscal et al., 2010) that preserves separation of security concerns from models to code through the extension of UML with new diagrams for RBAC, DAC, and MAC coupled with the automatic generation of enforcement code in AspectJ (Pavlich-Mariscal et al., 2010). Second, from a *collaboration perspective*, a framework for secure, obligated, coordinated, and dynamic collaboration was developed (Berhe et al., 2010) that extended NIST RBAC to allow for the definition and enforcement of security with new UML diagrams for collaborative RBAC applicable to situations when individuals need to interact with one another in certain ways to achieve a common goal. Third, from an *information perspective*, the work presented in this dissertation has focused on the definition of a modeling and design framework with enforcement process for information-based applications.

The objective of this chapter is to present a secure information engineering process for security and/or software engineers concerned with information security can follow to provide security assurance for the information that must be both modeled and protected. In order to achieve SIEP the underlying security model from Chapter 3 creates the logical base in which the seven new UML diagrams of Chapter 4 are defined from, with the policy generation process of Chapter 5 utilized to produce enforcement policies for the new UML diagrams that target information schemas that can be readily deployed and used for enforcement.

The remainder of this chapter is organized into three major sections. Section 6.1 covers the secure software engineering work that we published that considered functional, collaborative and information security concerns in combination (Pavlich-Mariscal et al., 2014) and serves as the basis of the SIEP in this chapter. Then, in Section 6.2, the SEIP is presented which utilizes the security model from Chapter 3, the UML extensions of Chapter 4, and the policy generation process from Chapter 5 for a design and development cycle for information security. Finally, in Section 6.3, we complete the discussion using the prototype mobile application tailored for the healthcare domain, from Section 2.6, which includes a high-level enforcement architecture in and its realization in our mobile personal health assistant (PHA) application.

6.1 Secure Software Engineering

Figure 6.1 details a secure software engineering process for functional, collaborative, and information modeling and design. From a functional perspective, (Pavlich-Mariscal et al., 2010) extended UML to represent RBAC, DAC, and MAC (see upper portion of Figure 6.1) via the introduction of the *Role Slice Diagram*, the *User Diagram*, the

Delegation Diagram, and *MAC extensions* coupled with a *Secure Subsystem Diagram* (middle right hand side of Figure 6.1). The *Secure Subsystem Diagram* denotes the subset of an application's overall classes and methods that are restricted and require permissions to be in place for authorized users, as we previously discussed in Section 4.2.2. The *Role Slice Diagram* denotes RBAC policies, providing the role slice, a stereotyped package that represents the permissions assigned to a role. A role slice uses method-based permissions to allow or deny users to access specific operations, regardless of the object to which it is applied. The *Delegation Diagram* can be utilized to represent all of the rules of delegation between roles. This diagram provides the *delegation slice*, a stereotyped package that contains all of the roles that a user can delegate authority to another user, who may also be allowed to further delegate the role. The *User Diagram* has stereotyped packages to denote users and stereotyped dependency relations to represent user-role assignments. MAC extensions enhance the previous three diagrams with sensitivity levels (e.g., confidential, secret, top secret) and their ordering relations to indicate classifications of methods, clearances of role slices, and, implicitly, to declare access constraints based in the relation between classifications and clearances. From an enforcement perspective, once defined, the diagrams are utilized to generate aspect-oriented enforcement code in AspectJ (bottom portion of Figure 6.1) that is able to verify, at runtime, whether the active user has a role with permissions over the protected method and grants or denies access accordingly. The end result is that aspects can effectively modularize access control concerns and enhance traceability from design to code.

From a collaborative perspective, (Berhe et al., 2010) (lower middle left of Figure 6.1) has focused on the extension of RBAC to define collaboration and sharing

capabilities across a workflow. Collaborative computing has emerged in many domains, with users interacting with one another towards some common goal. For example, in a health care setting, a patient's many providers (e.g., internist, cardiologist, physical therapist, etc.) need to interact with one another against a common set of data (patient's medical record). Unlike traditional security that defines separation of duty and mutual exclusion to prohibit what users can do, in a collaborative setting, the key is on defining when and how users collaborate. Thus, the work extended RBAC with a set of UML diagrams for collaboration on duty and adaptive workflow (Berhe et al., 2010) that interacts with our functional extensions in the top of Figure 6.1 (Pavlich-Mariscal et al., 2010): the *Extended Role Slice Diagram*, the *Team Slice Diagram*, the *Workflow Slice Diagram*, and the *Obligation Slice Diagram*. The *Extended Role Slice Diagram* defines the roles and privileges for each user within each collaboration step. The *Team Slice Diagram* defines the team members and their participation in the various collaboration steps. The *Workflow Slice Diagram* defines the steps and connections among them for a given team and specific collaboration. Lastly, the *Obligation Slice Diagram* defines the required permissions and participations for a particular collaboration. In addition, we provide the mapping of these new UML based collaboration design-time diagrams to actual machine-readable code-based policies for runtime enforcement.

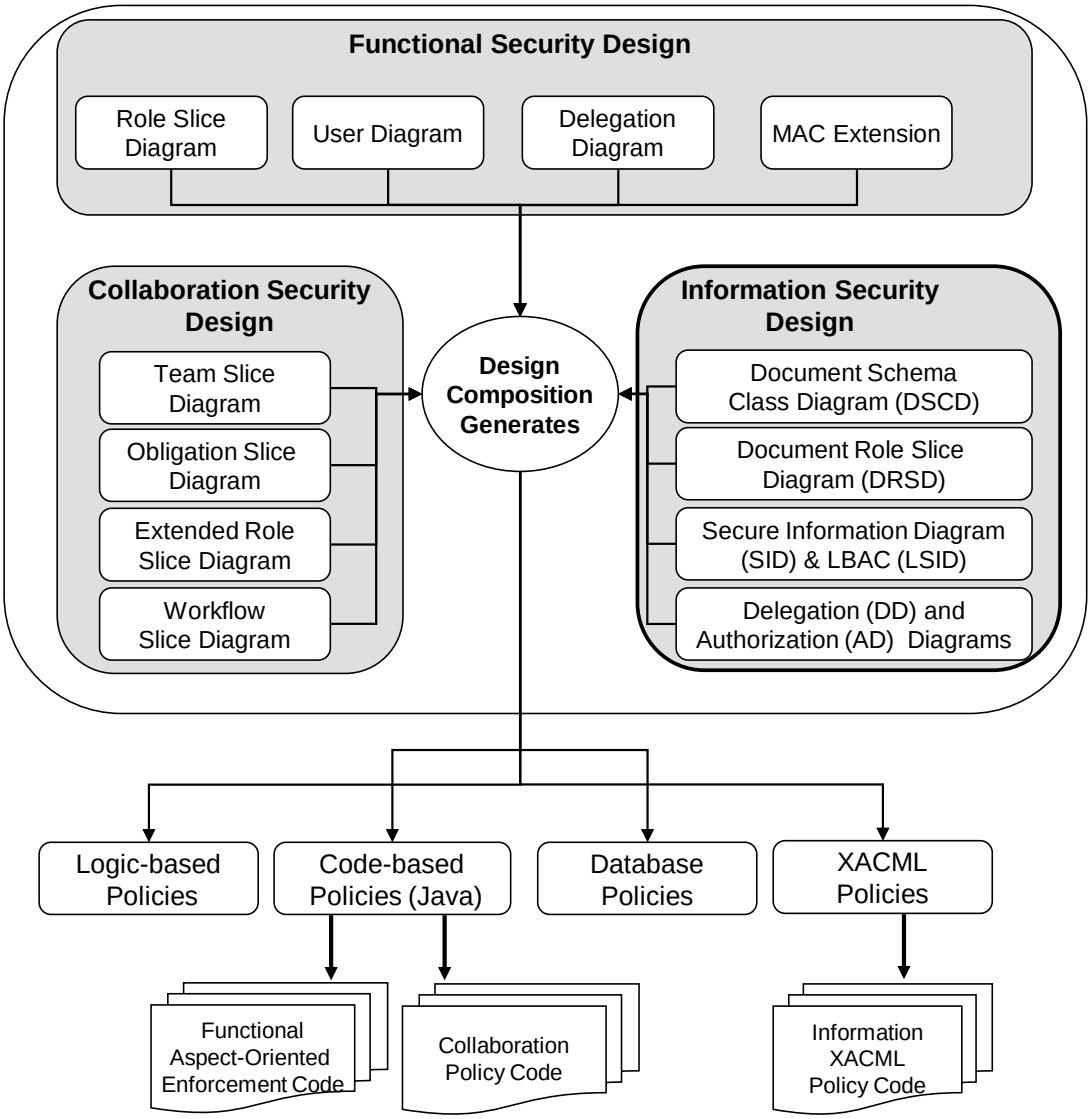

Figure 6.1: Secure Software Engineering.

Finally, from an information perspective, (De la Rosa Algarín, A. et al., 2013) (middle-right of Figure 6.1) has emphasized the control of information created by one application to be shared and/or exchanged with other applications. One dominant approach for information exchange is the use of tree-structured documents, such as the eXtensible Markup Language (XML) (Bray et al., 1998). In the case of XML, defining XML schemas has become an integral part of the application development process to handle exchange form database to server, from server to end user, among different

databases, etc. In support of information-based security, we have extended UML with seven new diagrams: the *Document Schema Class Diagram (DSCD)*, the *Secure Information Diagram (SID)*, the *Document Role Slice Diagram (DRSD)*, the *LBAC Secure Information Diagram (LSID)*, the *User Diagram* (UD), the *Delegation Diagram (DD)*, and the *Authorization Diagram (AD)*. The *DSCD* models the original tree-structured schema as a UML diagram. The *SID* identifies the subset of elements from the original schema that require some sort of security definition. The *DRSD* introduces RBAC capabilities that target elements of the original schema. The *LSID* extends the *SID* with LBAC features such as classifications in elements. The UD models the users and their properties of the information system. The *DD* represents the role-delegation capabilities between original users and delegable users. Lastly, the *AD* models the authorized schemas and instances for a respective user/role pair. These seven new diagrams allow us to automatically generate enforcement policies with the eXtensible Access Control Markup Language (XACML) via a mapping process as presented in Chapter 5. In turn, these enforcement policies can be readily deployed into any security architecture that utilizes the XACML specification's processing model. With these seven new UML diagrams augmented with the policy mapping process, a software engineer can consider and produce security enforcement code that targets information content by modeling the original schema (producing the DSCD), augmenting it with security features with respect to the different roles and permissions (producing the SID, DRSD, LSID, and AD), and then automatically creating an enforcement policy with the mapping process (XACML).

6.2 The Secure Information Engineering Process (SIEP)

The secure information engineering process (SIEP) is intended to provide both security and/or software engineers with the ability to support RBAC, LBAC and DAC of tree structured documents integrated with the overall design, development, deployment, and maintenance of an information application, as shown in Figure 6.3, and consists of five main ordered tasks that are further divided into smaller sub-tasks. To help drive the discussion of Figure 6.3, we reintroduce all of the diagrams presented in Section 4.2 using Figure 6.2 where: diagram A is the DSCD from Figure 4.5 in Section 4.2.1; diagram B is the UD from Figure 4.9 in Section 4.2.5; diagram C is the SID from Figure 4.6 in Section 4.2.2; diagram D is the DRSD from Figure 4.7 in Section 4.2.3; diagram E is LSID from Figure 4.8 in Section 4.2.4; diagram F is the DD from Figure 4.10 in Section 4.2.6; and, diagram G denotes AD from Figure 4.11 in Section 4.2.7. In the remainder of this section, we explore SIEP in Figure 6.3 utilizing the diagrams (A to G) and explaining the steps and actions of a security and/or software engineer[1].

To begin, the first major step in SIEP labeled (1) in the top portion of Figure 6.3 is the design of the main security component of the application. This can include functional and collaborative application characteristics. such as those presented by (Pavlich-Mariscal et al., 2010) and (Berhe et al., 2011), and is primarily focused in the controlling access to the application programming interfaces (APIs) and their methods (functional) coupled with the definition of the detailed workflows of users and their interactions towards a particular task (collaboration) . The second major step in SIEP labeled (2) in Figure 6.3 is the initial information security design. For this step, an engineer defines a DSCD (see

[1] Note that from this point forward, the use of the term engineer refers to either a security engineer or software engineer that is involved with designing the information security of an application.

diagram A of Figure 6.2) for the tree-structured schema that the information application will be utilizing labeled (2.1) in Figure 6.3, and then defines the general information security requirements labeled (2) in Figure 6.3). These general information security requirements can include, but are not limited to, roles of users that will be utilizing the information presented by the system, their permissions, the user's clearance levels and the information's sensitivity, as well as delegation and authorization aspects of the security definitions. Step (2.2) acts as the catalyst for the refinement of initial version of the UD (see diagram B of Figure 6.2), for the users of the information system.

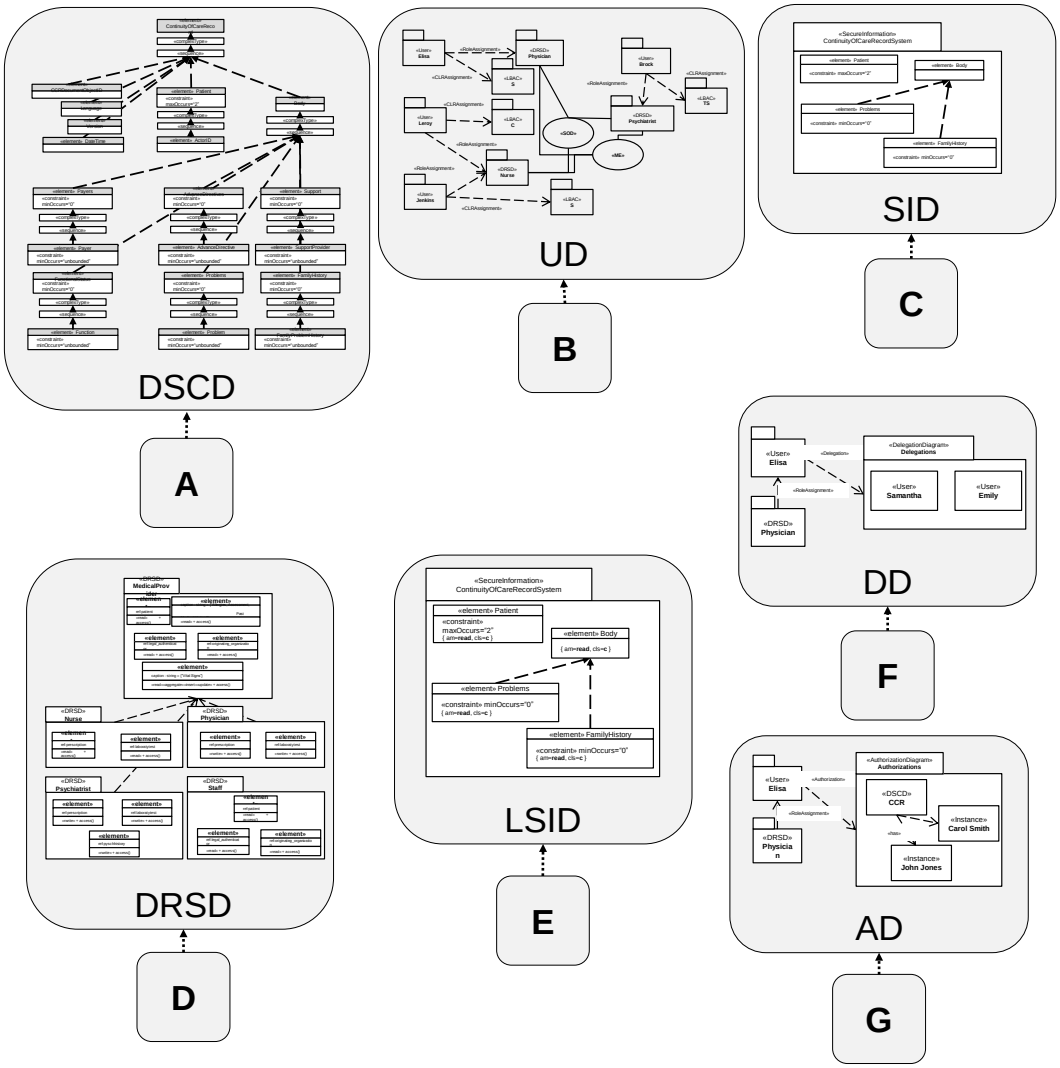

Figure 6.2: Diagrams Used through the Secure Information Engineering Process.

The third major step labeled (3) in Figure 6.3 provides and ability to define the different aspects of the information security design initially presented by the security model in Chapter 3 and the UML diagram extensions in Chapter 4. This initial step can define one or more SIDs (see diagram C of Figure 6.2) in Figure 6.3 labeled (3.1) that will identify the respective subsets of the DSCDs that require requires some level of security via the projection operation in Chapter 3, Definition 8. Step (3) in Figure 6.3 has three possible options for the engineer: the left path of Figure 6.3 labeled (3.2.a) and (3.2.b) for RBAC; the center path of Figure 6.3 labeled (3.3.a) and (3.3.b) for LBAC; and, the right path of Figure 6.3 labeled (3.4.a) and (3.4.b) for DAC and authorization. While all three paths as optional, the engineer must include one path as part of the SIEP associated with Step (3) in order to proceed through the step and terminate in step (3.5) for the UD definition in Figure 6.3, with the potential to loop back to Step (3.1) as the design is developed in iterations over time.

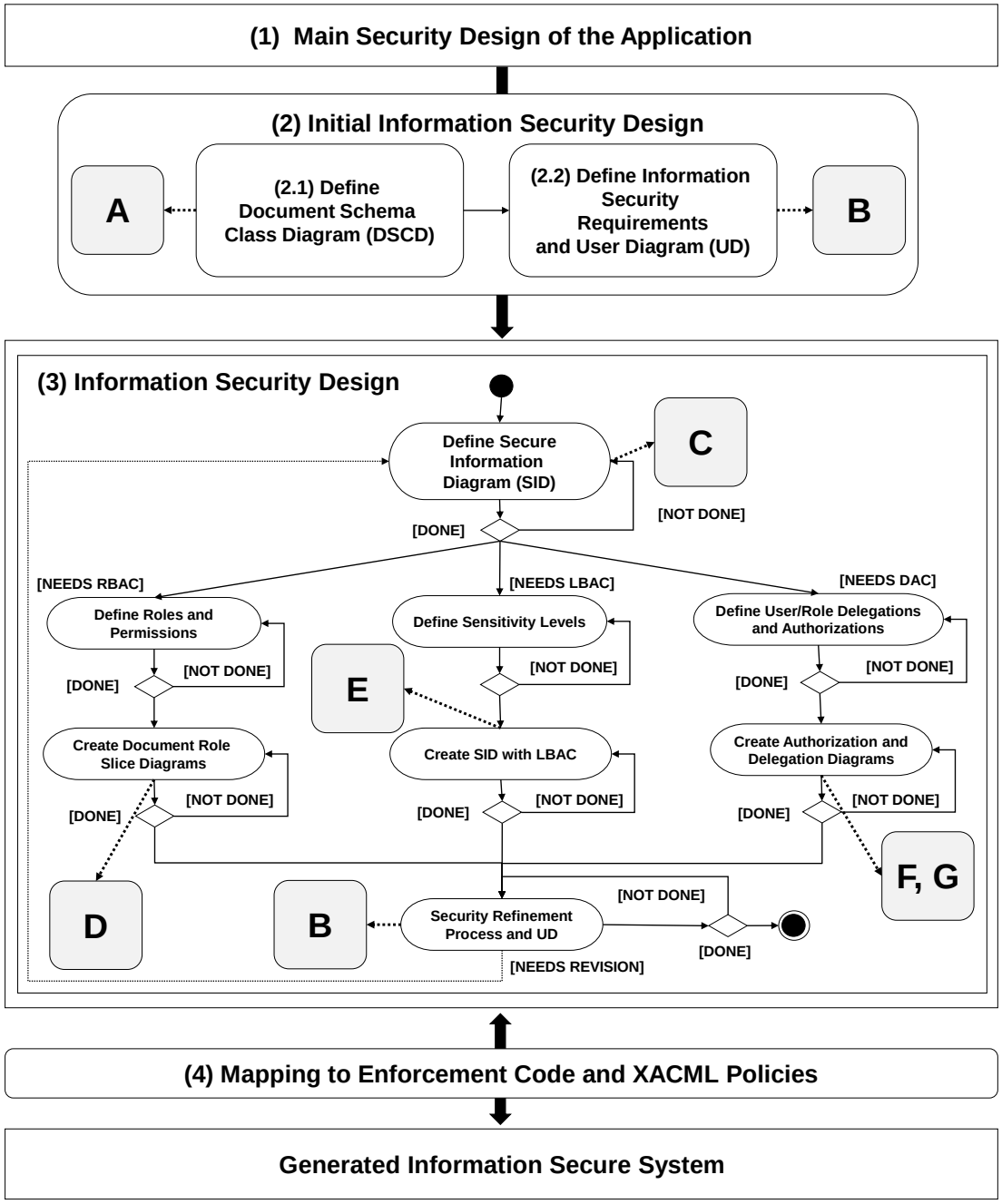

Figure 6.3: A Secure Information Engineering Process for RBAC, LBAC and DAC.

In the case of the left path in Figure 6.3 labeled (3.2.a) and (3.2.b) for RBAC capabilities, the engineer is able to begin to define roles and permissions as realized with the DRSD (see diagram D of Figure 6.2). The left path has feedback loops that allow the engineer to revisit the requirements and definitions of RBAC properties in order to

provide a robust security policy. In the case of the middle path in Figure 6.3 labeled (3.3.a) and (3.3.b), the LBAC requirements are realized by the engineer by first defining the sensitivity levels, which will act as the classification levels for elements and clearance levels for users (3.3.a) and by second creating the LSID (see diagram E of Figure 6.2) for the application's tree structured schema (3.3.b). The engineer can also use the LBAC feedback loops to polish these security requirements iteratively. . In the case of the right path in Figure 6.3 labeled (3.4.a) and (3.4.b), DAC and authorizations are defined by the engineer, first by creating the DD (see diagram F of Figure 6.2) and second by specifying the AD (see diagram G of Figure 6.2). Again, for this step, the engineer can loop back for revisions and subsequent iterations. Step (3.5) of Figure 6.3 allows the engineer to refine UD (see diagram B of Figure 6.2), where the users are tied to their RBAC roles and LBAC clearance levels. From Step (3.5), the engineer can either proceed to Step (4) or loop back to (3.1) in order to continue to iterate and create the design.

Once this step is complete, the engineer can then generate enforcement security policies (Step 4) by the mapping process presented in Chapter 5. After the process of defining roles, permissions, classification (sensitivity) levels for elements, authorizations and delegations is complete; an opportunity to further refine the security design follows looping back from (3.5) to (3.1) in Figure 6.3. Once the security design is properly refined, a direct mapping of the UML diagrams to enforcement policies in XACML is done in the major Step (5) in Figure 6.3, following the process presented in Chapter 5 with the mapping statements for RBAC, LBAC and DAC delegations and authorizations. This fifth step marks the final part of the secure information engineering process, yielding an information secure system via the product of enforcement policies.

6.3 Prototype Mobile PHA Application with Enforcement

Over the past few years, we have been developing two Personal Health Assistant (PHA) (De la Rosa Algarín, A. et al., 2013) mobile application (for Android (Burnette, 2009) and iOS (Goadrich & Rogers, 2011)) for medication management and reconciliation that allows: a PHA-Patient app that allows patients to view and update their personal health record stored in their Microsoft HealthVault (MSHV) (Microsoft HealthVault.2014) account and authorize medical providers to access certain portion of PHI; and, a PHA-Provider app where providers are able obtain the permitted information from their respective patients that they have been authorized to view. PHA-Patient (upper screenshots of Figure 6.4) allows users to perform a set of actions regarding their health information. A user 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, The PHA-Provider (lower screenshots of Figure 6.4) 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.

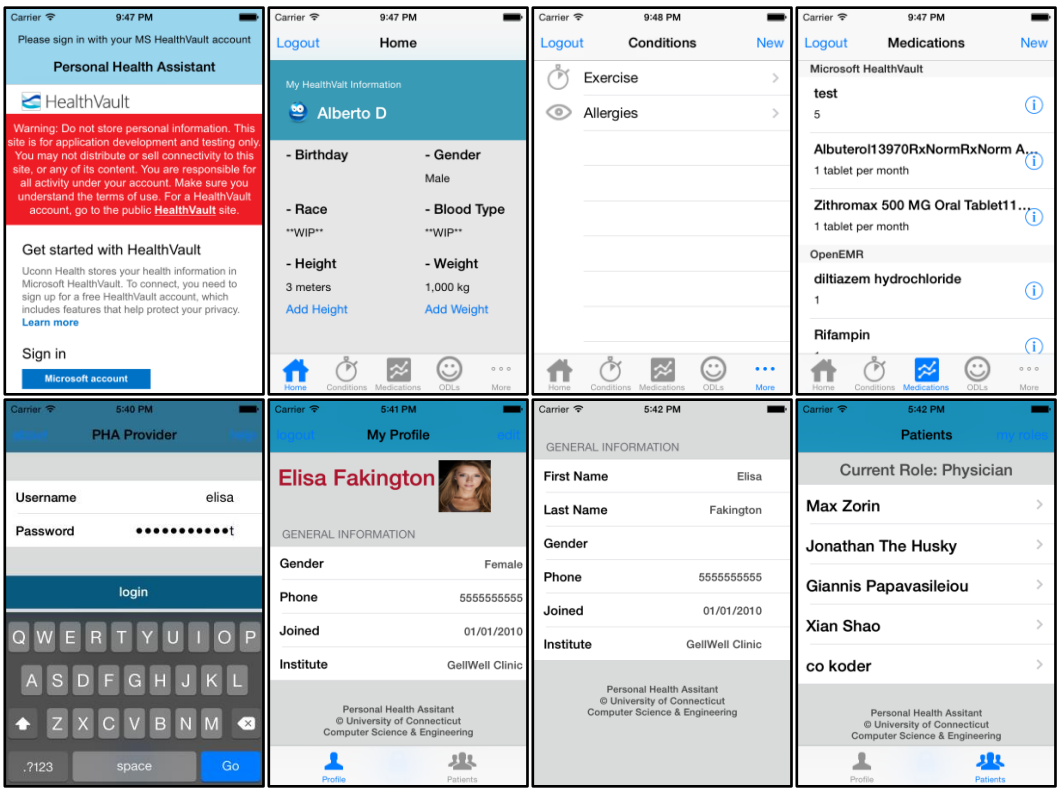

Figure 6.4: Main Screens of PHA-Patient and PHA-Provider Versions.

In order to demonstrate the security framework presented in this dissertation in an actual working healthcare application, we leverage both PHA-Patient and PHA-Provider to supply an example that transitions the new UML diagrams as given in Chapter 4 and utilized in SIEP in a working prototype to demonstrate the process. We focus on PHA-Provider which is capable of enforcing general security via the software's design as well as enforcing XACML policies generated via the process of Chapter 5 and Step (5) of Figure 6.3. The provider version of PHA is the analog of the application commissioned by the Get Better Clinic of the scenario in Section 2.6.

The remainder of this section is organized into six subsections that explain the way that the PHA architecture enforces security that leverages RBAC, LBAC and DAC in the form of an enforcement XACML policy. In Section 6.3.1, we discuss the general PHA architecture, describing the communication between the information system and mobile

application components. In Section 6.3.2, we describe how the general security, consisting of log in authentication and authorization, is handled between PHA-Provider and the RESTful services. In Section 6.3.3, we discuss the set of steps followed to enforce RBAC capabilities defined in the security policy that results from the SIEP of as presented in Section 6.3. In Section 6.3.4, we review the steps that are followed to properly enforce LBAC features, including read and write capabilities with different classifications and clearance levels. Lastly, in Section 6.3.5, we explore role delegation and the steps followed from the perspective of the HVMLS.

6.3.1 The PHA Architecture

The personal health record Microsoft HealthVault (MSHV) acts as the data source for PHA-Patient and PHA-Provider, and stores information in a proprietary format which to be exported via a .NET API which can then be used to generate a CCR compliant. MSHV, acts as the PHA's data source (top of Figure 6.5) 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 Continuity of Care Record (CCR) (Kibbe et al., 2004) schema compliant XML instance. To recreate the typical XACML enforcing architecture, our MSHV Middle-Layer Server (center of Figure 6.4) acts as the contained solution of policy access, information, decision, and enforcement points. The extensible Access Control Markup Language (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 JavaScript Object Notation, or JSON (Crockford, 2006))

of PHA for both the patient and provider versions (middle and bottom of Figure 6.5). To store the relations (mappings) between the authorized list of providers and their respective patients (used in both PHA-Patient and PHA-Provider), our Middle-Layer Server uses MySQL (Kofler, 2001) with a RESTful (Masse, 2011) API done in PHP (Schlossnagle, 2004) with the Slim Framework (Lockhart, 2012). 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.

JSON is utilized for the communication of PHA and the Middle-Layer Server (middle and bottom of Figure 6.5), 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 HealthVault Middle-Layer Server (HVMLS) (middle and lower left of Figure 6.5) is done with unmodified JSON objects, while the communication between the provider version and the Middle-Layer Server (middle and lower right of Figure 6.5) 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 PHA-Patient are translated to and from MSHV objects (upper left of Figure 6.5), since the patient is the owner of the data. Requests done by the PHA-Provider determine the format of the data to be utilized (upper right of Figure 6.5). If a provider is 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.

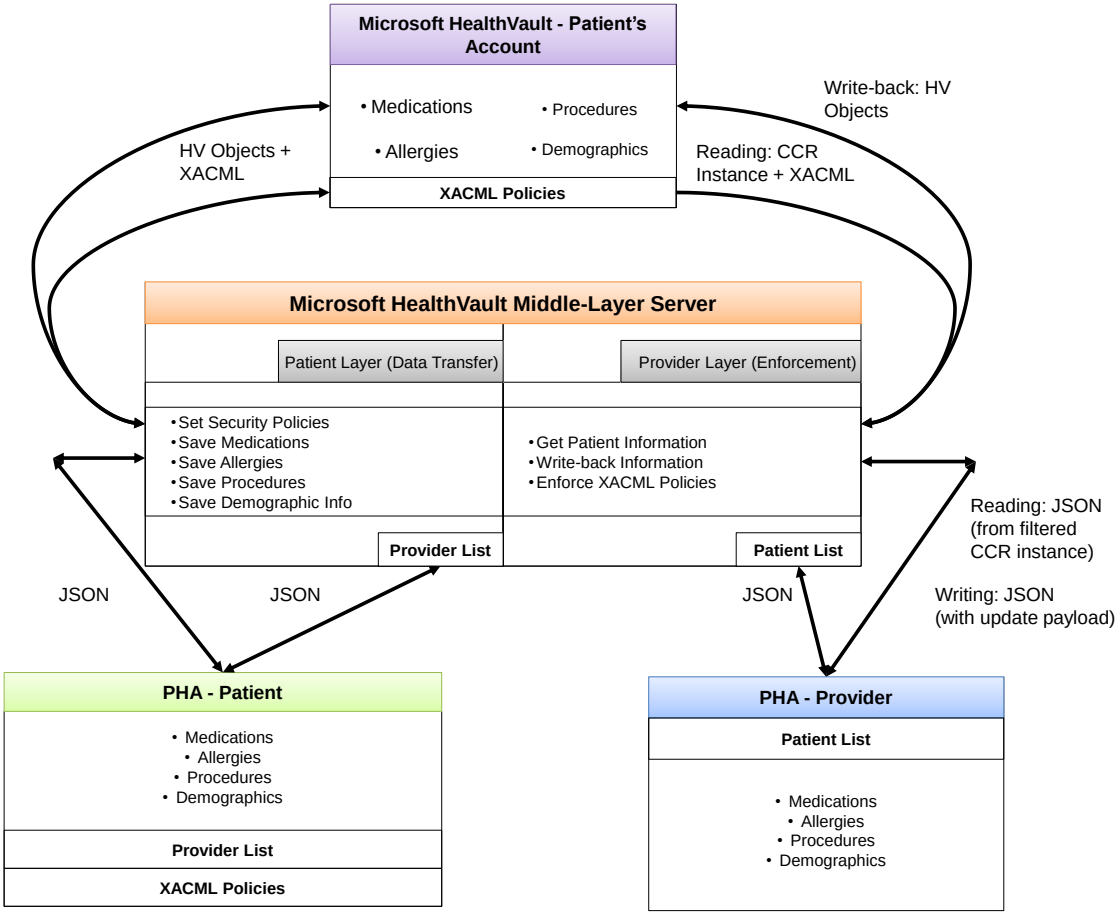

Figure 6.5: Microsoft HealthVault – Middle-Layer – PHA General Architecture.

6.3.2 General Security

The general security of PHA-Provider consists of managing the users and their log in credentials. As a first layer of security, in order to utilize the capabilities of the application, users must log in with their credentials stored behind the HVMLS service (via the use of a MySQL database). In order to verify validity of a user's credentials, PHA-Provider sends a request to the HVMLS service (Figure 6.6) endpoint *auth/provider/login*. In this POST request, the payload sent to the server consists of the username and password. If login is successful, a message in JSON syntax `[{"authentication": "SUCCESS"}]` is sent to the user. If the login is invalid, a message in

JSON syntax `[{"authentication": "FAILED"}]` is sent and no access is granted. As an extra layer of general access to the application, a user can set a PIN code that consists of 4 digits in order to expedite future login attempts once the credentials have been typed. This PIN code is stored in a salted hash (Salted password hashing - doing it right.2014) form in the device's internal storage via the implementation of application default properties, and is not stored in the back-end server.

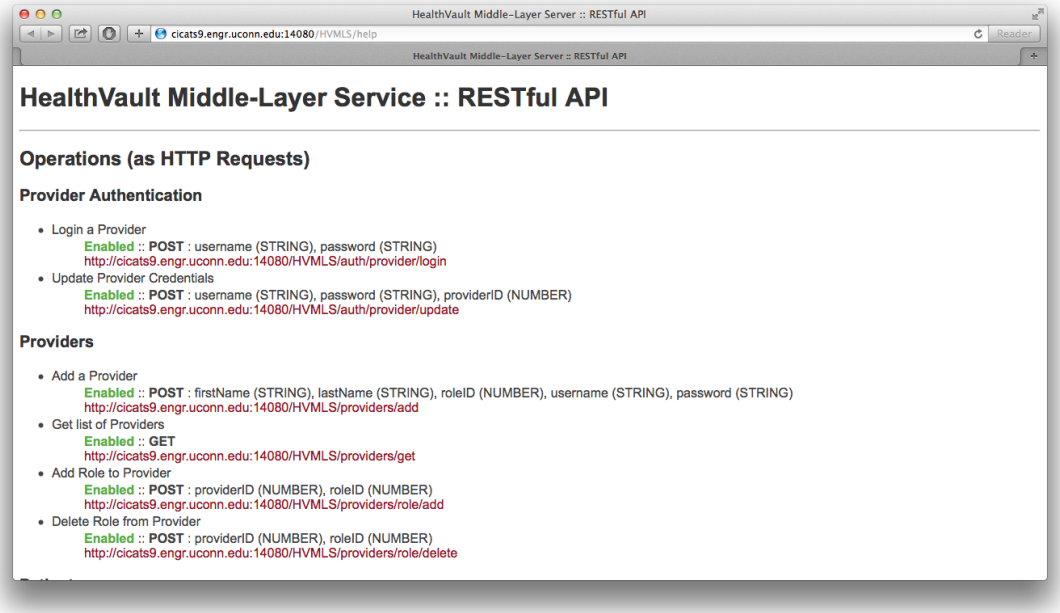

Figure 6.6: Sample of HVMLS RESTful Service Endpoints.

Once a successful login in PHA-Provider has occurred, the user (a provider) is greeted by their information screen (upper right of Figure 6.7) with the option to see their authorized patients (lower right of Figure 6.7). In these screens, the user can change their details with respect to contact information and primary role, or see the list of patients they have with respect to a specific role. For example, the user Elisa could have 3 different patients under her role of Physician, but under a different role she could have none.

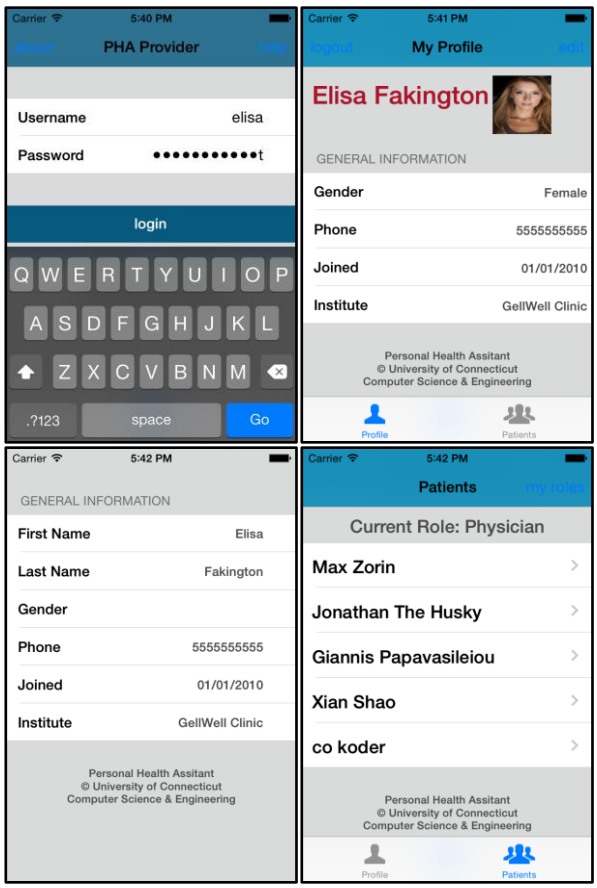

Figure 6.7: Main Screens of PHA-Provider.

When the successful login has been completed, the user's default role is broadcasted by the server to the application. In this process, all of the relevant XACML (Godik et al., 2002) policies generated via the process in Chapter 5, stored in the server, are fetched into memory to be utilized for enforcement per each request of PHA-Provider. In the example shown in Figure 6.8, with user Elisa under the role of Physician following the healthcare scenario of Section 2.6, the server fetches all the XACML policies that have a higher level target with a <Subject> subtree that consist of a <user> element with Elisa's details (user identifier and name). As an example, we revisit Figure 5.16 of Section 5.4 and reintroduce it as Figure 6.8 below. In this generated policy, for user Elisa (line 5, 40 and55 of Figure 6.8) and role Physician (line 13 and 44 of Figure 6.8) contains the RBAC

capabilities (lines 10-36 of Figure 6.8), LBAC features (lines 10-36 of Figure 6.8) and DAC delegations and authorizations (lines 37-66 of Figure 6.8).

6.3.3 RBAC Security Capabilities

In this section, we describe the way that the RBAC component of the XACML policy is enforced when handling reading and writing requests on XML instances whose schema has been secured. From Section 6.2, we assume that a patient has used PHA-Patient to authorize a provider with view and update capabilities on their data stored in MSHV. This authorization in PHA-Patient essentially supplies some of the user/role/permissions that are to be enforced against a particular provider as authorized by a patient. Assuming that this has occurred, the focus is on the usage of PHA-Provider in order to detail the process and steps that are taken when a user of the PHA-Provider attempts to access data on an authorized patient, and to serve as 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 HVMLS (center portion of Figure 6.5).

The process of securing the CCR instance for operations that have an access mode of 'read' (read and aggregate, see Defn. 34 of Section 3.4) is shown as a set of interconnecting steps in the flowchart of Figure 6.9, and begins with a request from the PHA-Provider. 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 with respect to that user and their current role. 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

```
<Policy PolicyId="ada-policy" RuleCombiningAlgId="deny-overrides">
<Description>Omitted due to length.</Description>
<Target>
<Subjects>
<user><id>6</id><name>Elisa</name></user>
</Subjects>
<Resources><AnyResource/></Resources>
<Actions><AnyAction/></Actions>
</Target>
<Rule RuleId="simple-RBAC+LBAC-rule" Effect="Permit">
<Target>
<Subjects>
<role><roleID>5</roleID><roleName>Physician</roleName></role>
</Subjects>
<Resources><element>
<elementID>el-3</elementID>
<elementName>Past Medical History</elementName>
</element></Resources>
<Actions><operation>
<operationName>insert</operationName>
<opAccessMode>write</opAccessMode>
</operation></Actions>
</Target>
<Condition>
<Apply FunctionId="…:integer-greater-than-or-equal">
<Apply FunctionId="…:integer-one-and-only">
<AttributeValue DataType="…#integer">
Secret
</AttributeValue>
</Apply>
<AttributeValue DataType="…#integer">
Secret
</AttributeValue>
</Apply>
</Condition>
   </Rule>
   <Rule RuleId="simple-delegation-rule" Effect="Permit">
   <Target>
   <Subjects>
   <user><id>6</id><name>Elisa</name></user>
   </Subjects>
   <Resources>
   <Roles><role>
   <roleID>2</roleID><roleName>Physician</roleName>
   </role></Roles>
   </Resources>
   <DelegationTargets>
   <user><id>30</id><name>Samantha</name></user>
   </DelegationTargets>
   </Target>
   </Rule>
   <Rule RuleId="simple-authorization-rule" Effect="Permit">
   <Target>
   <Subjects>
   <user><id>6</id><name>Elisa</name></user>
   </Subjects>
   <Resources><Schemas><schema>
   <schemaID>4</schemaID>
   <schemaName>Schema 4</schemaName>
   </schema></Schemas>
   <Instances><instance>
   <instanceID>4,2</instaneID>
   <instanceName>Carol Smith Health Record</instanceName>
   </instance></Instances></Resources>
   </Target>
   </Rule>
   </Policy>
```

Figure 6.8: XACML Enforcement Policy for User Elisa and Role Physician.

preferences. In terms of the XACML given in Figure 6.8, this would correspond to the read operation inside the <Rule> element. Since the PHA-Provider handles information in JSON format at the front-end, 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 6.9). This equivalent JSON object is then utilized by PHA-Provider to present the patient data to the provider who is able to view and update as necessary.

The process of securing the CCR XML for operations that have a write access mode (insert, update and delete, see Defn. 34 of Section 3.4) is shown as a set of interconnecting steps in the flowchart of Figure 6.10, and begins with a request from PHA-Provider. When a provider wants to update a patient's record (e.g., medication), the request is sent to the HVMLS tied to the update data as a JSON object (see right side of Figure 6.10) that 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 6.8 with the insert <operation> in lines 19-22.

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, HVMLS drops the request. While XPath and XQuery do not allow the process to update an XML instance, our approach as given in Figure 6.10 provides a means for updating XML documents (e.g. CCR instances) that is controlled via an XACML security policy with the assistance of HVMLS.

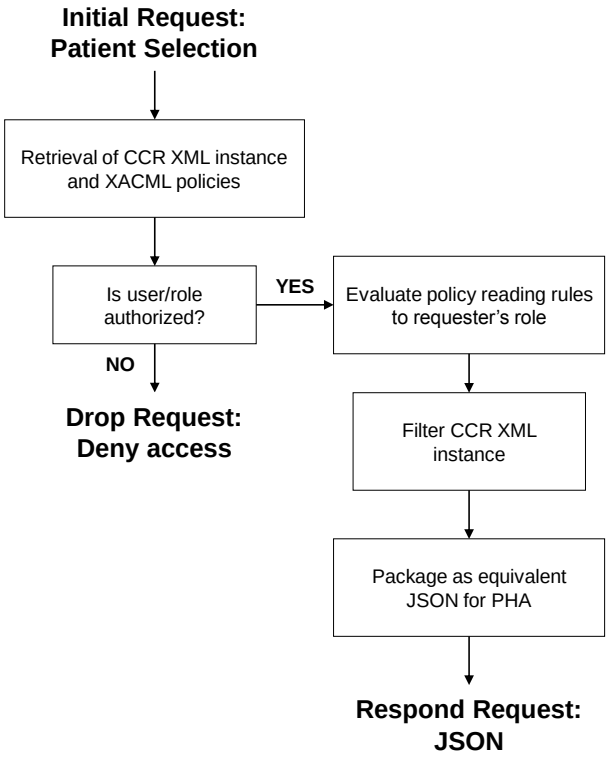

Figure 6.9: Enforcing Reading Permissions.

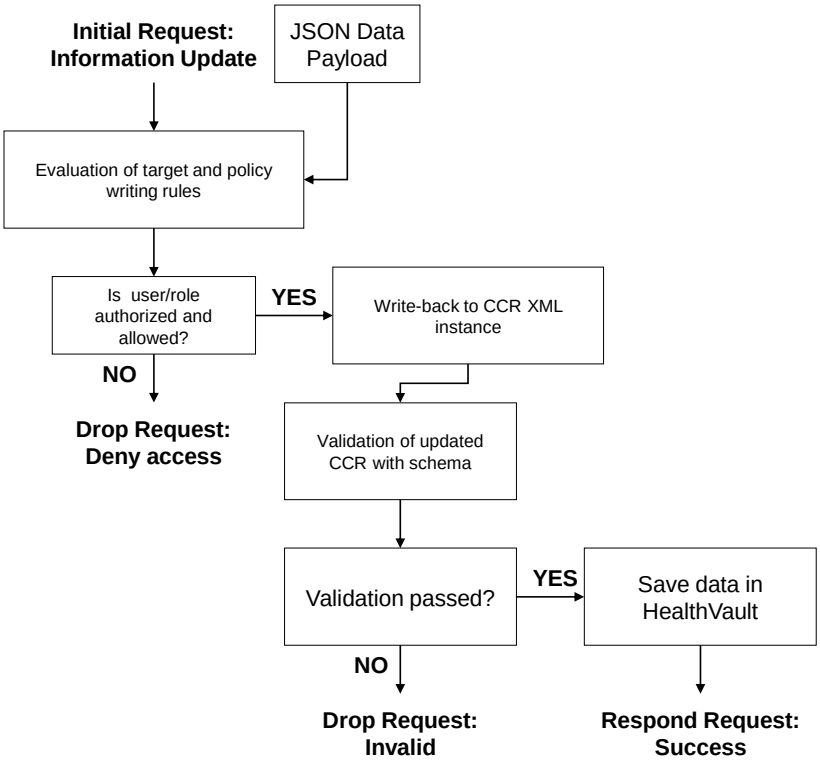

Figure 6.10: Enforcing Writing Permissions.

6.3.4 LBAC Security Features

In this section, we explain the way that the LBAC component of the XACML policy is enforced when handling reading and writing requests such as those in Section 6.2.2. Following the same assumptions of Section 6.2, we assume that the engineer has set classification levels to the different segments of the health record and a clearance level to the user. In the case of Elisa, her clearance level is *Secret* as defined by the segment in lines 34-25 of Figure 6.8. The process of securing the CCR instance with LBAC features for operations across the two access modes (read and write, see Defn. 34 of Section 3.4) is shown as a set of connecting steps in Figure 6.11. When the initial request is made on behalf of PHA-Provider, the server responds by evaluating the target, policy rules, and LBAC features used by the security architecture by using the requested element's classification and the user's clearance. If the user's clearance equals or exceeds what is required (e.g., simple-security, simple-integrity, etc. as discussed in Section 3.4), the server continues to read the XML instance's segment and generate output (responding the request with a success as shown in the right-most path of Figure 6.11) or validates the updated CCR with the schema in a similar fashion as the RBAC write from Section 6.3.3. If the validation is successful, in the case of operations with a *write* access mode, then the data is saved in HealthVault and a response of success is provided to the application. If the validation is not successful, then the attempt was not legal and the request is dropped without any changes done to the saved data in HealthVault.

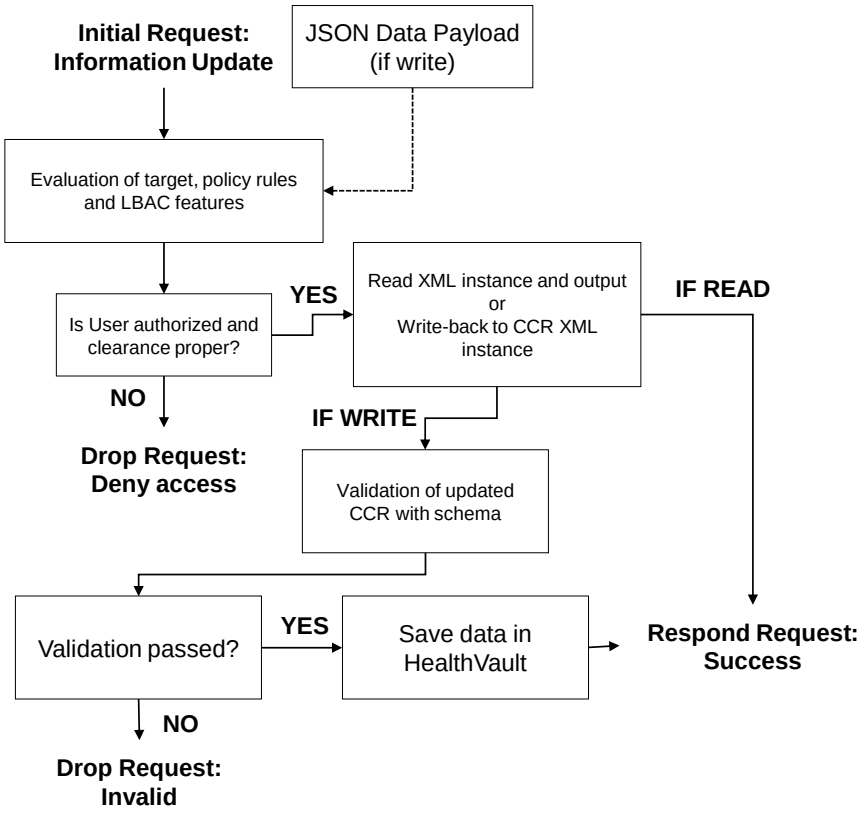

Figure 6.11: Enforcing LBAC Features for Operations with Read/Write Access Modes.

6.3.5 DAC Delegations

Enforcing security for delegation follows the steps in Figure 6.12. When the request to delegate a role is performed, the server verifies the user initiating the request (which can be an original user or a delegable user) and the receiving user against the user objects and their properties in the MySQL database. If the initiating user is an original user (see Defn. 36 of Section 3.5) and the receiving user is a delegable user (see Defn. 37 of Section 3.5), the update is recorded in the database and the role is deactivated for the original user. If the initiating user is a delegable user, then the server verifies whether there is a pass-on delegation authority for that role (see Defn. 38 of Section 3.5). If there is no pass-on delegation, the request is dropped and a response of invalid is sent to the

application. If there is pass-on delegation, then the process follows the same order as if the initiating user is an original user.

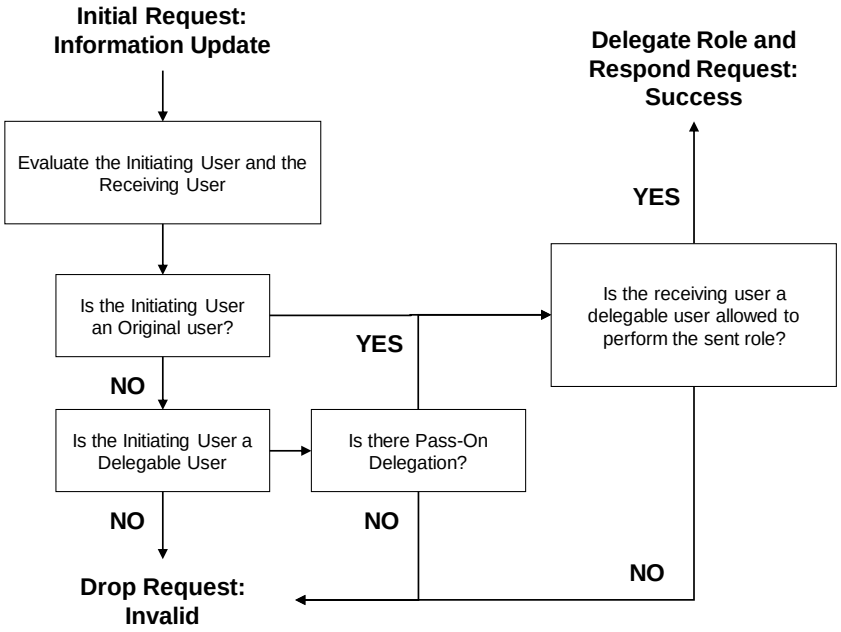

Figure 6.12: Enforcing Security in Role Delegation.

Chapter 7
Conclusion

The work presented in this dissertation is summarized as a security framework for tree-structured documents that leverages RBAC (Ferraiolo et al., 2001), LBAC (Sandhu, 1993) and DAC (Sandhu & Samarati, 1994). The main objectives have been four-fold: create an underlying security model that abstracts away from specific document formats (e.g. XML (Bray et al., 1998), JSON (Crockford, 2006), etc.) and considers their most basic form as tree-structured containers while supporting RBAC, LBAC and DAC capabilities as an integrated solution; representing the underlying security model with extensions to the Unified Modeling Language (UML) (Fowler, 2004) model and metamodel layers by leveraging previous work of (Pavlich-Mariscal et al., 2010) and (Berhe et al., 2010) that focused on functional and collaborative security, resulting in seven new diagrams that graphically represent the document's schema, the role slices, users, delegation and authorization properties; the introduction of an enforcement policy generation process that leverages those new seven UML extensions to create instances of policies ready to be deployed, exploiting XACML (Godik et al., 2002) as the language of choice; and, a secure information engineering process that utilizes all the previously discussed objectives to introduce a development cycle that focuses on information security, a process that could be integrated into those presented by (Pavlich-Mariscal et al., 2010) and (Berhe et al., 2010). The driving motivation for these four objectives has been illustrated with the healthcare domain by presenting a realistic scenario where a need for granular information security is necessary when multiple users, roles, clearances and information are present.

The remainder of this conclusion is organized as follows. Section 7.1 summarizes the dissertation, discussing the four main objectives mentioned above in further detail. Using this as a basis, Section 7.2 discusses the research contributions of this dissertation, primarily in the areas of information security and access control models, UML extensions for information security, policy generation processes, and secure information engineering. Then, on Section 7.3, we detail the ongoing and future research directions that include, but are not limited to, support for non-orthogonal RBAC and LBAC security requirements, other access control models (e.g. Attribute-Based Access Control (ABAC) (Yuan & Tong, 2005)), and compartments for a greater level of security by isolating information with respect to roles and users.

7.1 Summary of the Dissertation

The work presented in this dissertation tackles the areas of motivation and need for information security for those applications that require granular security assurance in personal and/or protected information; security and access control models such as RBAC, LBAC and DAC that are required to not only provide a basic level of security, but also interact with each other as an integrated solution to provide a more robust security mechanism; enforcement policy generation that is automatic and formal; and, a secure information engineering process that combines all the previous aspects into a formal development cycle process for security administrators and software developers to utilize. In support of these research areas, the discussion was organized throughout six chapters.

Information modeling is focused on representing, using, and exchanging information in large-scale applications that include healthcare collaborative and non-collaborative scenarios that use the Health Level 7's clinical document architecture (Alschuler et al.,

2002) or the continuity of care record (Kibbe et al., 2004). Chapter 1 presented the motivation and need for information security from a broad domain perspective. In Chapter 1 we discussed how these information systems may support a wide range of data formats such as XML, JSON, RDF and OWL. These information applications data often have a tree structure of index and entity nodes that allows for information to be modeled via schemas (that define structure) which can be used as blueprints for the creation of new documents (instances) and their validation (enforcement). In such settings, where sensitive data is utilized by users for time-critical applications, security that is achieved via access control is a paramount concern. These applications all present unique challenges to the objective of providing a high-degree of protection to information. The overarching need is the ability to provide granular access control in support of information modeling for structured documents is based on security policies defined in an local manner (e.g., institutions) and guidance defined and enforced at a more global scope (e.g., legal entities and active pieces of legislation), has proven to be difficult to achieve.

Chapter 2 reviewed background concepts utilized throughout this dissertation in a wide range of areas to support the detailed material in the remaining chapters. First, Section 2.1 briefly reviews XML and its usage in the two dominant healthcare standards: HL7 CDA and CCR. Next, Sections 2.2, 2.3 and 2.4 present the various access control models that are utilized in support of the work in this dissertation, respectively: RBAC to support roles, LBAC to support classification and clearance levels, and DAC to allow both authority and privileges to be passed from one user to another. Then, Section 2.5 presented a set of assumptions for the healthcare domain, more specifically, detailing the way that health information technology (HIT) systems leverage private and protected

information in interacting with patients for care and treatment. Section 2.6 presented a scenario of information usage in the healthcare domain using the two dominant standards (HL7 CDA and CCR) with an emphasis on the way to secure the information across differing degrees of granularity and requirements. Using this as a basis, Section 2.7 introduced the Unified Modeling Language (UML) (Fowler, 2004) and its metamodeling capabilities that are going to be utilized throughout this dissertation to define the new UML diagrams to support the modeling of XML and LBAC/RBAC/DAC. Lastly, Section 2.8 introduced XACML, a language utilized to create formal and enforceable security policies for XML documents.

The objective of Chapter 3 was to propose a security model for tree-structured documents that includes the ability to define RBAC, LBAC and DAC for information systems. The model presented in Chapter 3 defines the underlying concepts and capabilities that served as a foundation for the definition of new UML diagrams to model RBAC, LBAC, and DAC for tree-structured documents, with the intent to achieve fine-grained information security via access control as part of the overall software engineering process for information systems. The inclusion of security as part of an information system's design facilitates the modeling of RBAC, LBAC, and DAC at a schema-level that is then realizable against its instances. The key intent of our approach is for a schema-level security solution that defines and enforces fine-grained control of an information system's instances for: non-destructive (document-level) operations that utilize instances as a source of information (e.g., read and aggregate); destructive (document-level) operations that modify instance(s) to reflect a change (e.g., insert,

update, delete); and, other types of operations (policy-level) that act in the instance as a whole and not in the intrinsic data found within.

Building from the model of Chapter 3, Chapter 4 presented the second major component of our security framework for tree-structured documents involves the realization of the security model as a series of new UML diagrams that capture the characteristics of the security model and allow us to augment the software engineering process of UML with an information engineering process for tree-structure documents. Recall from Section 2.7 that UML provides a large variety of diagrams for the visualization of different software requirements: class, component, deployment, activity, use-case, state-machine, communication, sequence, etc. UML provide the benefit of reducing misinterpretation and promoting simple communication of domain requirements with its visual notation (Lange & Chaudron, 2005). However, while UML can be utilized to define security requirements, what is lacking in UML is actual diagrams that are dedicated to, in our interest, access control models (RBAC, LBAC, and DAC) that allow the definition of security requirements using new security UML diagrams that seamlessly integrate with the UML model and unified design process. This is particularly true for domains such as healthcare where the information to be utilized is private and often governed by legal constructs that assure its proper use and dissemination, as we have described in Section 2.6 and illustrated with a detailed example for our security model of Chapter 3.

With the introduction of the UML extensions of Chapter 4, Chapter 5 presented the third component of our framework that supports the automatic generation of a security enforcement policy when given a security design for a set of schemas as captured in our

new UML diagrams. UML has a long history for the automatic generation of code (REFS) in varied languages; our usage of our new UML diagrams to generate a security policy is consistent with this usage. In this chapter, we present a process for the generation of enforcement policies that transitions a UML design containing a Document Schema Class Diagram (DSCD), a Secure Information Diagram (SID), a Document Role Slice Diagram (DRSD), lattice-based access control SID (LSID), a User Diagram (UD), a Delegation Diagram (DD), and an Authorization Diagram (AD); see respectively Sections 4.2.1 to 4.2.7. To support the automatic generation of a security enforcement policy, we define a set of *mapping statements (MSs)* that are utilized to define the conditions under which the combination of the various diagrams (DSCD, SID, DRSD, LSID, UD, DD, and AD) can be utilized to support the creation of respective policies for RBAC, LBAC, DAC, and authorization. A mapping rule (MR) is defined to take the security model concepts and capabilities to Chapter 3 that both underlie correspond to different portions of the new the UML diagrams of Chapter 4 and use this combination to yield a portion of the security policy.

Then, in Chapter 6, we discussed the software engineering process that leverages the security model presented in Chapter 3, the UML diagram and metamodel extensions presented in Chapter 4, and the policy mapping process presented in Chapter 5. Over the past five years, major focus has been on extending UML with new diagrams that supports secure software engineering for RBAC, MAC, and DAC (Pavlich-Mariscal et al., 2014). First, from a functional perspective that focuses on object-oriented design, a framework of composable security features was defined (Pavlich-Mariscal et al., 2010) that preserves separation of security concerns from models to code through the extension of UML with

new diagrams for RBAC, DAC, and MAC with the automatic generation of enforcement code that allowed the security definitions to be separated (untangled) from the code. Second, from a collaboration perspective, a framework for secure, obligated, coordinated, and dynamic collaboration was developed (Berhe et al., 2010) that extended the NIST RBAC to allow for the definition and enforcement of security with new UML diagrams for collaborative RBAC applicable to situations such as medical care where physicians from different specialties need to collaborate with one another to treat a patient in an effective and timely manner. The last perspective is the work presented in this dissertation, which has focused on the definition of a modeling and design framework with enforcement process for information-based applications. To place the work of this dissertation into an appropriate context with our prior wok, this chapter has two objectives. First, this chapter reviews the overall integrated secure software engineering process that spans functional, collaboration, and information concerns. Second, using this as a basis, we then concentrate on the secure information engineering process that targets the information security concerns for an application that are the basis of our work in this dissertation as presented in Chapters 3, 4 and 5.

7.2 Research Contributions

As outlined previously, our research contributions are primarily in the areas of security and access control models that support RBAC, LBAC and DAC; UML security extensions in the model and metamodel layers for information access control with RBAC, LBAC and DAC; the generation of enforcement policies from UML diagrams; and, the secure information engineering process. While the research contributions presented in this dissertation are unique and extend the research that is being conducted

in the major areas we tackle as discussed in the beginning of Section 7.1, we further describe each into detail to differentiate from the relevant research of other efforts in the same areas.

E. **Security Model and Access Control Integration**: The contribution of this aspect of our work provides an underlying security model that leverages RBAC, LBAC and DAC with the purpose of securing tree-structure documents that could be XML, JSON, RDF, OWL, etc. The initial aspects of our work towards this objective tackled the integration from the UML model perspective (De la Rosa Algarín, A. & Demurjian, 2013). We note that a more formal alternative was to build a properly defined model in which integration was a key component. Towards that purpose, in Chapter 3 we presented the formal model that supports a wide array of capabilities (e.g. roles, clearances, classifications, authorizations, delegations, etc.) via the introduction of specialized operations such as the projection and decoration operations that act on a tree-structure and result in an altered version. As a special consideration of this model, we assert that RBAC and LBAC are orthogonal. That is, their capabilities do not affect each other. This allows for a security solution to support either or both at the same time.

F. **Seven UML Extensions for Tree-structured Document Security**: The contribution of this aspect of our work is to represent a tree-structured 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, LBAC, and DAC. Chapter 4 is dedicated to this research contribution, describing the seven new UML diagrams for information security that are built from extensions to the UML

metamodel: the Document Schema Class Diagram (DSCD), which is used to represent the tree-structured schemas a UML diagram; the Document Role Slice Diagram (DRSD), which is used to support RBAC capabilities; the Secure Information Diagram (SID), which is used to support a projection over elements; the LBAC Secure Information Diagram (LSID), which is used to support LBAC features; the User Diagram (UD), which is used to describe the users and their attributes in an information system; the Delegation Diagram (DD), which is used to describe the delegation capabilities from users; and, the Authorization Diagram (AD), which is used to describe the authorizations to a user/role combination. The introduction of these UML diagrams provides the benefit to generate enforcement code that is formalized by the underlying security model.

G. **Security Policy Generation**: The contribution in this aspect of our work leverages the UML diagrams presented in Chapter 4, which were built from the underlying security model of Chapter 3, and provides a process for automatic generation of enforcement policies leveraging the XACML specification. This is achieved via mapping statements from the new UML security diagrams for RBAC, LBAC and DAC, and the XACML Policy structure, which can be used as part of an enforcement mechanism to assure security. Towards this contribution, we have also introduced an automatic mapping algorithm that considers RBAC, LBAC and DAC security components and integrates them into a single enforcement policy for a required application.

H. **Secure Information Engineering**: This contribution provides a secure information engineering process to systems information by focusing on the

perspective of the information to be secured. This secure information engineering process involves the global security model and policy integration by modeling the tree-structured schemas to be secured with the respective access control security models (Contribution A), the use of UML diagrams for information security (Contribution B), and the generation of enforcement policies (Contribution C), all utilized as part of a development cycle for information security that can be integrated with the work of (Pavlich-Mariscal et al., 2010) and (Berhe et al., 2010) to provide an overall process for functional, collaborative an information concerns (Pavlich-Mariscal et al., 2014). 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 in their respective application.

We note that, as part of our research, a strong attempt to consider real world scenarios throughout the dissertation was made. More specifically, we introduced a healthcare driven scenario in Section 2.6, components of which were utilized to explain concepts of the major contributions (Chapters 3, 4, 5 and 6). The enforcement prototype of Section 6.2 has been the result of years of development that tie directly to the healthcare domain example, and the enforcement cases presented tie with the scenario of Section 2.6. This important final step demonstrates the practical application of our research.

7.3 Ongoing and Future Research

While we note that the research contributions of our work presented herein are unique, we realize that there are interesting research areas that serve as our ongoing work and future research directions. These future research directions consider alternations to some of our assumptions of the security model (non-orthogonal RBAC and LBAC);

support for other access control models (e.g. ABAC); support for concepts such as compartments; and others. We have identified the following research direction as of potential interest.

Non-Orthogonal RBAC and LBAC: The security model of Chapter 3 (contribution A) supports RBAC, LBAC and DAC capabilities in certain unison. The model asserts that RBAC and LBAC capabilities are orthogonal. What this means is that clearance levels from LBAC are assigned to the user and not the role (which is also assigned to the user). This avoid certain complexities when enforcing security as LBAC is considered to be the most constrained of the access control models. An interesting area of future work is to understand how the security model would change if RBAC and LBAC were non-orthogonal. That is, the clearance level for a user could also be assigned to the role. In those cases we ask ourselves, which clearance would take precedence? Would the decision depend on the context of the requests which are secured? Taking a page from the healthcare domain book: how would the delegation of roles in an emergent care situation result when the delegable user might not have enough permission? These questions paint an interesting picture in terms of complexity, something worthy of pursuing. Because of this, we consider this to be an important future area of research.

Support for other access control models: The security model of Chapter 3 (contribution A) and the UML extensions of Chapter 4 (contribution B) support RBAC, LBAC and DAC. These access control models are considered to be the major security models, being extensively utilized in cross-domain applications. That said, there are other models that become more and more useful as the shape of information changes. One example is the attribute-based access control (ABAC) model. Support of models such as

these will undoubtedly affect the way RBAC and LBAC are enforced (recall that DAC lives at the policy level, not the document level), and the overall integration might result in something different to what we have presented here. We ask ourselves: which access control model would take the primary spot of enforcement? How different would it be to support ABAC if RBAC and LBAC are orthogonal? How about when they are non-orthogonal? We hypothesize that ABAC will affect LBAC more than RBAC, but that is a hypothesis that requires research to be proved. Therefore, we conclude that this is another important area of research.

Support of information compartments: As a small future area of research, we wish to support the definition of compartments for information. Compartmentalization is extensively used in the defense community as a means to protect data by the method of isolation (no one knows the complete secret). Towards this purpose, should compartments be defined and assigned to roles or users? Should they be assigned to the user/role combination? How would clearance levels and classification of elements in the compartment interplay?

Collaboration workflows in information security: The work presented in this dissertation finds as inspiration the work presented by (Pavlich-Mariscal et al., 2010) and (Berhe et al., 2010). One area of research we seek to pursue is the collaboration workflows and obligations presented by (Berhe et al., 2010) when applied to information security. Unlike software security, which focuses on methods, information security can be very granular or very coarse. The potential of collaboration across a document opens the idea of granular and coarse collaboration, as well as those other topics tackled by (Berhe et al., 2010) in his research.

Automatic creation of DSCD: Chapter 4 presented the UML diagrams for information security. One of those diagrams, the Document Schema Class Diagram (DSCD), was built as part of a UML profile that created a relation between tree-structures and UML components. We want to explore if that process can be automated to the point where there are no errors. At the same time, we want to explore if a metamodel extensions to convert tree-structures into UML is more scalable. Potential schemas can have thousands of nodes, and automatic conversion to UML might not scale when using a simple profile approach.

Generating enforcement policies in different languages: Another area of interest relates to the generation of enforcement policies. Chapter 5 presented an approach to create an XACML instance policy from the UML diagrams, but we seek to explore the possibilities to generate SQL code, aspect oriented code (Pavlich-Mariscal et al., 2010) for information security from the diagrams presented in Chapter 5. As technology advances and standards are dropped in favor of others, we cannot assume that XACML will be the one option for the future. Another aspect with regards to generating enforcement policies is the performance and efficiency of the algorithm. We assert that there must exist other methodologies to generate a policy, other algorithms that are more efficient. Our ongoing work includes the development of these more efficient algorithms.

References

Accountable care organizations. (2011). Retrieved, 2014, Retrieved from http://www.cms.gov/Medicare/Medicare-Fee-for-Service-Payment/ACO/

Alghathbar, K. (2007). Validating the enforcement of access control policies and separation of duty principle in requirement engineering. *Information and Software Technology, 49*(2), 142-157.

Alschuler, L., Mair, M. W., Boyer, S., & Dolin, R. H. (2002). Hl7 clinical document architecture. *Context"," Health Information New Zealand.Auckland, New Zealand*,

Annas, G. J. (2003). HIPAA regulations—a new era of medical-record privacy? *New England Journal of Medicine, 348*(15), 1486-1490.

Basin, D., Doser, J., & Lodderstedt, T. (2006). Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology (TOSEM), 15*(1), 39-91.

Bauer, C., & King, G. (2005). Hibernate in action.

Baumer, D., Earp, J. B., & Payton, F. C. (2006). Privacy of medical records: IT implications of HIPAA. *Ethics, Computing, and Genomics.Jones and Bartlett, Sudbury, MA*, , 137-152.

Bell, D. E., & La Padula, L. J. (1976). *Secure Computer System: Unified Exposition and Multics Interpretation*,

Berhe, S., Demurjian, S., & Agresta, T. (2009). Emerging trends in health care delivery: Towards collaborative security for NIST RBAC. *Data and Applications Security XXIII*, , 283-290.

Berhe, S., Demurjian, S., Gokhale, S., Pavlich-Mariscal, J., & Saripalle, R. (2011). 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*, , 293-300.

Berhe, S., Demurjian, S., Saripalle, R., Agresta, T., Liu, J., Cusano, A., . . . Gedarovich, J. (2010). Secure, obligated and coordinated collaboration in health care for the patient-centered medical home. *AMIA Annual Symposium Proceedings, , 2010* 36.

Bernauer, M., Kappel, G., & Kramler, G. (2003). Representing XML schema in UML-an UML profile for XML schema.

Bernauer, M., Kappel, G., & Kramler, G. (2004). Representing XML schema in UML–A comparison of approaches. *Web Engineering*, , 767-769.

Bertino, E., Carminati, B., & Ferrari, E. (2004). Access control for XML documents and data. *Information Security Technical Report, 9*(3), 19-34.

Bertino, E., Castano, S., Ferrari, E., & Mesiti, M. (2002). Protection and administration of XML data sources. *Data & Knowledge Engineering, 43*(3), 237-260.

Bertino, E., & Ferrari, E. (2002). Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security (TISSEC), 5*(3), 290-331.

Bilykh, I., Jahnke, J. H., McCallum, G., & Price, M. (2006). Using the clinical document architecture as open data exchange format for interfacing emrs with clinical decision support systems. *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium On,* 855-860.

Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., Siméon, J., & Stefanescu, M. (2002). XQuery 1.0: An XML query language.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72.

Boudreau, T., Glick, J., Greene, S., Spurlin, V., & Woehr, J. J. (2002). *NetBeans: The definitive guide* O'Reilly Media, Inc.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (1998). Extensible markup language (XML). *World Wide Web Consortium Recommendation REC-Xml-19980210.Http://Www.W3.Org/TR/1998/REC-Xml-19980210,*

Burmester, S., Giese, H., & Schäfer, W. (2005). Model-driven architecture for hard real-time systems: From platform independent models to code. *Model Driven Architecture–Foundations and Applications,* 25-40.

Burnette, E. (2009). *Hello, android: Introducing google's mobile development platform* Pragmatic Bookshelf.

Cao, J., Sun, L., & Wang, H. (2005). Towards secure XML document with usage control. *Web Technologies Research and Development-APWeb 2005, ,* 296-307.

Carlson, D. (2008). UML profile for XML schema.

Clark, J. (1999). Xsl transformations (xslt). *World Wide Web Consortium (W3C).URL Http://Www.W3.Org/TR/Xslt,*

Clark, J., & DeRose, S. (1999). XML path language (XPath) version 1.0.

CodeSmith tools. (2014). Retrieved, 2014, Retrieved from http://www.codesmithtools.com/

Cohen, D., Lindvall, M., & Costa, P. (2003). Agile software development. *Data & Analysis Center for Software (DACS), New York,*

Combi, C., & Oliboni, B. (2006). Conceptual modeling of XML data. *Proceedings of the 2006 ACM Symposium on Applied Computing,* 467-473.

Conover, M. (2006). Analysis of the windows vista security model.

Conrad, R., Scheffner, D., & Christoph Freytag, J. (2000). XML conceptual modeling using UML. *Conceptual Modeling—ER 2000, ,* 291-307.

Crockford, D. (2006). JSON: The fat-free alternative to XML. *Proc. of XML, , 2006*

Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., & Samarati, P. (2000). Design and implementation of an access control processor for XML documents. *Computer Networks, 33*(1), 59-75.

Damiani, E., Fansi, M., Gabillon, A., & Marrara, S. (2008). A general approach to securely querying XML. *Computer Standards & Interfaces, 30*(6), 379-389.

De la Rosa Algarín, A., & Demurjian, S. A. (2013). An approach to facilitate security assurance for information sharing and exchange in big data applications. In B. Akhgar, & H. R. Arabnia (Eds.), *Accepted in emerging trends in information and communication technologies security* () Elsevier.

De la Rosa Algarín, A., Demurjian, S. A., Ziminski, T. B., Rivera Sanchez, Y. K., & Kuykendall, R. (2013). Securing XML with role-based access control: Case study in health care . In A. Ruiz Martínez, F. Pereñíguez García & R. Marín López (Eds.), *Architectures and protocols for secure information technology* (pp. 334-365) IGI Global.

De la Rosa Algarín, A., Demurjian, S. A., Berhe, S., & Pavlich-Mariscal, J. (2012). A security framework for XML schemas and documents for healthcare. Paper presented at the *Proceedings of 2012 International Workshop on Biomedical and Health Informatics,* 782-789.

De la Rosa Algarín, A., Ziminski, T. B., Demurjian, S. A., Kuykendall, R., & Rivera Sánchez, Y. (2013). Defining and enforcing XACML role-based security policies within an XML security framework. *Proceedings of 9th International Conference on Web Information Systems and Technologies (WEBIST 2013),* 16-25.

De la Rosa Algarín, A., Ziminski, T. B., Demurjian, S. A., Rivera Sanchez, Y. K., & Kuykendall, R. (2013). Generating XACML enforcement policies for role-based access control of XML documents. In W. M. P. van der Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw & C. Szyperski (Eds.), *Accepted in lecture notes in business information processing (LNBIP) (WEBIST selected papers)* () Springer.

Dinh-Trong, T. T., & Bieman, J. M. (2005). The FreeBSD project: A replication case study of open source development. *Software Engineering, IEEE Transactions On, 31*(6), 481-494.

Dittrich, K. R., Hartig, M., & Pfefferle, H. (1989). Discretionary access control in structurally object-oriented database systems. *Proc. of the 1988 Workshop on Database Security, Kingston, Ontario, Canada, Sponsored by IFIP WG, , 11*

DoD trusted computer system evaluation criteria. (2014). Retrieved, 2014, Retrieved from http://csrc.nist.gov/publications/history/dod85.pdf

Dolin, R. H., Alschuler, L., Boyer, S., Beebe, C., Behlen, F. M., Biron, P. V., & Shvo, A. S. (2006). HL7 clinical document architecture, release 2. *Journal of the American Medical Informatics Association, 13*(1), 30-39.

Downs, D. D., Rub, J. R., Kung, K. C., & Jordan, C. S. (1985). Issues in discretionary access control.

Faden, G. (2006). Solaris trusted extensions. *Sun Microsystems Whitepaper, April,*

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC), 4*(3), 224-274.

Ferranti, J. M., Musser, R. C., Kawamoto, K., & Hammond, W. (2006). The clinical document architecture and the continuity of care record: A critical analysis. *Journal of the American Medical Informatics Association, 13*(3), 245-252.

Fowler, M. (2004). *UML distilled: A brief guide to the standard object modeling language* Addison-Wesley Professional.

Goadrich, M. H., & Rogers, M. P. (2011). Smart smartphone development: iOS versus android. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education,* 607-612.

Godik, S., Anderson, A., Parducci, B., Humenn, P., & Vajjhala, S. (2002). OASIS eXtensible access control 2 markup language (XACML) 3.

Hamburg, M. A., & Collins, F. S. (2010). The path to personalized medicine. *New England Journal of Medicine, 363*(4), 301-304.

Harrison, G. (2000). *Oracle SQL high-performance tuning* Prentice Hall Professional Technical Reference.

HITECH act enforcement interim final rule. (2014). Retrieved, 2014, Retrieved from http://www.hhs.gov/ocr/privacy/hipaa/administrative/enforcementrule/hitechenforcementifr.html

IBM-informix database software. (2014). Retrieved, 2014, Retrieved from http://www.informix.com

Jacobson, I. (1999). *The unified software development process* Pearson Education India.

Jürjens, J., & Juerjens, J. (2005). *Secure systems development with UML* Springer.

Kang, M. H., Park, J. S., & Froscher, J. N. (2001). Access control mechanisms for inter-organizational workflow. *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, 66-74.

Kelly, S., & Tolvanen, J. (2008). *Domain-specific modeling: Enabling full code generation* Wiley. com.

Kibbe, D. C. (2005). Unofficial FAQ of the ASTM continuity of care record (CCR) standard. *Retrieved June, 21*, 2005.

Kibbe, D. C., Phillips, R. L.,Jr, & Green, L. A. (2004). The continuity of care record. *American Family Physician, 70*(7), 1220, 1222-3.

Kleppe, A. G., Warmer, J., Bast, W., & Explained, M. (2003). The model driven architecture: Practice and promise.

Klyne, G., Carroll, J. J., & McBride, B. (2004). Resource description framework (RDF): Concepts and abstract syntax. *W3C Recommendation, 10*

Kofler, M. (2001). *What is MySQL?* Springer.

Kuper, G., Massacci, F., & Rassadko, N. (2005). Generalized XML security views. *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies*, 77-84.

Lange, C. F., & Chaudron, M. R. (2005). Managing model quality in UML-based software development. *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop On*, 7-16.

Lanthaler, M., & Gütl, C. (2012). On using JSON-LD to create evolvable RESTful services. *Proceedings of the Third International Workshop on RESTful Design*, 25-32.

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer, 36*(6), 47-56.

Lee, M., Kim, H., Kim, J., & Lee, J. (2005). StarUML 5.0 developer guide'. *The Open Source UML/MDA Platform*,

Leonardi, E., Bhowmick, S., & Iwaihara, M. (2010). Efficient database-driven evaluation of security clearance for federated access control of dynamic XML documents. *Database Systems for Advanced Applications*, 299-306.

Liebrand, M., Ellis, H., Phillips, C., & Ting, T. (2002). Role delegation for a distributed, unified RBAC/MAC. *Proceedings of Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security King's College,* 29-31.

Liebrand, M., Ellis, H., Phillips, C., Demurjian, S., Ting, T., & Ellis, J. (2003). Role delegation for a resource-based security model. *Research directions in data and applications security* (pp. 37-48) Springer.

Lockhart, J. (2012). Slim framework. Retrieved, 2014, Retrieved from http://www.slimframework.com/

Masse, M. (2011). *REST API design rulebook* O'Reilly Media.

Mazzoleni, P., Bertino, E., Crispo, B., & Sivasubramanian, S. (2006). XACML policy integration algorithms: Not to be confused with XACML policy combination algorithms! *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, 219-227.

Mazzoleni, P., Crispo, B., Sivasubramanian, S., & Bertino, E. (2008). Xacml policy integration algorithms. *ACM Transactions on Information and System Security (TISSEC), 11*(1), 4.

McCarty, B. (2004). *SELinux* O&Reilly.

McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. *W3C Recommendation, 10*(2004-03), 10.

Merkow, M. (1999). cXML: A new taxonomy for E-commerce. *Internet.Com, Insights-EC Outlook, February.See also Http://Ecommerce.Internet.Com/News/Insights/Outlook*,

Microsoft HealthVault. (2014). Retrieved, 2014, Retrieved from https://www.healthvault.com/us/en

Montrieux, L., Jürjens, J., Haley, C. B., Yu, Y., Schobbens, P., & Toussaint, H. (2010). Tool support for code generation from a UMLsec property. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 357-358.

Moore, B., Dean, D., Gerber, A., Wagenknecht, G., & Vanderheyden, P. (2004). Eclipse development. *Using the Graphical Editing Framework and the Eclipse Modeling Framework.IBM RedBooks.(IBM Corp.), 379*

Mouelhi, T., Fleurey, F., Baudry, B., & Le Traon, Y. (2008). A model-based framework for security policy specification, deployment and testing. *Model Driven Engineering Languages and Systems,* , 537-552.

Mtsweni, J. (2012). Exploiting UML and acceleo for developing semantic web services. *Internet Technology and Secured Transactions, 2012 International Conferece For,* 753-758.

Müldner, T., Leighton, G., & Miziołek, J. K. (2009). Parameterized role-based access control policies for XML documents. *Information Security Journal: A Global Perspective, 18*(6), 282-296.

Ogle, J. H., Alluri, P., & Sarasua, W. (2011). MMUCC and MIRE: The role of segmentation in safety analysis. *Proceedings of the Paper Presented at the 90th Annual Meeting of the Transportation Research Board,*

Parastatidis, S., Webber, J., Woodman, S., Kuo, D., & Greenfield, P. (2005). An introduction to the SOAP service description language. *School of Computing Science, University of Newcastle, Newcastle upon Tyne CS-TR-898,*

Pavlich-Mariscal, J. A., Berhe, S., De la Rosa Algarín, A., & Demurjian, S. A. (2014). An integrated secure software engineering approach for functional, collaborative, and information concerns. *Accepted in Handbook of research on emerging advancements and technologies in software engineering* () IGI Global.

Pavlich-Mariscal, J. A. (2008). *A framework of composable security features: Preserving separation of security concerns from models to code* ProQuest.

Pavlich-Mariscal, J. A., Demurjian, S. A., & Michel, L. D. (2008). A framework of composable access control definition, enforcement and assurance. *Chilean Computer Science Society, 2008. SCCC'08. International Conference of The,* 13-22.

Pavlich-Mariscal, J. A., Demurjian, S. A., & Michel, L. D. (2010). A framework for security assurance of access control enforcement code. *Computers & Security, 29*(7), 770-784.

Poernomo, I. (2006). The meta-object facility typed. *Proceedings of the 2006 ACM Symposium on Applied Computing,* 1845-1849.

Popp, G., Jurjens, J., Wimmel, G., & Breu, R. (2003). Security-critical system development with extended use cases. *Software Engineering Conference, 2003. Tenth Asia-Pacific,* 478-487.

Practice fusion. (2014). Retrieved, 2014, Retrieved from http://www.practicefusion.com/

Rahaman, M. A., Roudier, Y., & Schaad, A. (2008). Distributed access control for XML document centric collaborations. *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE,* 267-276.

Ramirez, A., Vanpeperstraete, P., Rueckert, A., Odutola, K., Bennett, J., Tolke, L., & van der Wulp, M. (2003). ArgoUML user manual: A tutorial and reference description.

Randolph, N., Gardner, D., Anderson, C., & Minutillo, M. (2010). *Professional visual studio 2010* John Wiley & Sons.

Rao, P., Lin, D., Bertino, E., Li, N., & Lobo, J. (2009). An algebra for fine-grained integration of XACML policies. *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies,* 63-72.

Reid, R. J., Fishman, P. A., Yu, O., Ross, T. R., Tufano, J. T., Soman, M. P., & Larson, E. B. (2009). Patient-centered medical home demonstration: A prospective, quasi-experimental, before and after evaluation. *The American Journal of Managed Care, 15*(9), e71-87.

Rjaibi, W. (2006). Label-based access control (LBAC) in DB2 LUW. *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap between PST Technologies and Business Services,* 7.

Routledge, N., Bird, L., & Goodchild, A. (2002). UML and XML schema. *Australian Computer Science Communications, , 24*(2) 157-166.

Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON, , 26*(8)

Sachpazidis, I., Rizou, D., & Menary, W. (2008). Satellite based health network in peru and brazil. *Information Technology and Applications in Biomedicine, 2008. ITAB 2008. International Conference On,* 309-314.

Safran, C., Bloomrosen, M., Hammond, W. E., Labkoff, S., Markel-Fox, S., Tang, P. C., & Detmer, D. E. (2007). Toward a national framework for the secondary use of health data: An american medical informatics association white paper. *Journal of the American Medical Informatics Association, 14*(1), 1-9.

Sainz de Abajo, B., & Ballestero, A. (2012). Overview of the most important open source software: Analysis of the benefits of OpenMRS, OpenEMR, and VistA. *Telemedicine and E-Health Services, Policies, and Applications: Advancements and Developments.Hershey (PA): IGI Global, ,* 315-346.

Salted password hashing - doing it right. (2014). Retrieved, 2014, Retrieved from https://crackstation.net/hashing-security.htm

Sandhu, R. S. (1993). Lattice-based access control models. *Computer, 26*(11), 9-19.

Sandhu, R. S., & Samarati, P. (1994). Access control: Principle and practice. *Communications Magazine, IEEE, 32*(9), 40-48.

Schlossnagle, G. (2004). *Advanced PHP programming: A practical guide to developing large-scale web sites and applications with PHP 5* Pearson Education.

Shehab, M., Bertino, E., & Ghafoor, A. (2005). Secure collaboration in mediator-free environments. *Proceedings of the 12th ACM Conference on Computer and Communications Security,* 58-67.

Soley, R. (2000). Model driven architecture. *OMG White Paper, 308*, 308.

Sun, L., & Li, Y. (2006). DTD level authorization in XML documents with usage control. *International Journal of Computer Science and Network Security, 6*(11), 244-250.

Sun, Y., & Pan, P. (2005). PRES: A practical flexible RBAC workflow system. *Proceedings of the 7th International Conference on Electronic Commerce,* 653-658.

Tolone, W., Ahn, G., Pai, T., & Hong, S. (2005). Access control in collaborative systems. *ACM Computing Surveys (CSUR), 37*(1), 29-41.

UML ISO standard. (2014). Retrieved, 2014, Retrieved from http://www.omg.org/spec/UML/

Vogel-Heuser, B., Witsch, D., & Katzke, U. (2005). Automatic code generation from a UML model to IEC 61131-3 and system configuration tools. *Control and Automation, 2005. ICCA'05. International Conference On, , 2* 1034-1039.

WebMD. (2014). Retrieved, 2014, Retrieved from http://www.webmd.com/

Yuan, E., & Tong, J. (2005). Attributed based access control (ABAC) for web services. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference On,*

Zajac, P., Norris, M., & Keenum, A. (2014). Patient-centered medical home. *Osteopathic Family Physician, 6*(1)

Ziminski, T. B., De la Rosa Algarín, A., Saripalle, R., Demurjian, S. A., & Jackson, E. (2012). SMARTSync: Towards patient-driven medication reconciliation using the SMART framework. *Proceedings of 2012 International Workshop on Biomedical and Health Informatics,* 806-813.

Zisman, A. (2007). A static verification framework for secure peer-to-peer applications. *Internet and Web Applications and Services, 2007. ICIW'07. Second International Conference On*, 8-8.

Appendix A

- Carol Smith's Health Level 7 Clinical Document Architecture (HL7 CDA) Instance

```
<?xml version="1.0"?>
<!-- To use the impl_cdar2.xls stylesheet, remove the comment delimiters from the
stylesheet call below. -->
<!-- ?xml-stylesheet type="text/xsl" href="IMPL_CDAR2.xsl"? -->
<ClinicalDocument xmlns="urn:hl7-org:v3" xmlns:mif="urn:hl7-org:v3/mif"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:hl7-org:v3 CDA.xsd">
<typeId root="2.16.840.1.113883.1.3" extension="POCD_HD000040"/>
<id root="2.16.840.1.113883.19.4" extension="c266"/>
<code code="11488-4" codeSystem="2.16.840.1.113883.6.1" codeSystemName="LOINC"
displayName="Consultation note"/>
<effectiveTime value="20051014224411-0500"/>
<confidentialityCode code="N" codeSystem="2.16.840.1.113883.5.25"/>
<recordTarget>
<patientRole>
<id extension="12345" root="2.16.840.1.113883.19.5"/>
<patient>
<name>
<given>Carol</given>
<family>Smith</family>
<suffix></suffix>
</name>
<administrativeGenderCode code="F" codeSystem="2.16.840.1.113883.5.1"/>
<birthTime value="19320924"/>
</patient>
<providerOrganization>
<id root="2.16.840.1.113883.19.5"/>
</providerOrganization>
</patientRole>
</recordTarget>
<author>
<time value="2000040714"/>
<assignedAuthor>
<id extension="KP00017" root="2.16.840.1.113883.19.5"/>
<assignedPerson>
<name>
<given>Brock</given>
<family>Ketchum</family>
<suffix>MD</suffix>
</name>
</assignedPerson>
<representedOrganization>
<id root="2.16.840.1.113883.19.5"/>
</representedOrganization>
</assignedAuthor>
</author>
<custodian>
<assignedCustodian>
<representedCustodianOrganization>
<id root="2.16.840.1.113883.19.5"/>
</representedCustodianOrganization>
</assignedCustodian>
</custodian>
<documentationOf>
<serviceEvent classCode="PCPR">
<code code="xxx" codeSystem="xxx" codeSystemName="xxx" displayName="xxx"/>
<effectiveTime>
<low value="19600127"/>
<high value="20050329"/>
</effectiveTime>
<performer typeCode="PRF">
<functionCode code="PCP" codeSystem="2.16.840.1.113883.5.88"/>
<time>
<low value="1998"/>
```

```
<high value="2005"/>
</time>
<assignedEntity>
<id extension="1" root="1.3.6.4.1.4.1.2835.1"/>
<code code="59058001"
codeSystem="2.16.840.1.113883.6.96"
codeSystemName="SNOMED CT"
displayName="General Physician"/>
<addr>
<streetAddressLine>1 Fake Road</streetAddressLine>
<city>Fake City</city>
<state>KK</state>
<postalCode>123456</postalCode>
<country>USA</country>
</addr>
<telecom value="tel:(999)555-1212" use="WP"/>
<assignedPerson>
<name>
<prefix>Dr.</prefix>
<given>Elisa</given>
<family>Mathison</family>
<suffix></suffix>
</name>
</assignedPerson>
</assignedEntity>
</performer>
</serviceEvent>
</documentationOf>
<component>
<structuredBody>
<component>
<section>
<code code="10164-2" codeSystem="2.16.840.1.113883.6.1"
codeSystemName="LOINC"/>
<title>History of Present Illness</title>
<text>
<content styleCode="Bold">Carol Smith</content>
   is a 31 year old female referred for further asthma management.
   Onset of asthma in her <content revised="delete">twenties</content>
   <content revised="insert">teens</content>.
   She was hospitalized twice last year, and already twice this year.
   She has not been able to be weaned off steroids for the past several
   months.
   </text>
   </section>
   </component>
   </structuredBody>
   </component>
   </ClinicalDocument>
```

- Carol Smith's Continuity of Care Record (CCR) Instance

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <?xml-stylesheet type="text/xsl" href="stylesheet/ccr.xsl"?>
3.  <ContinuityOfCareRecord xmlns="urn:astm-org:CCR">
4.  <CCRDocumentObjectID>Ab13c1971-221a-9724-5d09-9981709c4204</CCRDocumentObjectID>
5.  <Language>
6.  <Text>English</Text>
7.  </Language>
8.  <Version>V1.0</Version>
9.  <DateTime>
10. <ExactDateTime>2014-02-13T15:38:06Z</ExactDateTime>
11. </DateTime>
12. <Patient>
13. <ActorID>A1234</ActorID>
14. </Patient>
15. <From>
16. <ActorLink>
17. <ActorID>Af81882df-f905-e064-61f4-1e06daf00609</ActorID>
18. <ActorRole>
19. <Text>author</Text>
20. </ActorRole>
21. </ActorLink>
22. </From>
23. <To>
24. <ActorLink>
25. <ActorID>A1234</ActorID>
26. <ActorRole>
27. <Text>patient</Text>
28. </ActorRole>
29. </ActorLink>
30. </To>
31. <Purpose>
32. <Description>
33. <Text>Summary of patient information</Text>
34. </Description>
35. </Purpose>
36. <Body>
37. <Problems>
38. <Problem>
39. <CCRDataObjectID>PROB1</CCRDataObjectID>
40. <DateTime>
41. <ExactDateTime>2014-02-13T15:38:06Z</ExactDateTime>
42. </DateTime>
43. <IDs>
44. <ID></ID>
45. <Source>
46. <Actor>
47. <ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
48. </Actor>
49. </Source>
50. </IDs>
51. <Type>
52. <Text>Problem</Text>
53. </Type>
54. <Description>
55. <Text></Text>
56. <Code>
57. <Value></Value>
58. <CodingSystem>ICD9-CM</CodingSystem>
59. </Code>
60. </Description>
61. <Status>
62. <Text>Active</Text>
63. </Status>
64. <Source>
65. <Actor>
66. <ActorID></ActorID>
67. </Actor>
68. </Source>
69. <CommentID></CommentID>
```

```
<Episodes>
<Number/>
<Episode>
<CCRDataObjectID>EP1</CCRDataObjectID>
<Source>
<Actor>
<ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
</Actor>
</Source>
</Episode>
<Source>
<Actor>
<ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
</Actor>
</Source>
</Episodes>
<HealthStatus>
<DateTime>
<ExactDateTime/>
</DateTime>
<Description>
<Text></Text>
</Description>
<Source>
<Actor>
<ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
</Actor>
</Source>
</HealthStatus>
</Problem>
   </Problems>
   <Alerts>
   <Alert>
   <CCRDataObjectID>A789bbc2b-e7ee-51d4-f153-05734e7bd44a</CCRDataObjectID>
   <DateTime>
   <ExactDateTime>2014-02-13T15:38:06Z</ExactDateTime>
   </DateTime>
   <IDs>
   <ID></ID>
   <Source>
   <Actor>
   <ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
   </Actor>
   </Source>
   </IDs>
   <Type>
   <Text>-</Text>
   </Type>
   <Description>
   <Text></Text>
   <Code>
   <Value></Value>
   </Code>
   </Description>
   <Source>
   <Actor>
   <ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
   </Actor>
   </Source>
   <Agent>
   <EnvironmentalAgents>
   <EnvironmentalAgent>
   <CCRDataObjectID>Afed996aa-e012-02f4-8dee-d24177756afa</CCRDataObjectID>
   <DateTime>
   <ExactDateTime></ExactDateTime>
   </DateTime>
   <Description>
   <Text></Text>
   <Code>
   <Value/>
   </Code>
```

```
141.    </Description>
142.    <Status>
143.    <Text></Text>
144.    </Status>
145.    <Source>
146.    <Actor>
147.    <ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
148.    </Actor>
149.    </Source>
150.    </EnvironmentalAgent>
151.    </EnvironmentalAgents>
152.    </Agent>
153.    <Reaction>
154.    <Description>
155.    <Text></Text>
156.    </Description>
157.    <Status>
158.    <Text>None</Text>
159.    </Status>
160.    </Reaction>
161.    </Alert>
162.    </Alerts>
163.    <Medications>
164.    <Medication>
165.    <CCRDataObjectID>Af9ec1fa4-73ea-3c94-e96a-96c76398222d</CCRDataObjectID>
166.    <DateTime>
167.    <ExactDateTime>2014-02-13T15:38:06Z</ExactDateTime>
168.    </DateTime>
169.    <Type>
170.    <Text>Medication</Text>
171.    </Type>
172.    <Status>
173.    <Text></Text>
174.    </Status>
175.    <Source>
176.    <Actor>
177.    <ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
178.    </Actor>
179.    </Source>
180.    <Product>
181.    <ProductName>
182.    <Text></Text>
183.    <Code>
184.    <Value></Value>
185.    <CodingSystem>RxNorm</CodingSystem>
186.    </Code>
187.    </ProductName>
188.    <Strength>
189.    <Value></Value>
190.    <Units>
191.    <Unit></Unit>
192.    </Units>
193.    </Strength>
194.    <Form>
195.    <Text></Text>
196.    </Form>
197.    </Product>
198.    <Quantity>
199.    <Value></Value>
200.    <Units>
201.    <Unit></Unit>
202.    </Units>
203.    </Quantity>
204.    <Directions>
205.    <Direction>
206.    <Description>
207.    <Text></Text>
208.    </Description>
209.    <Route>
210.    <Text>Tablet</Text>
211.    </Route>
```

```
212.    <Site>
213.    <Text>Oral</Text>
214.    </Site>
215.    </Direction>
216.    </Directions>
217.    <PatientInstructions>
218.    <Instruction>
219.    <Text></Text>
220.    </Instruction>
221.    </PatientInstructions>
222.    <Refills>
223.    <Refill>
224.    <Number></Number>
225.    </Refill>
226.    </Refills>
227.    </Medication>
228.    </Medications>
229.    <Immunizations>
230.    <Immunization>
231.    <CCRDataObjectID>Afde47991-11b3-f054-35e4-b0d0ffbeb5a3</CCRDataObjectID>
232.    <DateTime>
233.    <ExactDateTime>2014-02-13T15:38:06Z</ExactDateTime>
234.    </DateTime>
235.    <Type>
236.    <Text>Immunization</Text>
237.    </Type>
238.    <Status>
239.    <Text>ACTIVE</Text>
240.    </Status>
241.    <Source>
242.    <Actor>
243.    <ActorID>A208148c8-f5a7-f044-7d31-a3ea95e003a5</ActorID>
244.    </Actor>
245.    </Source>
246.    <Product>
247.    <ProductName>
248.    <Text></Text>
249.    </ProductName>
250.    </Product>
251.    <Directions>
252.    <Direction>
253.    <Description>
254.    <Text></Text>
255.    <Code>
256.    <Value>None</Value>
257.    </Code>
258.    </Description>
259.    </Direction>
260.    </Directions>
261.    </Immunization>
262.    </Immunizations>
263.    <Results>
264.    <Result>
265.    <CCRDataObjectID>A75d67a94-9ffa-86c4-1979-b57ce822803b</CCRDataObjectID>
266.    <DateTime>
267.    <ExactDateTime>2014-02-13T15:38:06Z</ExactDateTime>
268.    </DateTime>
269.    <IDs>
270.    <ID/>
271.    <Source>
272.    <Actor>
273.    <ActorID>Af81882df-f905-e064-61f4-1e06daf00609</ActorID>
274.    </Actor>
275.    </Source>
276.    </IDs>
277.    <Source>
278.    <Actor>
279.    <ActorID></ActorID>
280.    </Actor>
281.    </Source>
282.    <Test>
```

```
283.    <CCRDataObjectID>A135053dc-a2a1-7124-f510-0a4b6e5ef90a</CCRDataObjectID>
284.    <DateTime>
285.    <ExactDateTime>2014-02-13T15:38:06Z</ExactDateTime>
286.    </DateTime>
287.    <Type>
288.    <Text>Observation</Text>
289.    </Type>
290.    <Description>
291.    <Text></Text>
292.    <Code>
293.    <Value>Value</Value>
294.    </Code>
295.    </Description>
296.    <Source>
297.    <Actor>
298.    <ActorID></ActorID>
299.    </Actor>
300.    </Source>
301.    <TestResult>
302.    <Value></Value>
303.    <Code>
304.    <Value>Value</Value>
305.    </Code>
306.    <Description>
307.    <Text></Text>
308.    </Description>
309.    </TestResult>
310.    <NormalResult>
311.    <Normal>
312.    <Value></Value>
313.    <Units>
314.    <Unit>Test Unit</Unit>
315.    </Units>
316.    <Source>
317.    <Actor>
318.    <ActorID></ActorID>
319.    </Actor>
320.    </Source>
321.    </Normal>
322.    </NormalResult>
323.    <Flag>
324.    <Text></Text>
325.    </Flag>
326.    </Test>
327.    </Result>
328.    </Results>
329.    </Body>
330.    <Actors>
331.    <Actor>
332.    <ActorObjectID>A1234</ActorObjectID>
333.    <Person>
334.    <Name>
335.    <CurrentName>
336.    <Given>CAROL</Given>
337.    <Family>SMITH</Family>
338.    <Suffix/>
339.    </CurrentName>
340.    </Name>
341.    <DateOfBirth>
342.    <ExactDateTime>1983-04-13T00:00:00Z</ExactDateTime>
343.    </DateOfBirth>
344.    <Gender>
345.    <Text>Female</Text>
346.    <Code>
347.    <Value/>
348.    </Code>
349.    </Gender>
350.    </Person>
351.    <IDs>
352.    <Type>
353.    <Text>Patient ID</Text>
```

```
354.    </Type>
355.    <ID>16</ID>
356.    <Source>
357.    <Actor>
358.    <ActorID>A6e0d85e5-b441-22d4-4971-7a05aeb2df8b</ActorID>
359.    </Actor>
360.    </Source>
361.    </IDs>
362.    <Address>
363.    <Type>
364.    <Text>H</Text>
365.    </Type>
366.    <Line1>3814 FirstAve.</Line1>
367.    <City>Madison</City>
368.    <State>ND</State>
369.    <PostalCode>39816</PostalCode>
370.    </Address>
371.    <Telephone>
372.    <Value>(77) 382-5756</Value>
373.    </Telephone>
374.    <Source>
375.    <Actor>
376.    <ActorID>Af81882df-f905-e064-61f4-1e06daf00609</ActorID>
377.    </Actor>
378.    </Source>
379.    </Actor>
380.    <Actor>
381.    <ActorObjectID>Af81882df-f905-e064-61f4-1e06daf00609</ActorObjectID>
382.    <InformationSystem>
383.    <Name>Clinic A</Name>
384.    <Type>Facility</Type>
385.    </InformationSystem>
386.    <IDs>
387.    <Type>
388.    <Text></Text>
389.    </Type>
390.    <ID></ID>
391.    <Source>
392.    <Actor>
393.    <ActorID>Af81882df-f905-e064-61f4-1e06daf00609</ActorID>
394.    </Actor>
395.    </Source>
396.    </IDs>
397.    <Address>
398.    <Type>
399.    <Text>WP</Text>
400.    </Type>
401.    <Line1></Line1>
402.    <City>Farmington</City>
403.    <State>CT </State>
404.    <PostalCode>06030</PostalCode>
405.    </Address>
406.    <Telephone>
407.    <Value>860-679-2000</Value>
408.    </Telephone>
409.    <Source>
410.    <Actor>
411.    <ActorID>Af81882df-f905-e064-61f4-1e06daf00609</ActorID>
412.    </Actor>
413.    </Source>
414.    </Actor>
415.    <Actor>
416.    <ActorObjectID>Aa04b1505-4651-d224-a1a2-d4d8ea6b4b23</ActorObjectID>
417.    <InformationSystem>
418.    <Name>OEMR</Name>
419.    <Type>OpenEMR</Type>
420.    <Version>4.x</Version>
421.    </InformationSystem>
422.    <IDs>
423.    <Type>
424.    <Text>Certification #</Text>
```

```
425.    </Type>
426.    <ID>EHRX-OEMRXXXXXX-2011</ID>
427.    <Source>
428.    <Actor>
429.    <ActorID>Af81882df-f905-e064-61f4-1e06daf00609</ActorID>
430.    </Actor>
431.    </Source>
432.    </IDs>
433.    <Address>
434.    <Type>
435.    <Text>WP</Text>
436.    </Type>
437.    <Line1>2365 Springs Rd. NE</Line1>
438.    <City>Hickory</City>
439.    <State>NC </State>
440.    <PostalCode>28601</PostalCode>
441.    </Address>
442.    <Telephone>
443.    <Value>000-000-0000</Value>
444.    </Telephone>
445.    <Source>
446.    <Actor>
447.    <ActorID>Af81882df-f905-e064-61f4-1e06daf00609</ActorID>
448.    </Actor>
449.    </Source>
450.    </Actor>
451.    </Actors>
452.    </ContinuityOfCareRecord>
```

- Carol Smith's Continuity of Care Record (CCR) Instance in XML Tree Editor

Appendix B

XACML Policy and Rule Combination Algorithms in Pseudo-code

- Permit-Overrides:

```
1.  Decision permitOverridesCombiningAlgorithm(Decision[] decisions)
2.  {
3.    Boolean atLeastOneErrorD  = false;
4.    Boolean atLeastOneErrorP  = false;
5.    Boolean atLeastOneErrorDP  = false;
6.    Boolean atLeastOneDeny = false;
7.
8.    for( i=0 ; i < lengthOf(decisions) ; i++ )
9.    {
10.     Decision decision = decisions[i];
11.     if (decision == Deny
12.     {
13.       atLeastOneDeny = true;
14.       continue;
15.     }
16.     if (decision == Permit)
17.     {
18.       return Permit;
19.     }
20.     if (decision == NotApplicable)
21.     {
22.       continue;
23.     }
24.     if (decision == Indeterminate{D})
25.     {
26.       atLeastOneErrorD = true;
27.       continue;
28.     }
29.     if (decision == Indeterminate{P})
30.     {
31.       atLeastOneErrorP = true;
32.       continue;
33.     }
34.     if (decision == Indeterminate{DP})
35.     {
36.       atLeastOneErrorDP = true;
37.       continue;
38.     }
39.     if (atLeastOneErrorDP)
40.     {
41.       return Indeterminate{DP};
42.     }
43.     if (atLeastOneErrorP && (atLeastOneErrorD || atLeastOneDeny))
44.     {
45.       return Indeterminate{DP};
46.     }
47.     if (atLeastOneErrorP)
48.     {
49.       return Indeterminate{P};
50.     }
51.     if (atLeastOneDeny)
52.     {
53.       return Deny;
54.     }
55.     if (atLeastOneErrorD)
56.     {
57.       return Indeterminate{D};
58.     }
59.   }
60.   return NotApplicable;
61. }
```

- **Deny-Overrides**:

```
1.  Decision denyOverridesCombiningAlgorithm(Decision[] decisions)
2.  {
3.    Boolean atLeastOneErrorD  = false;
4.    Boolean atLeastOneErrorP  = false;
5.    Boolean atLeastOneErrorDP  = false;
6.    Boolean atLeastOnePermit = false;
7.
8.    for( i=0 ; i < lengthOf(decisions) ; i++ )
9.    {
10.     Decision decision = decisions[i];
11.     if (decision == Deny)
12.     {
13.       return Deny;
14.     }
15.     if (decision == Permit)
16.     {
17.       atLeastOnePermit = true;
18.       continue;
19.     }
20.     if (decision == NotApplicable)
21.     {
22.       continue;
23.     }
24.     if (decision == Indeterminate{D})
25.     {
26.       atLeastOneErrorD = true;
27.       continue;
28.     }
29.     if (decision == Indeterminate{P})
30.     {
31.       atLeastOneErrorP = true;
32.       continue;
33.     }
34.     if (decision == Indeterminate{DP})
35.     {
36.       atLeastOneErrorDP = true;
37.       continue;
38.     }
39.     if (atLeastOneErrorDP)
40.     {
41.       return Indeterminate{DP};
42.     }
43.     if (atLeastOneErrorD && (atLeastOneErrorP || atLeastOnePermit))
44.     {
45.       return Indeterminate{DP};
46.     }
47.     if (atLeastOneErrorD)
48.     {
49.       return Indeterminate{D};
50.     }
51.     if (atLeastOnePermit)
52.     {
53.       return Permit;
54.     }
55.     if (atLeastOneErrorP)
56.     {
57.       return Indeterminate{P};
58.     }
59.   }
60.   return NotApplicable;
61. }
```

- First-applicable:

```
1.  Decision firstApplicableEffectCombiningAlgorithm(RuleOrPolicies[] rulesOrPolicies)
2.  {
3.    for( i = 0 ; i < lengthOf(rulesOrPolicies) ; i++ )
4.    {
5.      Decision decision = evaluate(rulesOrPolicies[i]);
6.      if (decision == Deny)
7.      {
8.        return Deny;
9.      }
10.     if (decision == Permit)
11.     {
12.       return Permit;
13.     }
14.     if (decision == NotApplicable)
15.     {
16.       continue;
17.     }
18.     if (decision == Indeterminate)
19.     {
20.       return Indeterminate;
21.     }
22.   }
23.   return NotApplicable;
24. }
```

- Only-one-applicable:

```
1.  Decision onlyOneApplicableCombiningAlogrithm(RuleOrPolicies[] rulesOrPolicies)
2.  {
3.    Boolen atLeastOne = false;
4.    RuleOrPolicy selectedRuleOrPolicy = null;
5.    ApplicableResult appResult;
6.    for ( i = 0; i < lengthOf(rulesOrPolicies) ; i++ )
7.    {
8.      appResult = isApplicable(rulesOrPolicies[I]);
9.      if ( appResult == Indeterminate )
10.     {
11.       return Indeterminate;
12.     }
13.     if( appResult == Applicable )
14.     {
15.       if ( atLeastOne )
16.       {
17.         return Indeterminate;
18.       }
19.       else
20.       {
21.         atLeastOne = true;
22.         selectedRuleOrPolicy = rulesOrPolicies[i];
23.       }
24.     }
25.     if ( appResult == NotApplicable )
26.     {
27.       continue;
28.     }
29.   }
30.   if ( atLeastOne )
31.   {
32.     return evaluate(selectedRuleOrPolicy);
33.   }
34.   }
35.   else
36.   {
37.     return NotApplicable;
38.   }
39. }
```

Appendix C

- User Elisa with Role Physician's XACML Policy Instance for the Healthcare Scenario

```
1.  <Policy PolicyId="ada-policy" RuleCombiningAlgId="deny-overrides">
2.  <Description>Omitted due to length.</Description>
3.  <Target>
4.   <Subjects>
5.    <user><id>6</id><name>Elisa</name></user>
6.   </Subjects>
7.   <Resources><AnyResource/></Resources>
8.   <Actions><AnyAction/></Actions>
9.  </Target>
10. <Rule RuleId="simple-RBAC+LBAC-rule" Effect="Permit">
11.  <Target>
12.   <Subjects>
13.    <role><roleID>5</roleID><roleName>Physician</roleName></role>
14.   </Subjects>
15.   <Resources><element>
16.     <elementID>el-3</elementID>
17.     <elementName>Past Medical History</elementName>
18.   </element></Resources>
19.   <Actions><operation>
20.     <operationName>insert</operationName>
21.     <opAccessMode>write</opAccessMode>
22.   </operation></Actions>
23.  </Target>
24.  <Condition>
25.   <Apply FunctionId="…:integer-greater-than-or-equal">
26.    <Apply FunctionId="…:integer-one-and-only">
27.     <AttributeValue DataType="…#integer">
28.      Secret
29.     </AttributeValue>
30.    </Apply>
31.    <AttributeValue DataType="…#integer">
32.     Secret
33.    </AttributeValue>
34.   </Apply>
35.  </Condition>
36. </Rule>
37. <Rule RuleId="simple-delegation-rule" Effect="Permit">
38.  <Target>
39.   <Subjects>
40.    <user><id>6</id><name>Elisa</name></user>
41.   </Subjects>
42.   <Resources>
43.    <Roles><role>
44.      <roleID>2</roleID><roleName>Physician</roleName>
45.    </role></Roles>
46.   </Resources>
47.   <DelegationTargets>
48.    <user><id>30</id><name>Samantha</name></user>
49.   </DelegationTargets>
50.  </Target>
51. </Rule>
52. <Rule RuleId="simple-authorization-rule" Effect="Permit">
53.  <Target>
54.   <Subjects>
55.    <user><id>6</id><name>Elisa</name></user>
56.   </Subjects>
57.   <Resources><Schemas><schema>
58.    <schemaID>4</schemaID>
59.    <schemaName>Schema 4</schemaName>
60.    </schema></Schemas>
61.    <Instances><instance>
62.     <instanceID>4,2</instaneID>
63.     <instanceName>Carol Smith Health Record</instanceName>
```

```
64.    </instance></Instances></Resources>
65.  </Target>
66. </Rule>
67. </Policy>
```