ABSTRACT

SLANKAS, JOHN BOTIOUS. Implementing Database Access Control Policy from Unconstrained Natural Language Text. (Under the direction of Dr. Laurie Williams.)

With over forty years of use and refinement, access control, often in the form of access control rules (ACRs), remains a significant and widely-used control mechanism for information security. ACRs regulate who can perform specific actions on specific resources within a software-intensive system and are considered a critical component to ensure both confidentiality and integrity. Although software can and does implement access control at the application layer, failure to enforce data access controls at the persistence layer may allow uncontrolled data access when individuals bypass application controls or the ACRs are inconsistently implemented. *Our research goal is to improve security and compliance by ensuring access control rules explicitly and implicitly defined within unconstrained natural language product artifacts are appropriately enforced within a system's relational database.* Access control implemented in both the application and persistence layers strongly supports a defense in depth strategy. We specify a tool-based process to 1) parse existing, unconstrained natural language product artifacts; 2) classify whether or not a sentence in the product artifact implies access control and whether or not the sentence implies database model elements; and, as appropriate, 3) extract ACR elements; 4) extract database model elements; 5) map extracted data model to a database schema; and 6) implement role-based access control (RBAC) within a relational database. To validate our process, we examined these steps in three studies.

In our first study, we demonstrated a technique to parse natural language texts (step 1) and how to most effectively classify statements for particular properties (step 2). Specifically, we classified which sentences explicitly or implicitly contain non-functional requirements

(NFRs) among 14 NFR categories (e.g. capacity, reliability, and security). Our ensemble-based classifier had a F_1 measure of 0.68 for identifying security-related objectives.

In our second study, we identified sentences containing ACRs (step 2) and then inferred those rules from the text (step 3). To infer the rules, we extract semantic relations (i.e., the connection(s) among two or more items) from natural language (NL) artifacts such as requirements documents. Unlike existing approaches, our approach combines techniques from information extraction and machine learning. Our evaluation results show that ACRs exist in 47% of the sentences, and our approach effectively identifies those ACR sentences with a precision of 81% and recall of 65%. Our approach extracts ACRs from those identified ACR sentences with an average precision of 76% and an average recall of 49%.

In our final study, we followed the process from start to finish with a focus on identifying sentences containing database model elements (step 2), extracting the database model (step 4), mapping the extracting database model elements to the physical database schema (step 5), and implementing RBAC (step 6) within an applications database. Our approach extends existing work to extract conceptual database models from NL artifacts using relations based upon heuristics defined in those works, but then adopts machine learning techniques to learn new rules to extract database entities, attributes, and relationships. We then present guidance and tool support for mapping the extracted data model to a system's physical database schema. We performed system testing on the application to ensure the database appropriately enforces access control as defined. Our approach extracts database elements from the text with a F_1score of 0.90 utilizing the process' ACR extraction as a starting point.

Implementing Database Access Control Policy from Unconstrained Natural Language Text

by
John Botious Slankas

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2015

APPROVED BY:

Laurie Williams, Ph.D. Committee Chair	Rada Chirkova, Ph.D.
Julie Earp, Ph.D.	Emerson Murphy-Hill, Ph.D.

DEDICATION

To my son Joseph, who continues to teach and amaze me.

To my wife, Rachel, for making my life complete.

To my parents, Tom and Maggie, for providing a foundation and always being there.

BIOGRAPHY

John Botious Slankas was born in Albuquerque, New Mexico on July 11th, 1970. He graduated from Casa Roble Fundamental High School in 1988. While there, he was active in sports (water polo, wrestling, and tennis) and he started the school's computer club. He graduated from the University of Notre Dame with a Bachelor of Science in Mathematics in 1992. At the same time, he was commissioned a second lieutenant in the United States Air Force Reserves. Prior to active duty, John graduated with a Master of Science in Computer Science from the University of Arizona in December, 1993. He then spent four and half years on active duty attaining the rank of "captain" as a communications-computer officer. John then worked for Sybase as a senior consultant. Next, John worked at Wachovia Corporation for ten years leading teams in developing intranet-based solutions for the human resources and information security groups. In 2009, he made a decision to return to school full-time to pursue a Doctorate in Philosophy at North Carolina State University (NCSU). Currently, John is a senior research scholar for the Laboratory for Analytic Sciences at NCSU.

ACKNOWLEDGMENTS

While this dissertation only lists one name as the author, many individuals have had significant contributions.

First and foremost, I would like to thank my advisor, Laurie Williams for her enthusiasm, patience, encouragement, understanding, guidance, funding, and feedback. Despite her monotonically increasing busy schedule, Laurie always finds time for her students. Her reviews of my work have been invaluable. Thank you to my committee members for their time and much-needed advice: Rada Chirkova, Julie Earp, Emerson Murphy-Hill, and Tao Xie.

I owe a special thank you to all of the members of Laurie's research group: Raza Abbas, Andrew Austin, Amit Banthiya, Pat Francis, Eric Helms, JeeHyun Hwang, Da Young Lee, Jason King, Andy Meneely, Pat Morrison, Rahul Pandita, Akond Rahman, Maria Riaz, Yonghee Shin, Ben Smith, Will Snipes, and Chris Theisen. All of these individuals have read my papers and provided critical feedback. Many of them helped me classify sentences and other elements required for this research – you have my deep gratitude for your labor and assistance. Similarly, many members of Emerson Murphy-Hill's research group, the Developer Liberation Front have provided feedback and friendship while sharing lab space: Titus Barik, Xi Ge, Brittany Johnson, Kevin Lubick, Yoonki Song, and Jim Witschey. A special thank you to Julio Bahamon for keeping me sane by joining me at many NCSU football and basketball games and to Jason Gionta for keep me sane as fellow student in many of our initial classes. Xusheng Xiao was a wonderful co-author to have our comparison paper on our two approaches to a common problem.

I have been blessed to have many great teachers throughout my life. Every teacher has had some level of influence on me. My high school history teacher, Douglas Rennie, taught me more about writing well than anyone else. My high school science teacher, Tom Smithson, was truly my first mentor. My professors at the University of Notre Dame, the University of Arizona and North Carolina State University truly pushed me to master the subjects they taught.

I also need to thank my supervisors and co-workers at the Laboratory for Analytic Science. Each day I come to work, I am fortunate to work with a great team, full of outstanding individuals who mentally challenge me daily. Thank you to Marc Hoit for the recommendation and to Randy Avent for hiring me. Thank you as well to Forrest Allen and Alyson Wilson for their patience as this dissertation was completed.

Throughout my life, my parents have always been present and supportive. As with all my other accomplishments to date, this work would not have happened without their presence. They sparked my love for computers and Computer Science over thirty years ago with a purchase of an Apple][+ computer.

Finally, and most definitely not least, I would like to thank my wife, Rachel for her love, companionship, and patience. She has endured many days of me tied to a computer screen while working on research as well as trying to decipher explanations of my work.

This work was supported by the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI) and by the USA National Security Agency (NSA) Science of Security Lablet. Any opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSA.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A	accuracy
AC	access control
ACID	atomicity, consistency, integrity, durability
ACR	access control rule
ACRE	Access Control Rule Extraction
AU	audit
AV	availability
BASE	basically available, soft state, eventual consistency
CHL	challenge
CNL	controlled natural language
DAC	discretionary access control
DBMS	database management system
DME	database model element
DUA	data use agreement
EHR	electronic health record
ERD	entity-relationship diagram
FN	false negative
FP	false positive
HIPAA	Health Insurance Portability and Accountability Act
IE	information extraction
IR	information retrieval
JDBC	Java Database Connectivity
LG	legal
LF	look and feel
MAC	mandatory access control
ML	machine learning
MST	minimal spanning tree
MT	maintenance
NA	not applicable
NER	named entity recognition
NFR	non-functional requirement
NL	natural language
NLP	natural language processing
OCR	optical character recognition
OP	operations
P	precision
POS	part of speech
PR	privacy
PS	performance and scalability
R	recall
RBAC	role-based access control

RC	Recoverability
RDBMS	relational database management system
RE	relation extraction
REDE	Role Extraction and Database Enforcement
RFP	request for proposal
RL	reliability
RQ	research question
SC	security
SD	standard deviation
SQL	structured query language
SR	sentence representation
SLR	systematic literature review
SM	sequential minimal optimization
STDR	Stanford Type Dependency Representation
SVM	support vector machine
TN	true negative
TP	true positive
US	usability
XACML	eXtensible Access Control Markup Language

1 Introduction

Organizations face data security issues on a daily basis through both a need to protect information from unauthorized users as well as to comply with applicable regulations and standards. Specifically, authorization issues continue to be a significant concern for most software systems. In the 2011 CWE/SANS Top 25 Most Dangerous Software Errors [2011 CWE/SANS Top 25 Most Dangerous Software Errors, 2011], 30% of the errors directly relate to access control[1]. Another security issue, data breaches of confidential information, regularly appears in the news. The Privacy Rights Clearinghouse[2] reported 535 distinct data breaches involving over 30 million sensitive records for 2011 [Data Breaches: A Year in Review, 2011]. In March of 2015, the Washington Post already declared 2015 as the "year of health-care hack" due to the number of health-care related data breaches in January and February [Peterson, 2015]. These breaches cost organizations significant amounts of money from many factors: purchasing credit monitoring for affected individuals, loss of revenue, cost to correct existing deficiencies in systems, reputational loss, and legal fees. A data breach of just 13,687 records cost a non-profit healthcare consultancy firm over $300,000 [Digital Data on Patients Raises Risk of Breaches, 2011]. Data breaches also cost individuals directly. In 2014, over $16 billion was stolen from 12.7 million people in the United States due to identity theft. Over two-thirds of those individuals were previously notified of a data breach within the past year [Pascual & Miller, 2015].

---

[1] CWE-306,862,798,829,250,863,732,307,807
[2] https://www.privacyrights.org/

To mitigate data security issues, organizations must properly implement access control across all system components. With over forty years of use and refinement, access control, often in the form of access control rules (ACRs), remains a significant and widely-used control mechanism for information security. ACRs regulate who can perform specific actions on specific resources within a software-intensive system and are considered a critical component to ensure both confidentiality and integrity [Samarati & Vimercati, 2001]. Developers often utilize role-based access control (RBAC), considered the “gold standard” [Donston-miller, 2011], to enforce authorization policies within systems. Within RBAC, ACRs are associated with specific roles rather than individuals, who are assign to those roles based upon their responsibilities [Sandhu et al., 1996]. However, many software systems implement access control only at the application level. If individuals breach those controls through malicious activity or defective implementation, no additional layers of security exist to protect data within the database. Insider threats exist if individuals can bypass application level security and access data in the persistence layer. Additionally, programmers can intentionally or inadvertently implement functionality that violates the RBAC policy. A defense in depth security approach requires access control at *both* the application and persistence layers. However, creating and defining RBAC policy for the persistence layer can be a tedious, time-consuming, and error-prone endeavor. Developers must sift through existing documentation, application code, and database implementations.

Natural language processing (NLP) emerges as a possible solution to generate appropriate RBAC policies. Luisa et al. [Luisa et al., 2004] reported that 79% of all requirement documents are written in natural language texts. NLP has made significant gains with many practical, real-world systems now in general use. Companies utilize automated speech recognition system to handle routine customer inquiries such as a checking an airline's flight status. IBM's Watson has gone beyond winning Jeopardy and now provides cognitive interactive capability in such fields as healthcare and finance [IBM Watson: Ushering in a new era of computing, 2012]. While prior work exists to extract ACRs from natural language product artifacts, work remains to improve generalizability and ensure systems correctly implement the defined ACRs.

Our research goal is to improve security and compliance by ensuring access control rules explicitly and implicitly defined within unconstrained natural language product artifacts are appropriately enforced within a system's relational database.

1.1 Approach and Challenges

To meet the overall goal, we specify a process, which we call Role Extraction and Database Enforcement (REDE), which allows organizations to utilize existing, unconstrained natural language product artifacts to generate RBAC policies for databases. To assist users in following this process, we have implemented an open-source tool. REDE combines NLP, information extraction (IE), and machine learning (ML) techniques. Internally, REDE represents each sentence as a type dependency graph [de Marneffe et al., 2006] that shows words as vertices and relationships between words as edges. Figure 1 shows the graph for the sentence "The user enters a grade for each student".

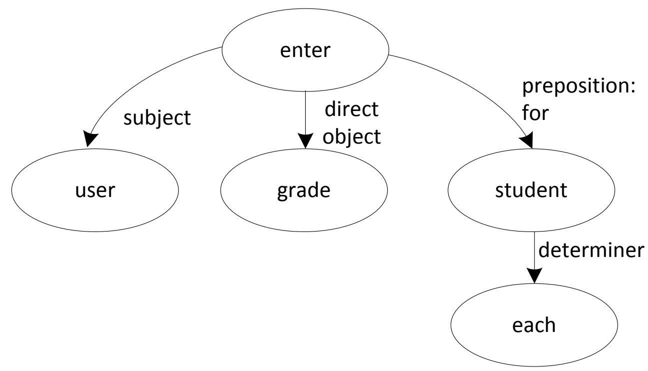


Figure 1. Type Dependency Graph

To produce these graphs, REDE parses each sentence in the artifact with the Stanford Natural Language Parser[3]. REDE then classifies each sentence as to whether or not the sentence contains ACRs and whether or not the sentence contains database model elements[4] (DMEs). For the example sentence, REDE would classify the sentence as true for both implications.

To extract the ACRs, REDE searches all of the graphs whose sentences imply access control for a basic pattern of *noun (subject)–verb–noun (direct object)*. REDE creates a frequency table of all of the discovered subjects and direct objects. The process assumes any subject or direct object that occurs more times than the median count to be a legitimate subject or resource (direct object) for the system. REDE then searches all combinations of the discovered subjects and resources within the graphs. For each found combination, the approach extracts the minimal spanning tree (MST) and assumes that any verb within the tree

[3] http://nlp.stanford.edu/software/corenlp.shtml

[4] Database model elements include entities, attributes of entities, and relationships between entities.

corresponds to the action. REDE adds the extracted MST to a set of valid REDE patterns. REDE then expands the pattern set by allowing a subject or resource vertex within the pattern to match any word of the same part of speech in other sentence graphs (i.e., “wildcarded”). REDE then searches all of the sentence graphs for instances of the ACR patterns. The found instances are assumed to represent ACRs, and the elements of the ACR are then extracted. REDE iterates these steps, searching for additional patterns until no more patterns can be found in the sentences. To minimize incorrect patterns from being created and applied, REDE constructs a naïve Bayes classifier to accept or reject patterns based upon domain-independent features found in other product artifacts (including those from other systems). In this example, REDE would extract two ACRs: 1) user; enter; grade; create and 2) user; enter; student; read. The user needs create permission to store a new grade in the database. Additionally, the user needs to be able to read the student record to know whose grade is being entered. Permissions are derived from a combination of the action of the ACR and structural elements between the action and object within the sentence.

To extract the DMEs, REDE uses a similar process as it does for ACRs. REDE first considers all of the actors and resources found in the ACR extraction step to be entities. REDE assumes any ACR with create or update permission identifies a database relationship. For these entities and relations, REDE applies eight basic patterns adapted from prior work in extracting data models from natural language [Chen, 1983; Hartmann & Link, 2007]. These initial patterns are then expanded by wildcarding different elements of the pattern. REDE then continually searches all of the sentence graphs to find a new location where one of the known patterns can be applied. Once the pattern is applied, the entity, attribute or relation elements are extracted. If new elements are found, then the basic patterns are applied to those elements to form new patterns. Those patterns are then wildcarded, and the process repeats until new elements are discovered. To minimize incorrect patterns from being created and applied, REDE constructs a naïve Bayes classifier to accept or reject patterns based upon domain-independent features found in other product artifacts. In this example, REDE would extract three entities [user, grade, student] and two relationships [enter(user, grade) and enter(grade, student)]. The actual patterns would contain the relationships among the vertices so that the "grades for student" would be appropriately captured.

To map the DMEs and ACRs to a physical database schema, REDE utilizes a series of steps. First, REDE creates additional ACRs based upon database relations that are not already ACRs. As the process creates access control rules at the table level, the process checks any ACRs defined with an attribute as the resource and replaces the attribute with the corresponding entity. Next, REDE resolves ambiguous or missing subjects by using the last known defined subject. REDE then resolves ambiguous resources (i.e., those objects defined by pronouns or ambiguous terms such as "data") by using the last known defined resource. REDE next merges any ACRs sharing the same subjects and resources. Finally, in conjunction with guidance from the user, REDE maps resources in the ACRs to the defined physical database schema for an application. Similarly, the process maps subjects to roles. Once the mapping is complete, REDE performs consistency and completeness checks. Finally, the process generates the SQL commands to implement RBAC in the system's persistence layer.

To successfully implement REDE, a number of challenges must be solved:

CHL1: Identify natural language text that contains access control policies and database objects;

CHL2: Extract various ACR elements (subject, objects, actions) correctly from the text;

CHL3: Extract DMEs correctly from the text; and

CHL4: Map the extracted database model from the natural language text to specific database tables.

1.2 Research Questions and Associated Studies

From the above challenges, we investigate these corresponding research questions:

RQ1: How effectively does REDE find sentences within natural language product artifacts indicating ACRs and DMEs? What features and algorithms are best suited to find such sentences?

RQ2: How effectively does REDE extract access ACR elements from the natural language product artifacts with semantic relation extraction? What patterns exist among sentences with ACRs?

RQ3: How effectively does REDE extract DMEs correctly from the text? What patterns exist among such sentences? How frequently do different types of database model elements appear within the product artifacts?

RQ4: How effectively does REDE map the extracted database model to an actual database schema?

We also add the following question to test the overall effectiveness of the system:

RQ5: How effectively does REDE implement access control within a system's relational database?

To address these questions, we utilized the following studies:

1. *Classification Study* [Slankas & Williams, 2013a] (RQ1): Collected a corpus of software document artifacts. As appropriate, labeled each sentence as to whether or not it contains specific types of non-functional requirements (NFRs) as well as if it implies a part of the database model. Evaluated different classification techniques and features utilizing the corpus to identify those sentences containing access control statements, database model elements, or both.
2. *Access Control Extraction Study* [Slankas & Williams, 2013b; Slankas et al., 2014] (RQ1, RQ2): Evaluated different techniques, parameters, and features to extract ACRs from natural language product artifacts.
3. *Database Model Extraction Study* [Slankas & Williams, 2015] (RQ1, RQ3, RQ4, RQ5): Evaluated different techniques, parameters, and features to extract database model from natural language product artifacts. Implement the resulting role-based access control within a system's database and evaluate its effectiveness. This study includes an end-to-end evaluation of the complete system.

1.3 Thesis Statement and Contributions

The following thesis statement summarizes this dissertation:

Access control rules explicitly and implicitly defined within unconstrained natural language product artifacts can be effectively identified and extracted; database model elements can be effectively identified and extracted; mappings can be identified among the access control rules, database model elements, and *the physical database implementation; and role-based access control can be established within a system's relational database.*

This dissertation makes the following contributions:

- Process to identify NFRs by categories within available natural language documentation;
- Distribution report containing the NFR categories and frequencies by document type;
- Empirical performance results for machine learning classifiers on NFRs;
- Sentence similarity algorithm for use with a *k*-nearest neighbor classifier or a *k*-medoids clustering algorithm;
- Bootstrapping mechanism to seed and iteratively discover ACR patterns in NL text;
- Bootstrapping mechanism to seed and iteratively discover DME patterns in NL text;
- Approach and supporting tool (built upon the combination of NLP, IE, and ML techniques) to extract ACRs and DMEs;
- Evaluation of ACRE for ACRs against product artifacts from systems in three domains: conference management, education, and healthcare;
- Start to finish evaluation of REDE for an open-source conference management system.
- Open-source tool to assist users in following REDE[5]; and

[5] https://github.com/RealsearchGroup/REDE

- Publically-available labeled corpus of identified NFRs[6], a corpus of identified ACRs and sentences[7] and a corpus of identified database elements[8].

1.4 Dissertation Structure

The rest of this dissertation is organized as follows.

- Chapter 2 presents the requisite background material for this dissertation with information on access control, database management systems, machine learning, and NLP.
- Chapter 3 presents the related work for classification, access control extraction, and other topics.
- Chapter 4 provides a detailed overview of REDE with several examples
- Chapter 5 describes our initial study in identifying sentences that belong to specific categories of (NFRs).
- Chapter 6 presents our second study on extracting access control elements from natural language product artifacts
- Chapter 7 describes our final study on extracting database model elements and then implemented the associated RBAC model in the system's database.
- Chapter 8 concludes the dissertation with a summary of our results and future research.

[6] https://github.com/RealsearchGroup/NFRLocator
[7] https://sites.google.com/site/AccessControlRuleExtraction
[8] https://github.com/RealsearchGroup/REDE/tree/master/data

2 Background

This chapter provides the requisite background knowledge for the dissertation. Section 2.1 discusses access control. Section 2.2 presents databases including a discussion of how RBAC is implemented within databases. Section 2.3 covers an overview of the machine learning techniques used within this dissertation. Section 2.4 describes NLP. Finally, Section 2.5 discusses the evaluation criteria for our research.

2.1 Access Control

Access control mechanisms regulate who can perform a specific action on specific resources within computer-based systems. In research performed collaboratively with Riaz and King [Riaz et al., 2014], we found that security objectives could be placed into six main categories: confidentiality; integrity; identification and authentication; availability; accountability; and privacy. Systems can implement access control mechanisms to help achieve each of these security objectives. Confidentiality requires controls to allow and prohibit specific individuals and processes from reading specific data. Integrity requires access control to prohibit improper modification. Access control mechanisms are used to establish identity and the identity's validity for identification and authentication. By limiting who can access systems or parts of systems, access control can help ensure availability. Identity establishment allows traces to be generated back to the responsible actor for accountability. Access control can limit who can see what data and for what purposes to ensure privacy is maintained.

The origination of access control can be tied back to seminal research performed on operating systems back in the 1960s. Very early, researchers recognized the need for "protection" to control system resources to specific processes [Ackerman & Plummer, 1967]. In 1971, Lampson [Lampson, 1971] formed the initial formal models for access control in the paper, "Protection". In this work, he characterized protection as an access matrix that defined access rights from domains (process, user, separate machine, etc.) to objects (other processes, files, memory segments, etc). Bell and LaPadula [Bell & LaPadula, 1973] provided a mathematical foundation to model a secure computer system. Bell and LaPadula also established the concepts for multilevel security policies. Saltzer and Schroeder [Saltzer & Schroeder, 1975] presented the first comprehensive overview of access control and security principles in their work, "The Protection of Information in Computer Systems". In 1983, the United States Department Defense [Defense, 1985] published the "Trusted Computer System Evaluation Criteria". The criteria established security features developers should build into products and systems that process classified or other sensitive data. Most notably, the criteria defined discretionary access control (DAC) and mandatory access control (MAC) which are still widely used today. In 1992, Ferraiolo and Kuhn [Ferraiolo & Kuhn, 1992] created a new model, role-based access control. The 1990s and 2000s saw a number of extensions to the RBAC model, but most can be summarized as additional characteristics upon which to make access control decisions [Helms et al., 2014].

Most access control models utilize a tuple containing at least a subject, a resource (object), and an action to represent a rule as to whether or not the subject (a user) can perform the requested action on the specified resource (object) within the system (i.e., the Clark-Wilson access control triple [Clark & Wilson, 1987]). Discretionary access control (DAC) [Defense, 1985], mandatory access control (MAC) [Defense, 1985], and role-based access control (RBAC) [Ferraiolo & Kuhn, 1992] are three popular access control models. Each of these models control access based on the subject's identity and what permissions they have been granted, but differ in how the access is granted. In DAC, the resource's owners have the authority to decide who can access the resource. In MAC, organizational rules determine who can access data. Military and intelligence systems often utilize MAC to ensure users have the appropriate security clearances to access data. Finally, RBAC controls access based on rules granted to roles with users assigned to roles. In 2000, Osborn et al. [Osborn et al., 2000] demonstrated how to implement both DAC and MAC with RBAC.

Our process utilizes an RBAC model. The subjects extracted during the process form the set of available roles, the extracted actions form the set of transactions, and the extracted objects form the set of objects to be protected.

2.2 Databases

Databases, "a collection of related data" [Elmasri & Navathe, 2010], form a critical component of a multitude of computer systems in use today. Medical appointments, airline reservations, bank records, and countless other types of data are stored within databases. Given the vast amount of data present, developers need to ensure such systems are more than just adequately protected.

2.2.1 History of Databases

Databases have been integral part of computer systems since the 1960's with the advent of navigational and networked-based databases [Elmasri & Navathe, 2010]. While these systems (most notably, IMS) are still in use today [DB-Engines, 2013], they have been surpassed in usage by relational databases. Relational databases trace their origins back to Codd's seminal paper, "A Relational Model of Data for Large Shared Data Banks" [Codd, 1970]. Since then, new functionality has consistently been added to relational databases to include spatial functionality, large object storage, replication, multiple access control models, and many other features. Object DBMSs appeared in the mid-1980's in response to the "perceived" object-relational mismatch problem. However, object DBMSs never gained significant market share, and much of their functionality has been incorporated into mainstream relational DBMSs and the SQL standard itself [Elmasri & Navathe, 2010][DB-Engines, 2013]. Due to scalability issues within several leading internet sites, a new class of DBMSs appeared in the mid-2000's – "NoSQL" DBMSs. While a wide variety of NoSQL DBMSs exist (and the term itself now encompasses any database that is not relational [Sadalage & Fowler, 2012]), most such databases tend to sacrifice or redefine consistency to increase availability (and hence, scalability). "New SQL" DBMSs have emerged since 2010. These systems adhere to the traditional relational model but have new database engines and architectures to handle similar loads compared to "NoSQL" DBMSs [Stonebraker, 2011].

2.2.2 Relational Database Management Systems

A relational DBMS places data into groups of tables. Each table represents an entity with a number of attributes (i.e., columns). Each row in a table is considered a record. The relational model has several forms of integrity. First, values in columns must be of specific types and, possibly, ranges. Second, the primary key used to identify a record must have a unique and legitimate value. Additionally, if record refers to an entry in another table, then that entry must exist. Otherwise, referential integrity is violated. This referential integrity allows for a single copy of data to exist in the system (part of "normalization") as well as for tables to be joined in a single result.

Relational databases adhere to a set of properties – atomicity, consistency, isolation, and durability (ACID) – to ensure that databases correctly process transactions (a logical group of a series of operations). Elmasri and Navathe [Elmasri & Navathe, 2010] define these concepts as follows:

Atomicity: a transaction is an atomic unit of processing; it should either be performed in its entirety or not at all.

Consistency: a transaction should take the database from one valid state to another. In relational DBMSs, consistency also implies that the data would be the same across all potential nodes.

Isolation: a transaction should appear as it is being executed alone even though many other transactions may be executing simultaneously.

Durability: a transaction's changes must persist (be permanent) once a transaction has been committed and not lost due to any failure.

2.2.3 Non-relational Database Management Systems

To meet performance demands, many designers of non-relational (a.k.a. "NoSQL") DBMSs deliberately chose to avoid or minimize the impact of specific functionality common to relational DBMSs within NoSQL systems. Rather than adopt ACID transactions, they follow the "basically available, soft state, eventual consistency (BASE) properties [Brewer, 2012]:

Basically available: system's availability is the primary concern.

Soft state: the system may have different versions of data on different nodes.

Eventually consistent: over time, the data will be the same version on all nodes.

The BASE properties allow the system designers to think in terms of trade-offs between consistency and latency (how fast the system responds to user requests) [Sadalage & Fowler, 2012]. For instance, by not requiring strict consistency, record inserts can scale higher in a clustered environment. Many NoSQL DBMSs also avoid processing overhead associated with encryption and access control checks [Okman et al., 2011].

NoSQL DBMSs differ from relational DBMSs in how they store data. Rather than using a group of tables, NoSQL databases typically use a more complex structure to store objects. This structure forms the unit of work when handling data. In a relational model, the data for a customer order would be placed into many tables: customer, billing method, order, order line item, shipping method, etc. Correspondingly, in a NoSQL model, the data may just be split into two primary objects: customer and order [Sadalage & Fowler, 2012]. Developers must carefully choose how to store data in NoSQL databases in a manner that reflects how the systems typically access the data. Data may be duplicated across multiple records and would need to be kept consistent by the application.

2.2.4 Database Modeling

Database modeling is the process of determining the relevant data needs for a particular application and then how to appropriately represent that data within a DBMS. Often, developers will produce three separate models: conceptual, logical, and physical [Elmasri & Navathe, 2010]. The conceptual model is a high-level data model consisting of entity types, relationships, and constraints. Typically, the logical model is derived from the conceptual model and details how the data is organized within a DBMS. The physical model builds upon the logical model contains the "internal storage structures, file organizations, indexes, access paths" for a specific DBMS [Elmasri & Navathe, 2010]. Developers can combined the logical and physical models. For our work, we extract a conceptual model from the natural language product artifacts, specifically, looking for three aspects three aspects: 1) entities – "real world" objects; 2) attributes – properties those entities have; and 3) relationships – the connections between two entities. We then map the produced conceptual model to a physical database schema to implement RBAC within the database.

2.2.5 Implementing Access Control within Databases

Most relational database management systems (RDBMS) support forms of DAC and RBAC[9]. Roles were added to the 1999 Structured Query Language (SQL) Standard [ANSI/ISO/IEC 9075-2:1999, Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation), 1999]. Similarly, the leading NoSQL DBMS [DB-Engines, 2013], MongoDB, has implemented a RBAC model [MongoDB, 2014]. For our research, the core issue is to identify which subjects and objects identified in our extraction of the access control tuples map to which roles and tables (or collections) in the DBMS. For DBMS that do not directly support RBAC, we can emulate the capabilities by ensuring user accounts who should belong to the same roles have the same permission sets established.

[9] The notable exception is MySQL which does not provide support for RBAC. http://dev.mysql.com/doc/refman/5.6/

2.3 Machine Learning

Our process extracts access control and database models from natural language product artifacts; we need extraction methods based upon known information as well as a way to discover and use new information. Combining the fields of computer science and statistics, "machine learning investigates how computer learn (or improve their performance) based upon data" [Han et al., 2011]. Machine learning based algorithms are the core of many systems involved in our lives. Such systems include credit card fraud detection, loan credit risk, medical diagnosis, optical character recognition (OCR), and e-mail spam recognition.

2.3.1 Supervised Machine Learning

Techniques and algorithms vary widely in machine learning but are generally divided into two primary categories: supervised learning and unsupervised learning. In supervised learning, people train classifiers with labeled data. People and systems then use these classifiers to decide in which class a previously unseen instance belongs. Using OCR as an example, the classifier would be trained on a set of images in which the characters have already been labeled with their actual class (specific letter, number, or punctuation). Then for testing or actual use, the classifier would use the features of how a character appears to decide upon the actual character (class) based upon those it learned previously. In contrast, unsupervised learning algorithms search data for common patterns (clusters). Data is not directly labeled, but rather groups of common instances are created. With a large sample of handwritten numbers, an unsupervised learning algorithm would ideally produce ten clusters, one for each possible number.

For our initial work, we chose to use a k-nearest neighbor classifier (k-NN), which is a supervised algorithm. k-NN classifiers work by classifying a test item based upon which items previously classified are closest to the current test item. The classifier finds the k nearest "neighbors" and returns a majority vote of those neighbors to classify the test item. A distance metric determines the closeness between two items. Euclidean distance often serves as a metric for numerical attributes. For nominal values, the distance is generally considered to be zero if both attribute values are the same or one if they differ. k-NN classifiers may use custom distance functions specific to current problem. Advantages of k-NN classifiers include ability to incrementally learn as new items are classified, to classify multiple types of data, and to handle large number of item attributes. Additionally, k-NN classifiers can output a ranked list of sentences that were most similar to the sentence being classified. This list provides visibility into how the classifier came to its decision. The primary drawback to k-NN classifiers is that if they have n items stored, classification takes $O(n)$ time. Additionally, irrelevant attributes can skew classification results. To train the classifier, we use an active learning approach [Han et al., 2011]. As new items are evaluated by the classifier, we validate the initial response, correct the response if necessary, and then place the correctly classified item into the classifier.

We also examined other classifiers. A naïve Bayes classifier works by selecting a class with the highest probability from a set of trained data sets given a specific document. Fundamentally, it assumes that each feature of a class exists independently of other features. Despite the simplification, the approach performs effectively in real-world problems. Naïve Bayes classifiers typically require fewer trained instances than other classifiers. Support vector machine (SVM) classifiers work by finding the optimal separator between two classes. As with naïve Bayes, text is represented as a word vector [Joachims, 1998]. In our work, we utilize a k-NN classifier as the primary classifier unless it does not locate any similar sentences. At that point, we use a majority vote of the three classifiers (k-NN, SVM, and naïve Bayes) to produce the classification result.

2.3.2 Unsupervised Machine Learning

With unsupervised learning, algorithms locate common attributes or classes within the data [Han et al., 2011]. We utilize a clustering algorithm, k-medoids to create groups of common sentences. These groups give us the ability to study sentences manually with common patterns. These patterns yield insights into the underlying properties consistent with sentences within a group. k-medoids works by first randomly selecting k objects C_i to function as the "center" for a possible cluster. Each of the remaining sentences is then placed into the cluster in which that sentence is closest to the center through a similarity score. Once these initial steps are complete, the algorithm finds, for each cluster, the sentence that has the lowest total distance from all other sentences within the cluster. These sentences then form a new set of "centers" and the algorithm again places each sentence into a cluster in which that sentence is closest to the center. The process repeats until the algorithm converges when no more changes occur to the set of "centers". From prior work [Slankas & Williams, 2012], we found that classification results would have high accuracy when the distance from one sentence to another was under a certain similarity ratio. Using this insight, we adapted the initial algorithm to consider the distance to existing centers prior to adding a new random center. If the distance was greater than the specified similarity ratio, we started a new cluster. Through this process, we also allowed provided options for *k* to be fixed to an initial value or to grow as needed based upon similarity scores when placing new sentences.

2.4 Natural Language Processing

Natural language processing is a diverse field of computer science including such areas as question answering, speech recognition, machine translations, and many others. We focus our research on two main areas: parsing & resolution (Section 2.4.1) and information extraction (IE) (Section 2.4.2).

2.4.1 Parsing and Resolution

NLP originates from research in the 1950s with the emergence of the artificial intelligence field. Most notably, Chomsky's [Chomsky, 1956] seminal work demonstrated NLP's difficulty. Over, 50 years later, improvements to existing solutions still need to be made for the vast majority of NLP tasks. NLP research in the 1960s and 1970s largely followed a rationalist approach with hand-built rules and grammar dominating the field [Jurafsky & Martin, 2009]. In the 1980s, researchers returned to an empirical approach for NLP through the emergence of probabilistic methods and large scale sets of annotated text [Jurafsky & Martin, 2009]. These probabilistic methods form the foundation of modern statistical parsers the represent the current state of the in NLP [Socher et al., 2013]. Modern parsers include the follow tasks: tokenization, sentence segmentation, part of speech (POS) tagging, lemmatization, named-entity recognition (NER), syntactic parsing, and coreference resolution.

Tokenization detects individual words, punctuation, and other items from the text. While tokenization can be straightforward in the English language due to spaces and punctuation marks that separate tokens, structures such as abbreviations and contractions complicate the task. Further, technical product artifacts may have structures such as Java package names[10] that generally trained tokenizers will be unable to detect token boundaries properly.

Sentence segmentation splits the tokens found in tokenization into groups.

Part of speech (POS) tagging identifies the part of speech (e.g., noun, verbs, adjectives, etc.) for each token. The Penn Treebank [Santorini, 1995] contains 36 different tags. The current state of the art achieves 97.3% accuracy for individual tokens and a “modest” 57% for accuracy for all of the tokens being correct within a given sentence [Manning, 2011].

Lemmatization generates the common root word for a group of words. For instance, sang, sung, and sings are all forms of a common lemma “sing”. A stem is the root of a word after a suffix has been stripped [Jurafsky & Martin, 2009]. Stemming algorithms are rule-drive approaches to derive base forms of words without taking into account the word’s context or part of speech. The Porter algorithm is one of the most widely used processes to create stems. In contrast, lemmatization takes into account the word’s part of speech and other information sources such as a vocabulary to derive the base form of a word [Manning et al., 2008]. The Current state of the art achieves approximately 99% accuracy for English [Gesmundo & Samardžić, 2012].

Named entity recognition (NER) classifies phrases into types named entities (e.g., people, organizations, locations, times, vehicles, events, etc.) from text [Jurafsky & Martin,

[10] For Java packages, periods form a hierarchy rather than mark the end of sentence or abbreviation.

2009]. Real-world NER architectures use a combination of high-precision rules, probabilistic matching, and machine learning techniques [Jurafsky & Martin, 2009]. The state of the art for the NER general task (based upon the CoNLL-2003 shared task) approach a F_1 score of 0.89 [CoNLL-2003, 2003].

Syntactic parsing assigns a parse-tree structure to a sentence [Jurafsky & Martin, 2009]. The tree structure provides a basis for many other tasks within NLP such as question and answer, IE, and translation. Current state of the art parsers have an F_1 score of 0.904 [Socher et al., 2013].

Coreference resolution identifies whether or not two expressions in a document refer to the same identity. For example, in the sentence "While doctors may write prescriptions, they are responsible for ensuring that the medication will not have any adverse interactions for the patient", two coreferences exist: 1) "doctors" and "they" and 2) "prescriptions" and "medications". The current state of the art has a 78% accuracy for the CoNLL-2012 Shared Task [Potts & Recasens, 2012]. Kennedy and Boguraev [Kennedy & Boguraev, 1996] introduced an algorithm to resolve pronoun anaphora resolution (finding the correct noun for a given pronoun) that does not require parsing and achieved a 75% accuracy on their test set.

2.4.2 Information Extraction (IE)

IE is an area of NLP concerned with finding structured data from text [Jurafsky & Martin, 2009]. IE differs from information retrieval (e.g. web searches) in that IE's goal is to extract information rather than provide a ranked list of relevant documents to a query. Common IE tasks include named entity recognition, reference resolution, relation extraction (RE) and event extraction. A relation typically expresses the relationship or connection between two entities, but can also be used to express the relationships among three or more entities; these entities may be any word or group of words within a sentence. Common relation types include “is-a” and “part-of”. For example, “a bicycle is a vehicle” is represented by *is_a(bicycle, vehicle)* and bicycles have two wheels is represented by *contains(bicycle, wheels)*. Similarly, we can have relation *activity(doctor, write, prescription)* to indicate that a doctor can write a prescription. The IE field has advanced largely due to the investments by the United States government through challenges sponsored by DARPA [Chinchor & Sundheim, 1996] and NIST[11]. State of the art systems for RE in these challenges typically have around 85% precision and 70% recall [Piskorski & Yangarber, 2013].

Access control policy extraction is most similar to the RE task in that the relation (action) between the subject and resource implies much of the access control policy within a statement. However, access control extraction differs from most RE tasks in that it is not constrained by small, fixed sets of binary relations [Jurafsky & Martin, 2009]. Additionally, access control extraction needs to infer the appropriate permissions based upon the identified relation (action). The permissions may be further constrained by other features within the sentences such as negativity or access limitations to just a particular person.

2.5 Evaluation Criteria

We approach our evaluation from two related dimensions: (1) the audience for the evaluation and (2) the goal of the evaluation. In the first dimension, we adopt Hirschman's groups of stakeholders consisting of researchers, funders, and users. [Hirschman, 1998].

[11] http://www.itl.nist.gov/iad/mig//tests/ace/

Hirschman stated “researchers want cheap, fast technology-centered evaluations that can be used for internal testing…evaluations that are diagnostic...and methods that show technology in a favorable light.” Similarly, funders desire evaluation that shows their investments have advanced the state of the art yet still show the utility of the technology. Users primarily “want user-centered evaluations that require testing with real users in realistic environments.” Users also desire evaluations of ease of use and cost versus performance trade-offs.

For the second dimension, we look at three different kinds of evaluation tailored to different goals: adequacy, diagnostic, and performance [Hirschman & Thompson, 1997]. Adequacy measures the appropriateness of a system for a given task. Our measurement of the complete system in generating RBAC from the natural language product artifact will be in this category. Diagnostic measures the system’s performance based upon “some taxonimization of the space of possible inputs” [Hirschman & Thompson, 1997]. Performance measures the system’s capability for a given area. These measurements are often used to see how a system’s performance changes based on different parameters, algorithms, and other methods used. Performance evaluation is also used to compare to alternatives. Precision and recall (defined in the next section) have been traditionally used in information retrieval (IR), NLP, and machine learning systems [Jurafsky & Martin, 2009][Han et al., 2011].

In addition to these two dimensions, we consider completeness, consistency, and correctness. We adopt Boehm’s definition for completeness which is a measure of “the extent that all parts are present, and each part is fully developed [Boehm, 1984]. Completeness requires that no information is left undefined as well as that there is no information is missing from the document. For completeness, we can measure the coverage of ACRs as compared to the discovered subjects and resources. We also compare the discovered resources to the defined entities within a physical database schema. We measure the whether or not we extracted all possible ACRs and DMEs from the documentation.

For consistency, we adopt the ISO/IEC/IEEE standard of "the degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component" [ISO/IEC/IEEE, 2010]. To examine consistency within documents, we look to see if there are conflicts among the ACRs defined. We also examine whether or not the same term has been used to refer to the same entity without introducing multiple synonyms. We can also examine the consistency of the ACRs extracted from the documents compared to those ACRs already existing within a DBMS.

For correctness, we adopt the ISO/IEC/IEEE standard of "the degree to which a system or component is free from faults in its specification, design, and implementation" [ISO/IEC/IEEE, 2010]. For this, we evaluate the ability of the REDE process to identify sentences containing ACRs and DMEs correctly. We also evaluate how well the REDE process extracts those ACRs and DMEs. We can also evaluate how correctly the ACRs perform when applied to an application's DBMS.

2.5.1 Classification Performance

We examined a variety of machine learning classifiers as well as different features passed to those classifiers. To compare classifier performance, we measure whether or not the classifier produces correct results. For binary (yes/no) decisions, a classifier either correctly or incorrectly predicts the decision. We term correct predictions as true positives (TP) and true negatives (TN) if the classifier correctly predicted yes or no respectively. For incorrect classifications, we term these as false positives (FP) and false negatives (FN). To evaluate binary classification, we use precision, recall, and the F_1Measure. Precision (P) is the proportion of correctly predicted statements:

$$P = TP/(TP + FP) \tag{1}$$

Recall (R) is the proportion of relevant statements found:

$$R = TP/(TP + FN) \tag{2}$$

The F_1Measure is the harmonic mean of precision and recall, giving an equal weight to both elements:

$$F_1 = 2 \times \frac{P \times R}{P + R} \tag{3}$$

In general, high recall implies that more of the applicable statements are being found in the text versus a low recall which implies applicable statements are missed. High precision implies that of those statements we believed to be applicable, a high percentage of them would actually be applicable. Similarly, low precision implies that most of the statements we thought to be applicable, were, in fact, not applicable. In the context of access control, high values for both precision and recall are desired. Lower precision implies that the process ultimately grants a role more incorrect permissions than a classification with higher precision. Lower recall implies missed access control sentences. In general, maximizing recall and precision simultaneously is not feasible [Buckland & Gey, 1994]. To boost recall, researchers will often "relax" rules or other such criteria to consider more elements; in doing so, precision scores will be lower due to more false positives. The F_1Measure compares the overall performance.

Accuracy (A) has also been used to evaluate classifiers:

$$A = (TP + TN)/(TP + TN + FP + FN) \tag{4}$$

However, accuracy can report misleading results if the class distribution is widely disparate [Han et al., 2011]. Consider the situation of credit-card fraud detection, and suppose fraudulent use only occurs in 15 of every 1,000 transactions. If a classifier simply returned "no fraud" every time, it would have an accuracy of 0.985. In contrast, both the precision and recall values would be zero as no true positives were discovered. If the classifier correctly predicted just one true positive in this situation, the precision would be 1, but the recall would be 0.067. The F_1Measure would be 0.125.

Evaluation requires dividing the labeled instances into two sets: training and test. Classifiers are trained using the training set and then evaluated with the test set. To create these sets, several approaches exist. One approach randomly divides the data into two sets under the constraint the two sets have the same classification proportion. In other situations, we can train a classifier on a set of documents, and then use a new document to evaluate the classifier. However, situations also exist in which a large number of labeled instances is not available. In these situations, we utilize a stratified n-fold cross-validation approach in which the labeled instances are randomly partitioned into *n* folds based on each fold of approximately equal size and equal response classification. For each fold, the models are trained on the remaining folds and then the contents of the fold are used to test the model. The *n* results are then averaged to produce a single result. We follow Han et al.'s recommendation [Han et al., 2011] and use 10 as the value for *n* as this produces relatively low bias and variance. The cross-validation ensures that all sentences are used for training, and each sentence is tested just once.

2.5.2 Inter-rater Agreement

As the research required a "gold standard" (i.e., the correct answers), we had two or more individuals mark sentences to what should be the correct classifications, ACRs, and database elements[12]. To evaluate how well two individuals agreed when classifying sentences and elements of those sentences categorically, we used the Cohen's kappa [Cohen, 1960], which measures the agreement between the two raters who each classified the same set of items into same group of possible categories. With multiple raters, we computed the Fleiss' kappa [Fleiss, 1971]. Fleiss' kappa measures the "degree of agreement actually attained in excess of chance" divided by the "degree of agreement that is attainable above chance". From the paper [Fleiss, 1971], the following equations are used for kappa κ:

N represents the total number of subjects,

n represents the number of raters per subject,

k represents the number of categories in which assignments are made

$i = 1, \ldots, N$ represents a particular subject

$j = 1, \ldots, k$ represents a particular category.

n_{ij} is the number of raters who assigned the ith subject to the jth category.

$$p_j = \frac{1}{Nn} \sum_{i=1}^{N} n_{ij} \tag{5}$$

p_j represents the proportion of all assignments to the jth category.

[12] For ACRs and database elements, one individual performed all of the marking. Another research then validated a random 20% sampling that contained injected mistakes. See Sections 6.2.2 and 7.2.2.

$$p_i = \frac{1}{n(n-1)} \sum_{j=1}^{k} n_{ij} \, (n_{ij} - 1) \tag{6}$$

p_j represents the extent of agreement among the nth rater for the ith subject.

$$\bar{P} = \frac{1}{N} \sum_{i=1}^{N} p_i \tag{7}$$

$\bar{P}$ represents the overall extent of agreement.

$$\bar{P_e} = \sum_{j=1}^{k} p_j^2 \tag{8}$$

$\bar{P_e}$ represents the agreement possible solely to chance. $1 - \bar{P_e}$ measures the degree attainable over what would be predicted by chance. The degree of attainment actually attained in excess of chance would be $\bar{P} \; - \bar{P_e}$. Thus,

$$\kappa \; = \frac{\bar{P} \; - \bar{P_e}}{1 - \bar{P_e}} \tag{9}$$

We implemented Fleiss' kappa to process spreadsheets used by the different raters to classify sentences. We also adopt Landis' and Koch's guidelines for agreement as present in Table 1 [Landis & Koch, 1977].

Table 1. Inter-rater Agreement Guidelines

Fleiss' Kappa	Agreement Interpretation
<= 0	Less than chance
0.01 – 0.20	Slight
0.21 – 0.40	Fair
0.41 – 0.60	Moderate
0.61 – 0.80	Substantial
0.81 – 0.99	Almost perfect

3 Related Work

This chapter reviews the work related to our process. Our research into related work began with investigating text mining through examination of several data mining and information retrieval textbooks [Manning et al., 2008; Han et al., 2011; Witten et al., 2011]. Xie et al. [Xie et al., 2009] wrote a broad survey of data and text mining techniques as applied to software engineering. From these initial overviews, we then explored published literature reviews as they related to software engineering and text processing. Our literature review then looks at specific works related to our approach's components.

The rest of this chapter is organized as follows. Section 3.1 covers literature reviews related to software engineering and text processing. Section 3.2 examines requirement classification while Section 3.3 looks at clustering requirements. Section 3.4 presents a more in-depth view of generating entity-relationship and class diagrams from textual sources. Section 3.5 discusses generating access control policies from natural language sources. Section 3.6 presents an overview of controlled natural languages, an alternate approach for converting between text and models. Section 3.7 examines sentence similarity measures as similarity applies to classification. Section 3.8 presents research related to information extraction. Section 3.9 covers literature reviews of schema and ontology mapping studies. In Section 3.10, we present a discussion of the papers surveyed in Sections 3.2, 3.3, 3.4, and 3.5.

3.1 Related Literature Reviews

Yue et al. [Yue et al., 2011] wrote a systematic literature review (SLR) [Kitchenham & Charters, 2007] on transformation approaches between user requirements and analysis models. They developed a model of a generic transformation process and created several taxonomies for analysis models, pre-processing approaches, information approaches, and requirements. From the 16 papers they studied in-depth, 12 papers derived structure model elements from text (e.g., objects). All 12 papers used rule-based approaches. Only three of 16 papers contained evaluations. From manually examining those three papers [Harmain & Gaizauskas, 2003; Diaz et al., 2005; Ambriola, 2006], we found coverage, precision, and recall metrics. These measures were computed by comparing the paper's documented approaches against those generated by humans. The materials used for evaluation came case studies extracted from textbooks [Harmain & Gaizauskas, 2003], use cases from an undocumented source [Diaz et al., 2005], and industrial pilot studies [Ambriola & Gervasi, 2006]. Yue et al. [Yue et al., 2011] documented five characteristics that approaches for transforming requirements into analysis models should possess:

1) Requirements should be easy to document using the format required by the approach;
2) Generated analysis models should be complete;
3) The approach should contain the least number of transformations required (high efficiency);
4) The approach should be automated; and
5) The approach should support traceability management.

They found that none of the approaches surveyed met this criteria.

Casamayor et al. [Casamayor et al., 2012] examined text mining to assist software design. Their work surveyed existing approaches to requirements engineering, identification of non-functional requirements, detecting aspects, traceability, source code administration, and bug tracking. Overall, Casamayor et al. listed the primary NLP and ML techniques used.

In 2012, Meth et al. wrote an SLR on requirements elicitation from unstructured text documents [Meth et al., 2013]. In this review, the authors examined 36 relevant publications divided into four categories of abstract identification, model generation, quality analysis, and identification (classification). They found 21 of the 36 papers utilized a semi-automated approach, 10 were fully automated, two utilized a combination of automated and semi-automated techniques, and three envisioned fully automatic approaches. Unfortunately for the research field, they echoed Yue et al.'s work [Yue et al., 2011] in that only 16 of the 36 publications had no evaluation concepts defined. The other publications used a combination of correctness, completes, and efficiency metrics to evaluate their studies. Meth et al. also discuss what degree of automation is appropriate. Based on the lack of precision and recall rates failing to come even close to the 100% mark, they concluded requirements engineers must be involved in the process to "check and sometimes manually rework results of the automatism".

In 2013, Borg et al. [Borg et al., 2013] published a systematic mapping study [Budgen et al., 2008; Petersen & Feldt, 2008] examining trace-recovery through information retrieval methods. With the 79 documents they analyzed, they found the vast majority of previous studies do not go beyond identifying precision and recall measures. Further, they identified a dire need of industrial case studies for evaluation use. While their study does not directly impact our work, their focus was very much related to natural language processing, machine learning, and software engineering. The key takeaway from their work was to ensure we used appropriately sized case studies in our evaluations.

3.2 Classifying Requirements

While text classification, especially with regards to term frequency, inverse document frequency (TF-IDF), has been studied for a relatively long period [Salton & McGill, 1986], NFR classification first appeared in the literature in 2007 [Cleland-Huang et al., 2007]. In their work, Cleland-Hung et al. [Cleland-Huang et al., 2007] applied TF-IDF with an additional parameter to specify the frequency of indicator terms for a NFR category as compared to the appearance of those terms in the requirement currently under test. To evaluate their scheme, they collected 15 sample requirements specifications from term projects in a Master's level class at DePaul University. They made this collection public through the PROMISE Data Repository [Menzies et al., 2012]. They reported a 0.813 recall, meaning that they successfully found 81% of the possible NFRs in the dataset. Their precision was a 0.124, indicating a large number of false positives. While not published in their work, the F_1 score was 0.239. Although they intentionally choose to maximize recall, users could be frustrated with their process due to the large numbers of false positives to examine and discard.

In 2009, Casamayor et al. [Casamayor et al., 2010] repeated Cleland-Huang et al.'s experiment, but with using a multinomial naïve Bayes classifier coupled with an Expectation Maximization algorithm to boost the classifiers performance by labeling untrained data with guesses based on the already trained data set. They achieved an accuracy of 0.97 after having classified approximately 60% of the data set.

Zhang et al. [Zhang et al., 2011b] repeated Cleland-Huang et al.'s experiment again in 2011, but utilized a SVM with a linear kernel as their classifier. They utilized a combination of different methods to represent words: n-grams (2-gram and 3 grams), stemmed-based words, and multi-word expressions. They reported significantly higher precision results, although lower recall results than Cleland-Huang et al. but did not provide the specific values for precision and recall. The three F_1 scores presented for specific NFR categories were between 0.60 and 0.65. They demonstrated that the performance of individual words was higher than models with multi-words (i.e, a 2-gram or 3-gram model) for the dataset.

Schneider et al. [Schneider et al., 2012] classified requirements as to whether or not the requirement had a security impact or not. They utilized a naïve Bayes classifier and tested three documents using a document-fold approach (i.e., they trained on two of the documents and tested on the third). The authors reported a 0.463 F_1 score. In separate work, the author of this dissertation along with three co-authors classified sentences for six security objectives utilizing a classification method similar to that in Section 5.1.2. The method produced a F_1 score of 0.54 for a document fold validation and a 0.80 F_1 score for a 10-fold cross validation [Riaz et al., 2014].

Separately, Ko et al. [Ko et al., 2007] published a work on classifying requirements into specific topics. Their semi-automatic process had analysts enter a list of topic words. Their process then bootstrapped a series of keywords for each topic word based on a similarity score when they looked at how frequently a keyword appeared with a given topic word in the same sentence. From these keywords, they collected sentences that where the “ideal” matches for given topics and then used these sentences to train a naïve Bayes classifier. Overall, they reported results of a 0.861 F_1 score.

Our work builds upon the prior work in that we utilized the same data set from Cleland-Huang et al. as part of our experiments, but we analyzed significantly more examples with multiple document types. We also examined the use of custom distances functions with a k-NN classifier. We studied the effect of different sentence characteristics on classification performance.

3.3 Clustering Requirements

Duan et al. [Duan & Cleland-Huang, 2007] used various clustering techniques (agglomerative hierarchical, k-means, and bisecting divisive) to group related sentences from requirement-based artifacts to other artifacts created during a software development process (e.g., design, code, test cases). The authors found that standard clustering algorithms produced meaningful results; they also needed to create customized techniques to determine the optimal number of clusters. To solve this, the authors created a new metric to measure the cohesion and coupling of themes within and across clusters.

Niu et al. [Niu et al., 2012] added support to label clusters of requirements automatically in their work. Ideally, these cluster labels would immediately identify the core topic for each cluster to an analyst. The authors used an agglomerative clustering technique. This technique works by placing all sentences in their own cluster and then finds the two clusters that are most similar and combines them. This process repeats until all sentences have been combined into a single cluster. The extracted clusters can then be chosen either by when the distances between all pairs of clusters are above a certain similarity ratio, or only a certain number of clusters exist. Unfortunately, the authors did not describe their distance measurement (Section 3.7). Additionally, their process generated 10 labels for each cluster. They had a human expert also provide 10 labels for each cluster. They report a Kappa statistic of 0.35, indicating a fair level of agreement.

While our process does not directly use clustering, *k*-medoids support has been built into the associated tool that was built to validate the process. Support for other cluster methods that rely on similarity scores (agglomerative and divisive) would be easily implementable. We do compute clusters to validate sentence classifications.

3.4 Model Generation

In this section, we examine related work on building models from natural language product artifacts with an emphasis on creating entity-relationship diagrams (ERDs) and class diagrams. As our process implements access control within the system's environment – specifically, the database - creating an entity-relationship model is critical to our process. While class diagrams also contain behaviors, they contain entities, associated state (attributes) and have relationships. Additionally, class diagram generation appears more frequently than ERD generation in recent literature. Both ERDs and class diagrams use equivalent techniques (primarily heuristics and rule-driven approaches).

Different approaches exist to discover database entities from natural language. In 1983, Chen [Chen, 1983] published a series of 11 heuristics to create entity relationship diagrams from natural language. Chen did not publish any formal evaluation of these rules, relying only on a small, worked example. Hartman and Link [Hartmann & Link, 2007] extended this work in 2007 through additional heuristics and refinements, primary to support extended ERDs. Hartman and Link show how their heuristics relate back Chen's original set of heuristics. As with Chen, they do not perform any formal evaluation, but do work through a small sample of 12 sentences. Both approaches require humans to analyze the text.

Omar [Omar, 2004] also looked a heuristic based approach in her PhD thesis in which she documented 29 heuristics, including 15 heuristics from other researches. She evaluated her work through 11 examples used as training sets and 30 examples used as test data sets. Each dataset is 4 to 8 lines in length and was created by Omar. For performance, the system had a 0.90 recall and a 0.85 precision. In 2008, Omar [Omar et al., 2008] extend her work in looking at semantic analysis to further improve automated database model extraction. The authors identified 9 semantic roles that "may be helpful interpreting possible elements of the ER model". They then documented heuristics to determine two of the roles and how those roles may then aid in entity detection. No formal evaluation was presented.

Du [Du, 2008] also looked at NLP techniques for automatically generating data models. Du used a combination of heuristics and typed dependency sentence representations to generate an initial data model that analysts could then further refine. Du relied upon a user-based study evaluate this human-based approach on 20 problems consisting of 10 to 15 statements each. The human test subjects were two undergraduate students and eight graduate students in the School of Information Sciences at the University of Pittsburgh. Du showed statistical significance ($p = 0.0014$) to support the hypothesis that students utilizing the approach produced better results in less time ($p = 0.0049$).

Elbendak et al. [Elbendak et al., 2011] generated class diagrams from use cases following a heuristic based approach. They evaluated their process and associated tool on 6 case studies that ranged in size from 100 to 550 words in length. The authors reported and average recall of 0.90 and an average precision of 0.85. They compared their results to "nine independent software engineers who between 5 and 20 years of experience in UML". In their user-based evaluation, they found that the test participants averaged a 0.58 recall and a 0.82 for precision. The authors extensively documented their rules and process.

Sagar and Abirami [Vidya Sagar & Abirami, 2014] generated class diagrams from use cases based upon heuristics, but utilized the Stanford NLP rather than a member-based shallow parser used by Elbendak et al. [Elbendak et al., 2011]. Their heuristics were well-defined, and they worked through an ATM problem statement [Rumbaugh et al., 1991] as a case study. As with the prior work, they used recall, precision, and over-specification[13] as evaluation metrics. They evaluated three case studies, two of which Elbendank et al.'s also evaluated. Sagar and Abirami showed significant improvement on one of the two test cases. On the other test case, both systems had a perfect recall (1.00) and 0.91 precision. Sagar and Abirami's work was clearly and explicitly motivated by Elbendak et al.'s work. Sagar and Abirami succeeded in fully automating the process and improving performance.

Our process utilizes heuristics to bootstrap a machine learning based approach to extract database models with the intention that the process can learn rules we process additional sentences. As with Du, we utilize typed dependencies from the Stanford NLP. We report how frequently patterns appear within the texts that infer elements of the database or class diagrams (to our knowledge, no existing work has shown any such distribution).

[13] Over-specification is a score as to how much extra information was generated by a process that was not part of the recognized solution. The score is computed by dividing the number of extra elements of information generated by the total number of information elements in the recognized solution.

3.5 Natural Language and Access Control

Other researchers have explored using natural language sources to generate access control policies. Fernandez and Hawkins [Fernandez et al., 1997] first presented a basic overview of extracting RBAC from use cases in 1997. While they discussed how to model ACRs for RBAC, they did not present any heuristics or other guidelines to extract such ACRs. Similarly, they did not have any evaluation present in their paper.

He and Antón [He & Antón, 2009] proposed an approach to generate access control policies from natural language based upon available project documents, physical database implementation, and existing policies. Utilizing a series of heuristics, humans would analyze the documents to find access control policies. In addition to heuristics to find the elements of the typical access control tuple (subject, resource, action), they establish heuristics to identify policy constraints (temporal, location, relationship, privacy, etc.) and obligations. He and Antón validated their process with four cases studies and provided the number of ACRs created, the number of inconsistencies found, the number of modified records, and other counts to show the effectiveness of their process. They then evaluated their process with a user study utilizing graduate and undergraduate students enrolled in software engineering classes. They found their process generated significantly more ACRs than when the students did not use a process to write ACRs.

More recently, Xiao et al. [Xiao et al., 2012] defined an approach, Text2Policy, where they parse use cases to create eXtensible Access Control Markup Language[14] (XACML)

[14] http://www.oasis-open.org/committees/xacml

policies. Their approach required use cases and the ability to parse sentences into one of four specific sentence patterns to extract the subject, action, and resource for an access control policy. Using XACML as a target simplified their process as well as in that they did not have to map objects in the natural language to existing application elements or database entities. We present a more detailed comparison with Text2Policy in Section 6.4.2

Rather than using a completely rule-based or heuristic-based approach, our process utilizes machine learning to make a decision if a sentence involves access control and then from discovered semantic relation patterns, extract the access control policy. We then implement those policies within a system's database(s).

3.6 Controlled Natural Language Approaches

Other researchers resolved converting natural language to and from policies through a controlled natural language (CNL). Schwitter [Schwitter, 2010] defined a CNL as "an engineered subset of a natural language whose grammar and vocabulary have been restricted in a systematic way in order to reduce both ambiguity and complexity of full natural language." While a CNL provides consistent, semantic interpretations, a CNL limits authors to the defined grammar and typically require language-specific tools to stay within the language constraints. Project documents previously created cannot be used as inputs without processing the documents manually into the corresponding tool(s) for the CNL. Policy authored outside of tools must confirm to strictly limited grammars to be automatically parsed. Brodie et al. [Brodie et al., 2006] used this approach in the SPARCLE Policy Workbench. By using their natural language parser and a controlled grammar, they effectively translated from natural language into formal policy. Inglesant et al. [Inglesant et al., 2008] demonstrated similar success with their tool, PERMIS, which utilized an RBAC authorization model. However, they reported issues with users not comprehending the predefined "building blocks" imposed by using a CNL. Recently, Shi and Chadwick [Shi & Chadwick, 2011] presented their results of an application to author access control policies using a CNL. They showed the improved performance of their CNL interface through more complete and correct policies. A user survey indicated a strong preference for their interface versus the prior system. Our process removes the CNL constraint, working against original, unconstrained texts.

3.7 Sentence Similarity and Classification Features

While similarity among words and texts has been researched for many years, sentence similarity has just become an active research area in recent years [Park et al., 2000; Li et al., 2006]. Li et al. listed three reasons why existing techniques for documents were not sufficient:

1) The difference in the number of possible words compared to the number of words in just a sentence is substantial. This difference creates problems when using a word vector representation[15] of a sentence's words. A higher percentage of the vector entries are blank than compared to those of a large document. In turn, this makes it more difficult to find similarity among sentences - i.e., "the curse of high dimensionality" [Bellman, 1957].

[15] In a word vector representation, each entry in the vector is indexed by a particular word and typically continues the number of times that word appears in the document or sentence.

2) IR based systems usually eliminate specific stop words[16]. These words may have a structural context that can change a specific sentence's meaning.
3) Sentences may not share the same words albeit they have the same semantic meaning.

Park et al. [Park et al., 2000] examined requirements similarity through lexical affinity, co-occurrences of two words with the two words being at most five words away from each other in a sentence. Those pairs then formed the "features" for a given sentence. To measure the distance between two sentences, they computed the cosine of the two corresponding sets of pairs. They determined ambiguous sentences solely by the presence of a word from a list of words that indicated ambiguity. For similarity, they measured precision as the number of correct answers in the given rank divided by the number of sub-documents. They authors scored 0.727 for their method. For ambiguity, they used precision and recall as defined in Section 2.5.1. Here, the authors had 1.00 recall, with a 0.47 precision. The author's dataset consisted of 33 documents with 1090 sentences total "from real fields". The test and training sets were created by using odd-numbered sentences for training and even-numbered sentences for test. We did not find further use of their similarity technique within our literature search. We did use the same method for detecting ambiguity within our process.

[16] "Stop words" are sets of common words that are typically removed from documents in information retrieval systems as their commonality reduces their ability to help differentiate among texts. Common examples include determiners ("a", "an", "the") and prepositions.

Li et al. [Li et al., 2006] created their semantic similarity measurement based on WordNet[17] and whether or not words were in the same synset[18] or if there was overlap among the two word's synsets. They compared the performance of their method to the results of a set of participants who computed a score for sentence pairs. They then used the Pearson correlation coefficient to compare their results with the participant scores. They reported a correlation result of 0.816, significant at the 0.01 level.

Zhang et al. [Zhang et al., 2011a] discuss three categories of sentence similarity measures: symbolic, semantic, and structural. Symbolic similarity examines whether the words are the same between two sentences. Semantic similarity measures how closely two sentences convey the same meaning. Structural similarity measures the relations between words and the placement of words in a sentence. Zhang et al. calculate six statistical methods that are symbolic or structural to compute similarity. These methods were based on word sets (Jaccard, Dice), word vectors (cosine), edit distance (Levenshtein), word order, and word distance. They found that word set produced the best result – with a F_1 measure of 1.00. The authors test set was 40 sentences selected from the NIST05 corpus for machine translation. These 40 sentences were divided into 10 groups with four sentences to each group. The assigned sentences with a group all had similar meaning.

Falessi et al. [Falessi et al., 2010] presented a paper comparing the performance of multiple NLP techniques across the categories of algebraic models, similarity functions, term

[17] http://wordnet.princeton.edu/
[18] A "synset" is a group of related words with the same semantic meaning.

weighting, and term extraction. They applied a method from each of the categories and then ran that method to compute their evaluation metrics. The authors found using a combination of vector space models, cosine-based similarity, raw frequency counts, and extracting only verbs and nouns produced the best result. With this combination, they had recorded a 0.935 precision and a 0.936 recall for detecting pairs of similar sentences that were extracted from a variety of requirement documents from a large European aerospace firm.

In contrast, Oliva et al. [Oliva et al., 2011] present a method combining semantic and structural information. They computed semantic distances by measuring distances between words in WordNet and structural distances through looking at common phrases among sentences. Using Pearson's correlation, they had a 0.76 correlation of agreement to two human annotators.

Our work relies on finding the appropriate distance metric to use between two sentences. We explored the various methods presented in these papers as well as studied other possibilities. We also examined the use of various sets of stop words in regards to classifier performance.

3.8 Information Extraction

A semantic relation expresses the relationship between two entities. Common semantic relation types include "is a" and "part of." For example, "a bicycle is a vehicle" is represented by *is_a(bicycle, vehicle)* and bicycles have two wheels is represented by *contains(bicycle, wheels)*. More complex relations include location and roles. The sentence "Barack Obama is President of the United States of America" has the semantic relation *leads(Barack Obama, United States of America)*. While an exhaustive taxonomy of all semantic relations is infeasible, Storey presented an initial taxonomy for use in database models [Storey, 1993]. To extract ACR elements and DMEs from the natural language product artifact, we utilize semantic relation extraction. A number of different ways exist to identify semantic relations within text. Initial solutions employed hand-written patterns to detect hyponym ("is-a") relationships among words [Hearst, 1992]. While hand-written patterns usually have high precision, they tend to suffer low recall by missing semantic relation occurrences. Snow et al. [Snow et al., 2004] used dependency paths and grammatical relationships within a sentence to discover additional relationship patterns. Akbik and Broß [Akbik & Broß, 2009] used a similar process to extract a diverse range of semantic relations with the goal to extract arbitrary relations for semantic search. Zhou et al. [Zhou et al., 2005] documented a variety of possible features to use in relation classification.

In 2009, Mu et al. [Mu et al., 2009] proposed an "extended functional requirements framework" that contained 10 semantic properties[19] for each requirement. To discover these properties, the authors defined rules for each semantic property type that could be matched to the output of an NLP parser. When any found match, they automatically extracted the matching words into a property for their framework. Their test document was an auto-marker system that followed the IEEE-STD-830 standard with 249 sentences. Their average F_1 score was 0.625 across all ten categories. Looking at just subject, predicate, and object extractions, they had an average F_1 score of 0.91. Our work is similar in that we utilize patterns from NLP

[19] agentive, action, objective, agent feature, object feature, locational, temporal, manner, goal, and constraint

parsing output to extract information. However, we use an iterative learning process to learn to patterns from an initial set of patterns used to bootstrap the process.

We utilize word patterns and grammatical relationships to extract the access control policy elements from sentences as well as to extract the database model elements. We incorporate machine learning algorithms so the process can learn and apply new patterns as more and more natural language text has been evaluated. We also studied which features to use for relation extraction in terms of access control elements and database model elements.

3.9 Schema and Ontology Mapping

As the database model extracted from natural language product artifacts most likely has differences in both name and structure from the actual database schema, a process must exist to map from the extracted design to the actual schema. This problem is similar to the challenges of matching one database schema to another database schema as well as matching different ontologies. Multiple survey papers have been written on database schema matching [Batini et al., 1986; Rahm & Bernstein, 2001; Doan & Halevy, 2005] and ontology matching [Kalfoglou & Schorlemmer, 2003; Choi et al., 2006]. Research continues in this field with such recent work as Po and Sorrentino [Po & Sorrentino, 2011] who utilized probabilistic techniques to derive lexical relationships and disambiguate word sense across different ontologies. Our initial approach computed word similarity and relied upon the user to decide if the mapping was correct.

3.10 Discussion

Table 2 presents a summary of the selected works from this chapter as well as the relevant studies that form the body of this dissertation. Excluding our work, 11 of the 23 cited papers used some form of precision and recall to measure correctness and completeness. 9 of the 23 papers used some form of student or author based materials for evaluation; this highlights a strong need for documents from industry or other organizations. 14 papers used some form of requirements text. However, such texts may not be available either due to the project members not writing the documents or the software development process used alternate means to create shared understanding between developers and stakeholders as to what needed to be created. In five of the seven papers for model generation (Elbendak et., 2011 and Vidya Sagar & Abirami, 2014 were the two exceptions), the authors utilized documents in which non-relevant sentences were already removed from the input. Such removal greatly simplifies the analysis as such the process does not have to determine the relevancy of a sentence. This removal of non-relevant sentences also increases precision scores as rule-based approaches would not generate as many false positives. Eight of the 23 papers made their test documents available. All test cases and source code for this research have been made openly available[20].

[20] https://github.com/RealsearchGroup/NFRLocator
https://sites.google.com/site/accesscontrolruleextraction/

Table 2. Selected Related Work Summary

Paper / Authors	Technique	Automation	Evaluation Metrics	Input Types	Test Documents
Classifying Requirements					
[Cleland-Huang et al., 2007]	TF-IDF with customized probability score	Full	Precision, Recall, Specificity	Requirements	15 Student-based projects 625 Requirement statements
[Casamayor et al., 2012]	Naïve Bayes with Expectation Maximization	Full	Precision, Recall, Accuracy, F-Measure	Requirements	15 Student-based projects 625 Requirement statements
[Zhang et al., 2011b]	Support Vector Machines	Full	Precision, Recall, Accuracy, F-Measure	Requirements	15 Student-based projects 625 Requirement statements
[Schneider et al., 2012]	Naïve Bayes	Full	Precision, Recall, F-Measure	Requirements	3 real-world systems, 510 sentences, document fold validation
[Ko et al., 2007]	Bootstrapping keywords from topics, Naïve Bayes	Semi	Precision, Recall, F-Measure, Effort	Requirements	Undocumented 389 requirement sentences
Clustering Requirements					
[Duan & Cleland-Huang, 2007]	Multiple clustering techniques	Full	Hubert Index, CC-Ratio, Theme Cohesion and Coupling	Requirements, design elements, source code	3 systems medium (470 requirements total)
[Niu et al., 2012]	Agglomorative Clustering Chi-Square Selection	Full	Cohen's Kappa	Requirements	3 system requirement documents large
Model Generation					
[Chen, 1983]	Heuristics	Manual	None	Textual Descriptions	Small sample test case
[Hartmann & Link, 2007]	Heuristics	Manual	None	Textual Descriptions	Small sample test case
[Omar, 2004]	Heuristics	Semi	Precision, Recall	Textual Descriptions	41 small sample test cases
[Omar et al., 2008]	Heuristics with semantic roles	N/A	None	Textual Descriptions	None
[Du, 2008]	Heuristics, Tool Supported Process	Semi	User study	Textual Descriptions	20 small sample test cases

Table 2. Continued

Paper / Authors	Technique	Automation	Evaluation Metrics	Input Types	Test Documents
[Elbendak et al., 2011]	Heuristics, Tool Supported Process	Semi	Precision, Recall, Over-specificity, User study	Use cases	6 sample test cases (small) 100 to 550 words
[Vidya Sagar & Abirami, 2014]	Heuristics, Tool Supported Process	Semi	Precision, Recall, Over-specificity, User study	Use cases	3 sample test cases (small to medium)
Natural Language and Access Control					
[Fernandez et al., 1997]	Heuristics	N/A	None	Use cases	
[He & Antón, 2009]	Heuristics	Manual	User study	Requirements	4 cases studies
[Xiao et al., 2012]	Various ML and NL algorithms	Full	Precision, Recall	Use cases	iTrust, collect ACP statements, proprietary IBM application
Sentence Similarity					
[Park et al., 2000]	Lexical Affinity and word matching	Full	Precision, Recall	Requirements	33 requirements document with 1090 sentences
[Zhang et al., 2011a]	Various	Full	Precision, Recall	sentences	NIST05 BLEU small (40 sentences)
[Falessi et al., 2010]	Various	Full	Precision, Recall, ROC, Lag, Credibility	requirement text	large industrial large
[Oliva et al., 2011]	Semantic and structural Similarity	Full	Pearson Correlation, Precision, Recall, Accuracy	Text corpus	Microsoft Paraphrase Corpus, Li's unknown
[Li et al., 2006]	Semantic Similarity	Full	Pearson Correlation against human ratings	sentences	Custom unknown
Information Extraction					
[Mu et al., 2009]	Information extraction	Full	Precision, Recall	Requirements	Auto-marker system with 249 sentences
Dissertion Related Studies					
[Slankas & Williams, 2013a]	Various ML Algorithms	Full	Precision, Recall, F-Measure, Effort	Various	12 documents with 11,876 sentences
[Slankas & Williams, 2013b]	Various ML Algorithms	Semi		Use cases	iTrust
[Slankas et al., 2014]	Various ML Algorithms	Semi		Use cases	iTrust, CyberChair, IBM Course Management
[Slankas & Williams, 2015]	Various ML Algorithms	Semi		Training manual	Open Conference Systems

4 Role Extraction and Database Enforcement

Role Extraction and Database Enforcement (REDE), is a six-step process to extract roles, database objects, and other security elements from natural language product artifact and generate database-enforced access control policies for systems that utilize relational databases. An open-source tool exists to support users following the process.

For input, the process takes natural language product artifact(s), a physical database schema, existing database role names, and, optionally, a list of domain-specific words. Any number of existing project artifacts such as requirements, use cases, designs, test scripts, and training material can serve as the natural language product artifact. The physical database schema consists of the existing table definitions. Database designs may be substituted if the actual implementation is unavailable. The process outputs the SQL commands to establish role-based access control in a database. Additionally, the process can generate consistency and traceability reports. Figure 2 shows an overview of the process.

The REDE process consists of six primary steps:

1. *Parse unconstrained natural language product artifacts*. In this step, we read the entire artifact into the tool built to support the process. The tool then parses each sentence with the Stanford Natural Language Parser and outputs a graph in the Stanford Type Dependency Representation (STDR) [de Marneffe et al., 2006]. From the STDR generated by the parser, we create our sentence representation (SR) to track additional attributes for the sentence and each word.

2. *Classify sentence*. Next, we use a classifier to examine whether the sentence infers access control. We also classify the sentence as to whether or not it infers part of the database model.
3. *Extract access control elements*. The process finds subjects, actions, and objects based upon learned semantic relation patterns.
4. *Extract database model elements.* Similar to the previous step, this step uses semantic relation patterns to extracts database entities, attributes, and relationships.
5. *Map data model to physical database schema.* This step maps the data model extracted from the previous step to the actual schema used in the system's database(s). Ambiguous subjects and objects are resolved into their actual entities as well.
6. *Implement Role-based Access Control*. Based on the extracted access control elements, the system generates the necessary commands to generate role-based access control within a relational database

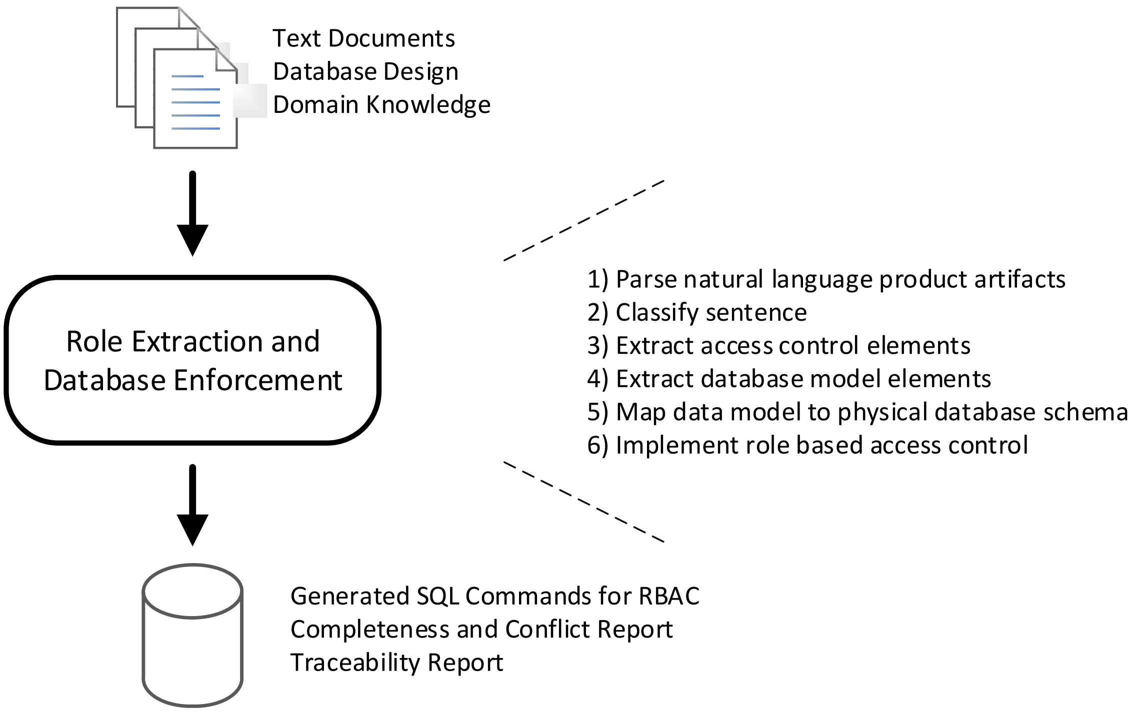


Figure 2. Role Extraction and Database Enforcement (REDE) Process

The process produces ancillary benefits as well. Access control policies are traceable back to their document sources. Policy conflicts in which a subject has contradictory permissions can be detected within the documentation. Current access control policies can be validated against those produced by the process. Authors can be made aware of ambiguity present in their documents.

In this chapter, we first describe models used by the process to represent sentences, ACRs, and the data model in Section 4.1. Section 4.2 covers the process in-depth. Section 4.3 covers additional challenges and corresponding solutions.

4.1 Representation Models

REDE uses three models to represent information about sentences, ACRs, and database model elements.

4.1.1 Sentence Representation (SR)

Internally, REDE represents sentences with a dependency graph [de Marneffe et al., 2006] as depicted in Figure 3 for the sentence “a nurse can order a lab procedure for a patient.” In Section 4.2.1, we discuss how REDE produces these graphs. Each vertex represents a word from the sentence along with the word’s part of speech. In Figure 3, “NN” represents a noun, “VB” represents a verb, and “MD” represents a modal verb. Edges represent the grammatical relationship between two words. For instance, “nurse” functions as the nominal subject (nsubj) for “order”, and “lab procedure” is the direct object (dobj) to be ordered. The indicators correspond to the subject (“S”), action (“A”), resource (“R”) typically defined within an ACR. Dependency graphs can be considered as trees in most situations and are typically rooted by the sentence’s main verb. When conjunctions are present, vertices may have multiple parents, and thus the structure needs to be treated as a graph.

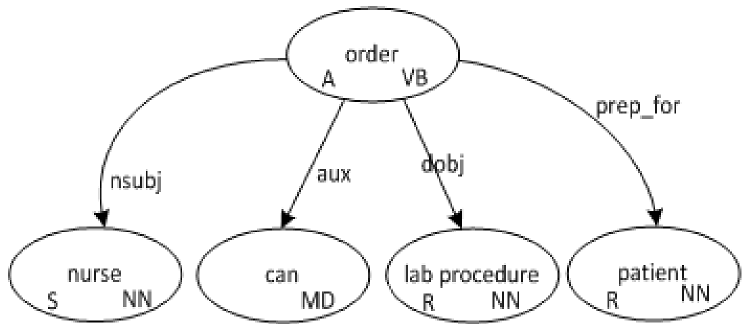


Figure 3. REDE Sentence Representation

4.1.2 Access Control Rule (ACR) Representation

To represent an ACR, we use the structure presented in Figure 4**.** We limit permissions to have the values of "create", "retrieve", "update", and "delete" as we are primarily concerned with controlling the ability to view and manipulate data within a DBMS. We do use "execute" for permissions that do not map to one or more of the four preceding permissions.

$$A(\{s\},\{a\},\{r\},[n],[l],\{c\},H,p)$$

Figure 4. ACR Representation

- A defines the overall ACR;
- s contains an ordered set of vertices that comprise the subject of a rule;
- a represents the action;
- r represents the resource;
- n contains the vertex representing negativity if required for the rule;
- l contains the indicating vertex if the rule should be limited to a particular subject s;

- c contains additional vertices required to provide context to a given action for a set of permissions;
- H represents the subgraph of a sentence's dependency graph that contains the vertices and necessary edges to connect all of the vertices listed in s, a, r, n, l, c; and
- p represents the permissions typically associated with an action.

From the example in Figure 3, we define the two rules in Figure 5.

$$A((nurse), (order), (lab\ procedure), (\ ), (\ ), (V\!: nurse, order, lab\ procedure;$$
$$E\!: (order, nurse, nsubj); (order, lab\ procedure, dobj)\ ), create)$$
$$A((nurse), (order), (patient), (\ ), (\ ), (V\!: nurse, order, patient;$$
$$E\!: (order, nurse, nsubj); (order, patient, prep\_for)\ ), read)$$

Figure 5. Extracted ACRs

4.1.3 Database Model Representation

To represent the various elements with database model representation, we use three separate models to track entities, attributes of entities, and relationships among entities. For entities, we track the subgraph of SRs (from Section 4.1.1) from the root vertex to the entity with the words in the vertex(s) above the entity vertex wildcarded. From the sentence in Figure 3, we would produce the entity graphs as displayed in Figure 6. "E" indicates that the vertex represents any entity vertex. The "%" is a wildcard – any word can be substituted as long as the part of speech matches.

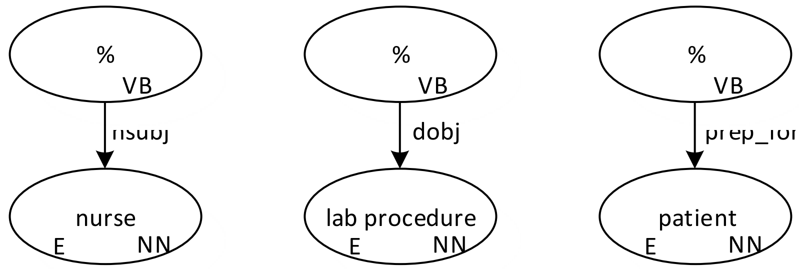


Figure 6. Entity Graphs

We formally define this in Figure 7.

$$D_e(\{e\}, H)$$

Figure 7. Entity Graph Representation

- D_e defines the overall database element for an entity;
- e contains an ordered set of vertices that comprise the entity; and
- H represents the subgraph of a sentence's dependency graph that contains the vertices and necessary edges to connect all of the vertices listed in e to the root vertex.

To represent entity-attributes, we track the subgraph containing the necessary vertices and edges as displayed in Figure 8, which contains the entity-attribute of lab procedure – pH value from the sentence "The technician records the lab procedure's pH value". The "A" indicates the vertex containing the attribute and the "E" indicates the corresponding entity.

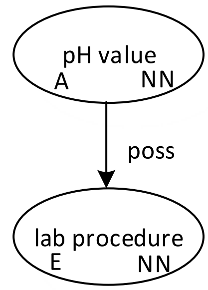


Figure 8. Entity-attribute Graph

We formally define this in Figure 9.

$$D_a(\{e\}, \{a\}, H)$$

Figure 9. Entity-attribute Graph Representation

- D_a defines the overall database element for a entity-attribute;
- e contains an ordered set of vertices that comprise the entity;
- a contains an order set of vertices that comprise the attribute;
- H represents the subgraph of a sentence's dependency graph that contains the vertices and necessary edges to connect all of the vertices listed in e and a.

To represent relationships, we track the subgraph containing the necessary vertices and edges as displayed in Figure 10. The "R" indicates the vertex that contains the identifying relationship and the "E" indicates a vertex that belongs to the relationship. The ordering of entities is immaterial.

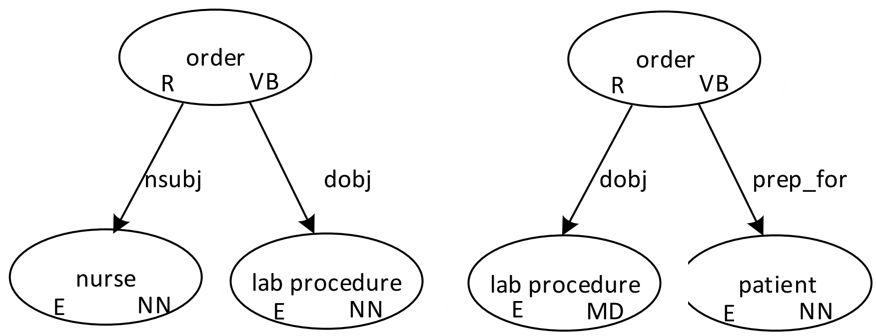


Figure 10. Relationship Graphs

We formally define this in Figure 11.

$$D_r(\{r\}, \{e_1\}, \{e_2\}, H)$$

Figure 11. Relationship Graph Representation

- D_r defines the overall database element for a relationship;
- r contains an ordered set of vertices that comprise the entity;
- e_1 contains an order set of vertices that comprise the first entity;
- e_2 contains an order set of vertices that comprise the second entity; and
- H represents the subgraph of a sentence's dependency graph that contains the vertices and necessary edges to connect all of the vertices listed in r, e_1 and e_2.

4.2 Role Extraction and Database Enforcement (REDE) Process

This section details REDE – our approach to allow organizations to utilize existing, unconstrained natural language product artifacts to generate RBAC policies for databases. To assist users in following this process, we have implemented an open-source tool. REDE combines NLP, information extraction (IE), and machine learning (ML) techniques to identify appropriate sentence, extract ACRs, and extract DMEs. To demonstrate REDE, we utilize a fragment of a use case for a professor entering grades within a course management system (see Figure 12).

> Use Case: Enter Student Grades
> The system displays a list of courses taught by the professor. The professor selects a course. The system displays a list of enrolled students for the course. The instructor can enter a grade for each graduate student.

Figure 12. Sample Product Artifact

4.2.1 Step 1: Parse natural language product artifacts

We first read the entire text into the tool built to support the approach. We separate the input into lines by either a carriage return or by periods at the end of sentences. Next, we apply a concise document grammar (Figure 13) to label each line to a specific type:

- *title*: Lines which follow capitalization rules for titles. We separate them from other lines in our process because titles rarely indicate an access control related requirement.
- *list start*: These lines represent the header or description of a list that follows.
- *list element*: These lines represent individual items contained within an ordered or unordered list. These lines are combined with the start of the list when sent to the parser and for classification. Combining the two provides additional context to both human analysts and machine classifiers.

- *normal sentence*: These lines represent statements that are not considered titles, list starts, or list elements.

document	→	*Line*
Line	→	*listID* title *line* | title *line* | *sentence line* | λ
sentence	→	normalSentence | listStart (":" | "-") *listElement*
listElement	→	*listID sentence listElement* | λ
listID	→	*listParanID* | *listDotID* | number
listParanID	→	"(" *id* ")" *listParanID* | *id* ")" *listParanID* | λ
listDotID	→	*id* "." *listDotID* | λ
Id	→	letter | romanNumeral | number

Figure 13. Document Grammar

Further, we identify heading and list identifiers (e.g., "4.1.1" and "•"). The process removes these identifiers from the sentence passed into the natural language parser later in this step as the identifiers created issues with the generated output from the parser. If any irregularities are found, the parser defaults the current token (sentence) to "normal sentence" to recover.

Within Figure 13, italicized words represent nonterminal symbols that can be replaced by other symbols on the right-hand side. Words in normal font are terminal symbols. Characters within quotation marks are also specific terminal symbols. **λ** represents an empty expansion of a nonterminal.

By running the process on the sentences within the sample text document (Figure 12), we produce the output in Table 3.

Table 3. Sample Sentences

ID	Label	Text
S1	Title	Use Case: Enter Student Grades
S2	Normal	The system displays a list of courses taught by the professor.
S3	Normal	The professor selects a course.
S4	Normal	The system displays a list of enrolled students for the course.
S5	Normal	The instructor can enter a grade for each graduate student.

In our approach, after identifying the different sentence types, we parse each line (sentence) using the Stanford Natural Language Parser[21] and output a graph in the Stanford Type Dependency Representation (STDR) [de Marneffe et al., 2006]. While the Stanford Parser has several output formats available, we choose the STDR because it incorporates the sentence's syntactic information in a concise and usable format and captures the grammatical relationship between words. Dependency parse trees have become a critical component for many semantic role labeling systems as the NLP community evolved in the early 2000s from a strict constituent-based (i.e., shallow parsing) systems [Surdeanu & Johansson, 2008]. From the STDR generated by the parser, we create the sentence representation (SR) since we need to track additional attributes for the sentence and each word.

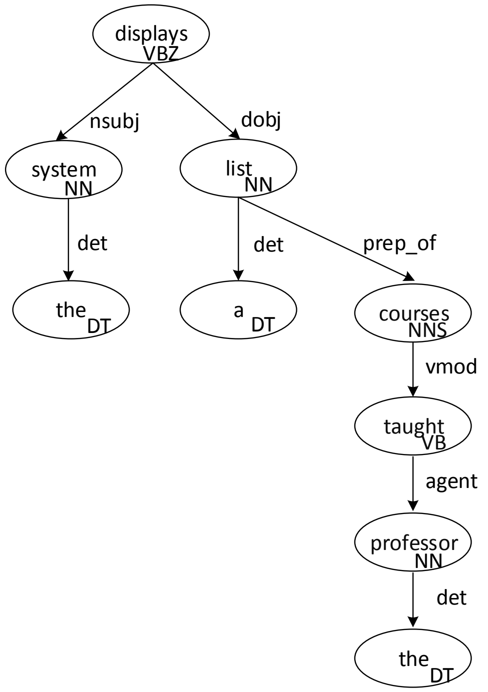
As our process depends considerably on the correctness of the parser output, we established two mechanisms to correct possible errors. First, to replace shorthand or remove text that the parsing would not recognize, the process applies a series of regular expressions to the text. Specifically in our work, we use this mechanism to replace 'w/' with 'with' and '/'

[21] http://www-nlp.stanford.edu/software/corenlp.shtml

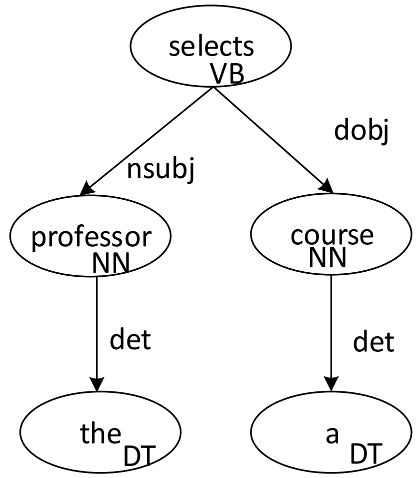
with ‘or.’ Additionally, as the Stanford Parser processes sentences, it tags each word with a part of speech. Due to differences in text[22] used to train the parser versus text used by our process, the parts of speech may be incorrect. To overcome this issue, we inserted a custom method into the parsing pipeline to override the part of speech tags if they are incorrect. For instance, we discovered that the parser always tagged “displays” as a plural noun whereas in most sentences in our text “displays” is a verb. The custom method looks for specific patterns among a group of words and then replaces the incorrect part of speech. Incorrect tags can lead to an incorrect dependency graph being generated. In turn, an incorrect dependency graph can limit the effectiveness of the REDE process. Both overrides are configurations established when the tool starts and applied without user intervention to the entire text.

Figure 14 displays the STDR for the “normal” sentences from our sample text (Figure 12). Each vertex contains a word from the sentence along with that word’s part of speech. In the figure, “DT” represents a determiner, “MD” a modal verb, “NN” a noun, “NNS” a plural noun, “VB” indicates a verb, “VBN” a past-participle verb. Edges represent the grammatical relationship between two words. For instance, “instructor” functions as the nominal subject (nsubj) for “enter” and “grade” is the object to be entered.

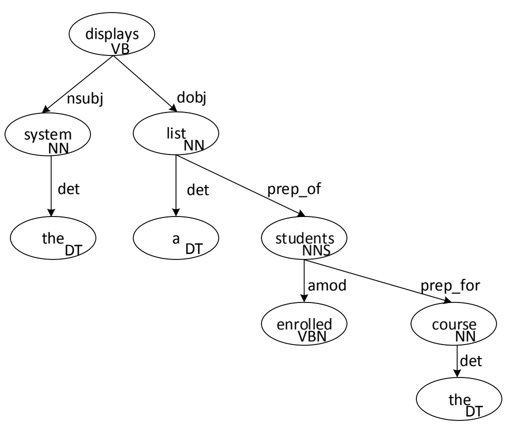
[22] The Standard Parser is trained on news articles from the Wall Street Journal in the 1980s. [Stanford Natural Language Processing Group, 2015]



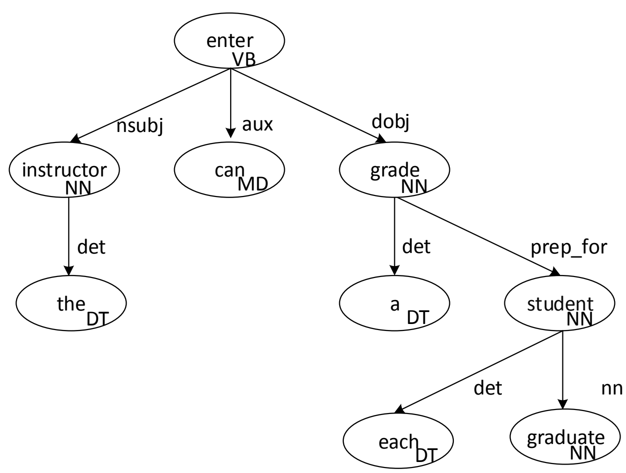
S2: The system displays a list of courses taught by the professor.



S3: The professor selects a course.



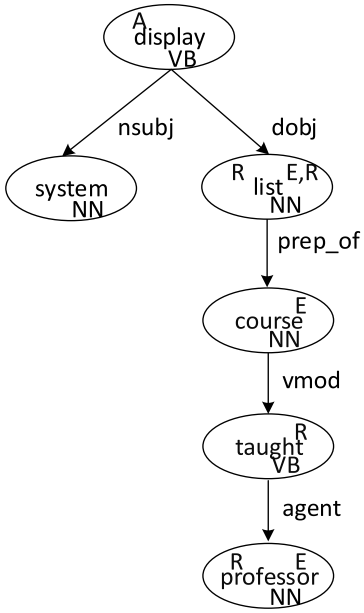
S4: The system displays a list of enrolled students for the course.



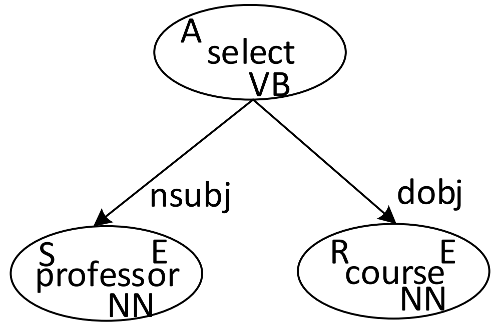
S5: The instructor can enter a grade for each graduate student.

Figure 14. Type Dependency Graphs for the Sample Sentences

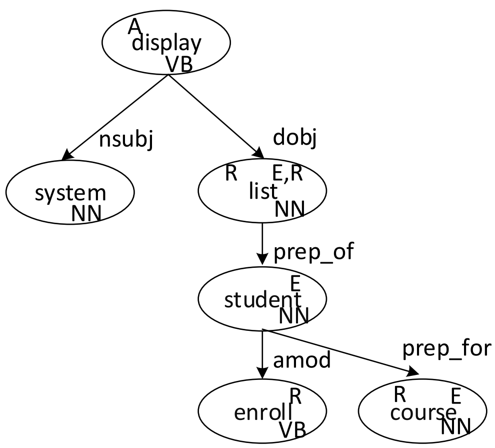
From the STDR generated by the parser, we create our sentence representation (SR) as REDE needs to track additional attributes for the sentence and each word. Additionally, some words in the original sentence are not required for our purposes and, hence, are removed from the SR. Figure 15 shows our corresponding SR for the same sentences as in Figure 14. The primary differences between the two graphs are the number of vertices and how each word is represented within a vertex. Within our SR, vertices correspond to words in the sentence and contain the word, the word's lemma, part of speech, domain flag, and access control policy indicators. The indicators in each vertex in the upper left hand corner correspond to the subject("S"), action("A"), resource("R") typically defined within an access control tuple. The indicators in each vertex in the upper right-hand corner correspond to aspects of the data model. "E" marks an entity, "EA" marks an entity's attribute, and "R" marks a relationship. The process assigns these indicators in steps three and four.



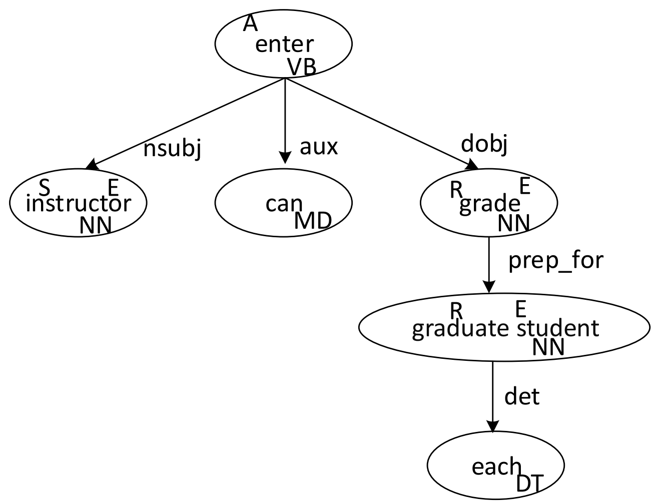
S2: The system displays a list of courses taught by the professor.



S3: The professor selects a course.



S4: The system displays a list of enrolled students for the course.



S5: The instructor can enter a grade for each student.

Figure 15. Sentence Representation (SR) Graphs for the Sample Sentences

Using a pre-order traversal, the process creates the SR from the Stanford graph. As the process creates each vertex, two changes are made to the vertices. First, to avoid multiple versions of the same word, we use the lemma of the original word. Second, to avoid differences in the part of speech, we collapse the parts of speeches for all nouns and verbs to their base category. For example, we treat all plural nouns and proper nouns as just nouns. Similarly, verbs with different tenses are treated collectively as a single group. We use a very small stop word list to remove common determiners ("a", "an", and "the") from the SR. Additionally, we check if it is feasible to collapse adjective and noun modifiers into parent noun vertices. Figure 15 demonstrates this collapsing for sentence "5" as "graduate" and "student" are combined. By removing extraneous vertices from the SR, we reduce the overall size of each graph, which in turn, provides fewer irrelevant attributes to a machine learning algorithm and provides more concise patterns to be used in extracting access control policies.

4.2.2 Step 2: Classify Sentence

Next, REDE uses an ensemble classifier to determine whether a sentence contains an ACR or implies a DME. The ensemble classifier is composed of a k-NN classifier, a naïve Bayes classifier, and a SVM classifier. In prior work [Slankas & Williams, 2012], we found that if we used a similarity ratio that the distance had to be under to the nearest neighbor(s) in order to use the classification result, the k-NN classifier F_1 performance would be 1.0 (no misclassifications) , although not all of the sentences would be classified. As such, we decide to utilize multiple machine learning algorithms to produce the final classification result. If the k-NN classifier's similarity ratio is below a certain ratio (0.6) based upon the computed distance to the nearest neighbor(s) compared to the length of the sentence, we return the k-NN classifier's answer. Otherwise, we return a majority vote of the k-NN, naïve Bayes, and SVM classifiers. We term this classifier as "Combined SL."

The k-NN classifier works by taking a majority vote of the existing classifications of the k nearest neighbors to the item under test. Thus, in our situation, to classify a sentence, the classifier needs to find which existing classified sentences are most similar to the current sentence under test. k-NN classifiers use a distance metric to find the closest neighbors. This metric is the sum of the differences among the attributes used to determine the classification. Typically, Euclidean distance serves as a metric for numerical values while for nominal values (e.g., words), the distance is generally considered to be zero if both values are the same or one if they differ. Our situation is more complex as we have a variable number of attributes (words, parts of speech, named entities) to consider for each sentence based upon the sentence length. Additionally, certain words may be more closely related (e.g., the words are synonyms). As such, we need to utilize a custom distance metric to compute a value representing the difference between two sentences.

Our distance metric is a modified version of Levenshtein distance [Levenshtein, 1966]. Rather than using the resulting number of edits to transform one string into another as the value as the Levenshtein distance does, our metric computes the number of word transformations to change one sentence into another. Rather than strictly using just zero or one as the difference between words, the metric uses the function defined in Figure 16. The function first checks the structure of graph around each vertex to ensure the structure corresponds to the other vertex. Next, the function checks if the two vertices are the same (lemmas are equal). In line 7, we check if both words are numbers. Next, line 8 checks if both words are the same type of named entity such as a person or an organization. Then in line 9, the function checks if the two words are related through sets of cognitive synonyms (synsets) within WordNet[23] via semantic relationships (hypernym or hyponym). If a relationship value is found, then a value between 0.1 and 0.4 is returned based upon the number of relationships traversed. Finally, a default value of 1 is returned if none of the other conditions are met.

```
computeVertexDistance(Vertex a, Vertex b)
 1: if a = NULL or b = NULL return 1
 2: if a.partOfSpeech <> b.partOfSpeech return 1
 3: if a.parentCount <> b.parentCount return 1
 4: for each parent in a.parents
 5: if not b.parents.contains(parent) return 1
 6: if a.lemma = b.lemma return 0
 7: if a and b are numbers, return 0
 8: if ner classes match, return 0
 9: wnValue = wordNetSynonyms(a.lemma,b.lemma)
10: if wnValue > 0 return wnValue
11: return 1
```

Figure 16. Compute Vertex Distance Logic

Once we determine if the sentence is related to access control or not, the user may review the determination and correct it if necessary within the tool.

Figure 17 shows a screenshot of the tool's user interface. The top table contains the document with individual columns to display the line number, sentence type, assigned

[23] http://wordnet.princeton.edu/

classification, and completion status, assigned cluster (groups of similar sentences, optional functionality), and the sentence themselves. The dialog in the lower left allows users to review and manually enter or correct access control policies (discussed in the next section). The area in the lower right displays the SR.

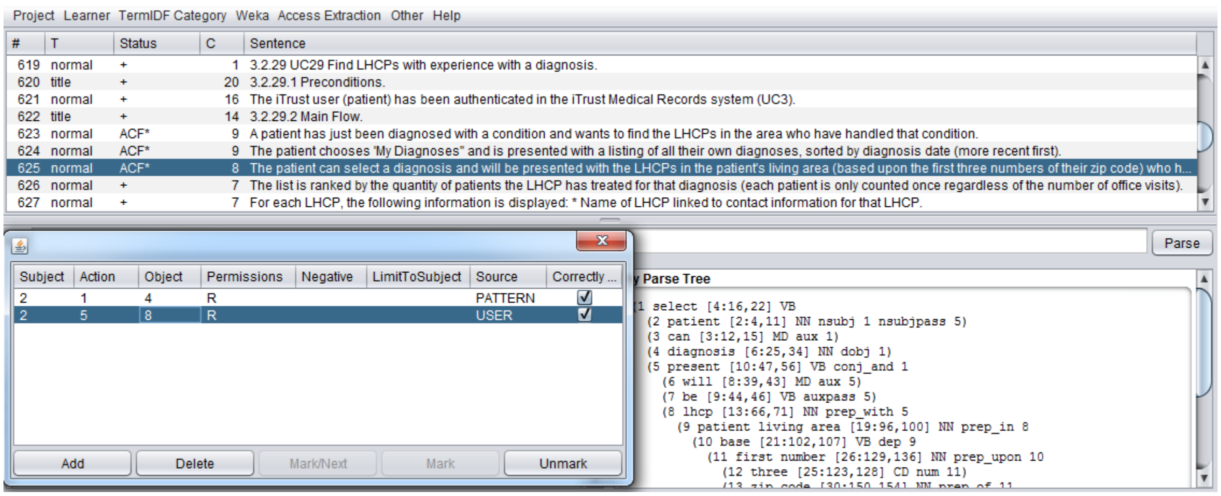


Figure 17. Tool Screenshot of Rule Extraction and Database Enforcement (REDE)

4.2.3 Step 3: Extract Access Control Elements

Next, we need to extract the subject, action, and resource elements from the SR. We construct a relation extraction algorithm for the identification of ACR elements and subsequent extraction of the ACR. The algorithm follows a well-known bootstrapping technique [Jurafsky & Martin, 2009] but has been adapted specifically for ACR extraction. The basic concept is to start with a small, well-known set of ACR patterns and then expand those patterns to find other, closely related patterns.

To initialize the algorithm (presented in Figure 18), we seed a set of ten basic ACR patterns with each pattern consisting of just three vertices as shown in Figure 19. Each pattern is the same, except a different verb[24] is used for a "Specific Action". Wildcards are used to match any noun in sentences containing the pattern. We initially chose the words "create", "retrieve", "update", and "delete" because the words are commonly associated with viewing and manipulating data. We then examined the frequencies of all verbs within the test documentation and chose to add more verbs associated with data and appearing with high frequencies within the document. Based on the application domain or other natural language product artifacts, users may choose a different set of starting actions. From these patterns, we match all occurrences of the subjects and resources within the document along with their associated frequency counts. From the counts, we compute the median values for the subjects and resources. We then assume any word that occurs more than the median belongs to the application domain. Without a threshold, the potential for misidentified subjects and resources is much greater as any word matching the pattern would be accepted.

[24] create, retrieve, update, delete, edit, view, modify, enter, select

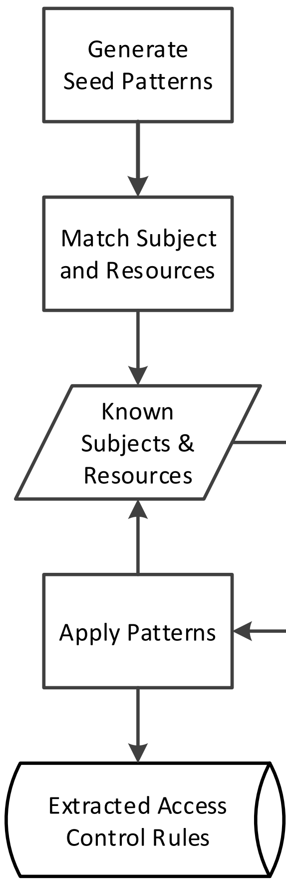


Figure 18. ACR Extraction Overview

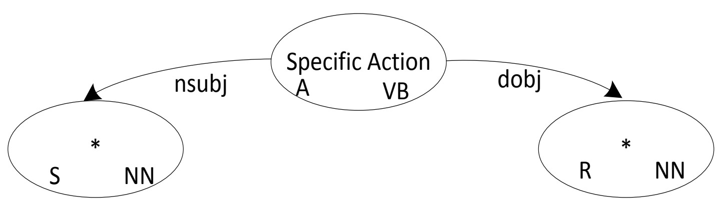


Figure 19. Basic ACR Seed Pattern

The process then stores the subjects and resources in a list of known subjects and resources. We also add any domain-specific words the user might have specified as input to the process to the known list of resources. From this listing, we then search the artifacts to see whether any subject exists along with any domain-specific words the user may have entered at the start of the process. For each sentence that does match the condition, we extract the dependency pattern between subject and resource vertices. We then assume that any verbs existing in that pattern are the actions. If more than one verb exists in the shortest path from the subject to the object, we combine the verbs but use the last appearing verb when defining permissions. In the sentence, "the administrator chooses to create a new patient", we combine "choose" and "create" to "choose create" for the action. The subject would be "administrator" and the object would be "patient". We derive permissions for each pattern by finding the closest synonym (via WordNet[25]) that has an already defined permission.

Once we extract the pattern, we apply a series of transformations to extract additional patterns that may locate additional ACRs. Specifically, we transform patterns that have an active voice into passive voice and vice versa. We also transform the patterns to assume conjunctions may exist for two or more subjects, two or more actions, and two or more resources. To find additional subjects and resources, we apply wildcards to the identified subject or resource vertices. Only the subject or the resource vertex is wildcarded to minimize semantic drift in bringing in unrelated sentences or patterns to access control.

Furthermore, to ensure that the patterns are appropriate, we apply a naïve Bayes classifier to judge whether an identified pattern is appropriate. To detect the features used by the classifier, we use a forward, stepwise selection model [Han et al., 2011]. From this model, we found the pattern itself (the POS tags and relationships between vertices) to have the biggest

---

[25] http://wordnet.princeton.edu/

influence. After that, the relationships between the resources and subjects had the next largest influence. After that, the identified parts of speech for the resources and subjects improved classification performance slightly. We did not use any further features (size, specific words, use of wildcards) since we found those features began to decrease the classifier's performance.

From the pattern set, we then search the natural language product artifacts for sentences matching one or more patterns. Once we find any match, we check to see whether other patterns match the same sentence. If more than one pattern matches and one pattern is contained within another pattern, we discard the former as the latter pattern provides a more specific match. Additionally, we check the matched sentences for any children vertices (of the matched pattern) that imply negativity or subject limitation (i.e., we check whether a relevant indicator exists just outside of the matched pattern).

The process then stores the extracted ACR in a list for validation and output to the user. The process adds any new subjects or resources to the list of known subjects and resources. If newly discovered subjects or resources exist, then the algorithm iterates until no new items or patterns are discovered. Once the algorithm converges, the user may inject two additional patterns into the process to find more ACRs. Similar to the Basic ACR Seed Pattern in Figure 19, the first injected pattern allows any pronoun to be a subject within an ACR pattern. The second pattern searches for only actions and resources, leaving resolution of the subjects to another algorithm. The injection occurs after the algorithm initially converges to avoid spurious matches that would occur if the patterns were injected at the start of the algorithm. Additionally, the user may manually identify ACR patterns through identifying ACRs in the sentences. The information from these patterns is fed into the algorithm to search for additional extracted elements.

For the sample sentences in Table 3, we would generate the ACRs as presented in Table 4. As the first sentence was not classified as having ACRs, the REDE process did not attempt to extract any ACRs. Ambiguous rules were generated for S2 and S4. The ambiguous words will be resolved in the fifth step when we examine ambiguous words for possible modifiers (e.g., “of courses”).

Table 4. Extracted ACRs for the Sample Sentences

ID	Extracted ACRs
S1	*none*
S2	(system, display, list, read) (system, display_by, professor, read)
S3	(professor, select, course, read)
S4	(system, display, list, read) (system, display_for, course,read)
S5	(instructor, enter, grade, create) (instructor, enter_for, graduate student, read)

4.2.4 Step 4: Extract Database Model Elements

After extracting the ACRs, the process builds upon the knowledge gained to extract the DMEs. As with the previous step, we construct an algorithm to identify and extract DMEs. As before, the algorithm follows a bootstrapping technique [Jurafsky & Martin, 2009] adapted for database model extraction. The algorithm can extract entities, entity-attributes, and relationships that come together to create a system’s database model.

To initialize the algorithm (presented in Figure 20), we use the set of subjects and resources discovered in the prior step as the initial set of entities. For the set of relationship words, we use any extracted actions that were not part of the set of list verbs (e.g., create, retrieve, update, etc.) used to the basic ACR patterns. From these two sets of “known entities and relations” and the eight template patterns (Figure 21), we form the initial set of patterns. To expand patterns, we replace specific words within vertices of entities or relationship with wildcards that can match any other word with the same part of speech. To minimize semantic drift, we only allow one vertex within a pattern to be wildcarded. We also transform patterns that have active voice into passive voice and vice versa.

Once the patterns have been expanded, we apply a naïve Bayes classifier to judge whether or not the new pattern is appropriate. The classifier is trained using domain-independent features such as relationships and parts of speech from product artifacts previously analyzed. The process then searches for instances of the patterns within the natural language product artifacts being analyzed. The DMEs are extracted for found matches. Additionally, we look to see if these elements were previously discovered or not. If they are new, we add the elements to the list of known entities and relationships. The process repeats until no new elements are extracted. The algorithm is design to allow injection of patterns that may not be suitable for an initial search as they contain issues that may lead to spurious results (e.g., only one element is a relationship is defined), but could lead to the discovery of additional entities, attributes, and relationships that would otherwise be missed. Additionally, the user may manually identify database model patterns that can be injected into the process.

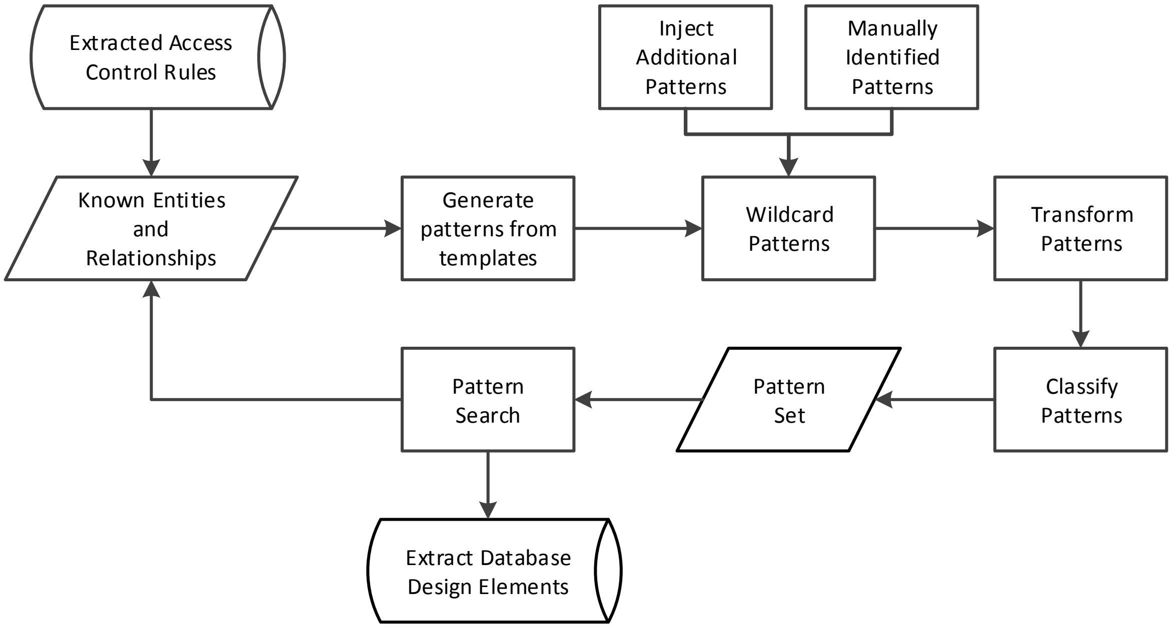


Figure 20. Database Design Extraction Overview

Figure 21 presents eight basic template patterns. These patterns were constructed from the sample sentences and heuristics from prior work [Chen, 1983; Hartmann & Link, 2007; Vidya Sagar & Abirami, 2014] and by evaluating the pattern frequencies from the solved test data sets in Omar's work [Omar, 2004].

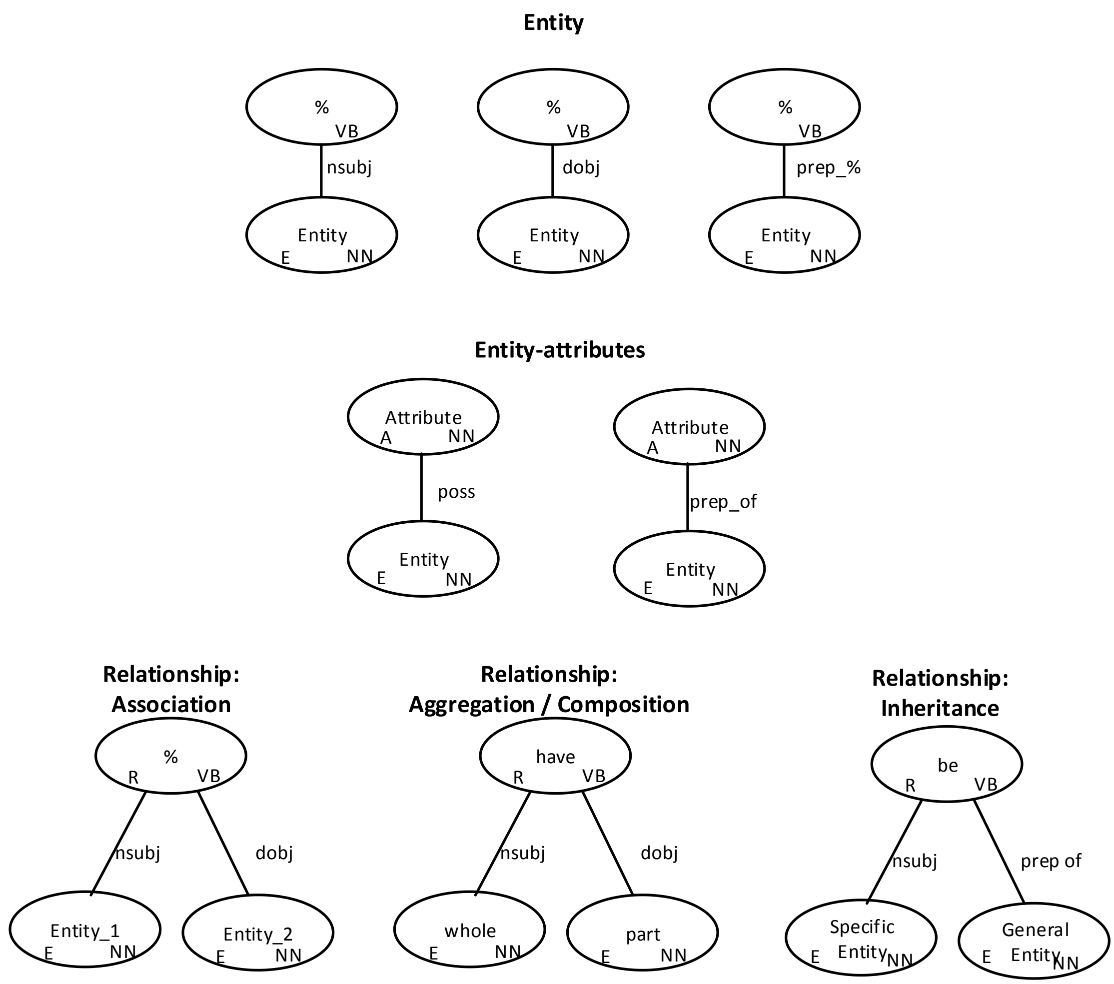


Figure 21. Database Design Template Patterns

To mark whether or not a relationship was an association, aggregation / composition, or inheritance, we examined the verbs involved within the graph patterns. If the verb was one of "have", "compose", "contains", or "belongs", we consider the pattern as an aggregation/composition relationship. If the verb contains "be", we consider the pattern as an inheritance relationship. If neither of those two conditions is met, we consider the relationship as an association by default.

For the sample sentences in Table 3, we discover the entities as listed in Table 5. Note that a relation was not created for "(select, professor, course)" as the corresponding ACR just had read permissions.

Table 5. Extracted DMEs for the Sample Sentences

ID	Extracted DMEs
S1	*None*
S2	Entities: course, professor Relations: (teach, professor, course)
S3	Entities: professor, course Relations: *none*
S4	Entities: student, course Relations: (enroll, student, course)
S5	Entities: instructor, grade, graduate student Relations: (enter, instructor, grade), (grade, grade, graduate student)

4.2.5 Step 5: Map data model to physical database schema

To map the DMEs and ACRs to a physical database schema, REDE utilizes a series of steps. First, REDE creates additional ACRs based upon database relations that are not already ACRs. As the process creates access control rules at the table level, the process checks any ACRs defined with an attribute as the resource and replaces the attribute with the corresponding entity. Next, REDE resolves ambiguous or missing subjects by using the last known defined subject. REDE then resolves ambiguous resources (i.e., those objects defined by pronouns or ambiguous terms such as "data") by using the last known defined resource. REDE next merges any ACRs sharing the same subjects and resources. Finally, in conjunction with guidance from the user, REDE maps resources in the ACRs to the defined physical database schema for an application. Similarly, the process maps subjects to roles.

4.2.6 Step 6: Implement Role-based Access Control

Once all of the previous steps have been completed for all of the sentences in the natural language product artifact, the tool produces a series of reports. First, the tool generates a report with access conflicts or other validation issues. The process compares each ACR to all of the other ACRs to find conflicts that occur when one sentence gives a role access to a table, but another sentence would revoke the same role's access from the table. The report also includes any subjects or actions not mapped to their corresponding database elements as well as any sentences classified as access control, but missing part of the access control tuple. As necessary, the user makes corrections to the extract data to fix discovered issues.

Another report details coverage of discovered entities and resources to tables in the system's persistence layer. As we assume a default of no-access, 100% coverage is not required. However, low coverage values may indicate a need for further access control policies. Coverage may vary based upon the product artifacts used in the analysis.

Finally, the tool produces the SQL commands to establish role-based access control within the database. Specifically, necessary roles are created, and the privileges expressed through access control sentences are assigned to those roles.

Roles are created through the command

`CREATE ROLE` *`role_name`*

Privileges are granted to a role by

`GRANT` *`permission_list`* `ON` *`table_name`* `TO` *`role_name`*

where `permission_list` is one or more of the following values: `DELETE, INSERT, SELECT, or UPDATE.` The relevant permissions were assigned based upon the identified action.

In situations where permissions need to be explicitly revoked from a user, the process generates the following command

```
REVOKE permission_list ON table_name FROM role_name
```

The tool can generate a report showing the traceability of access control rules back to the originating sentences in the natural language product artifacts.

4.3 Additional Challenges and Solutions

Many significant, complex problems exist for NLP, IE, and ML; these problems carry forward into ACR and DME extraction. In this section, we detail many of the issues that arise in these areas.

4.3.1 Ambiguity

Ambiguity issues appear in many contexts, including what meaning words have. For example, the sentence "the bank collapsed yesterday" creates confusion as one does not know whether the "bank" was a financial institution or the side of levee [Jurafsky & Martin, 2009]. Systems must disambiguate such terms based on the context – within the sentence itself or other sentences in the document. Similarly, the same word can create issues for ACR extraction. For example, "the patient *enters* his zip code to find the closest physician" and "the administrator *enters* a new patient" have separate permissions implied for the verb "enter". In the first case, the patient searches for physicians based on a zip code, so a read permission is necessary. In the second case, the administrator effectively creates a new patient in the system and, hence, some form of a create permission is necessary. To solve this issue, we rely on the structure of the subgraph produced by the process to distinguish when different permissions may be necessary. A third situation arises for ambiguity when words do not clearly indicate the relevant entity or resource. To detect this type of ambiguity, we adopted Park et al.'s solution [Park et al., 2000] to maintain a word list that indicated ambiguity. To resolve some instances of this ambiguity, the process looks to see if a preposition follows the ambiguous word. If so, the process uses the object of the preposition to resolve the ambiguity. For example, in the sentence "A doctor select from a list of medications to complete the prescription", "list' is ambiguous. However by using the information in the prepositional, the process can determine list refers to a medication list. In other situations, the system may not be able to resolve the ambiguity and highlights such cases to a user. Ambiguity issues that arise with other words such as "entry", "data", and "record" can also be considered as resolution issues (see Section 4.3.5) in which the ambiguous word needs to tie back to the referent.

4.3.2 Synonyms

Synonyms for the same term are another frequent source of ambiguity in NL texts. For example, "professor" and "instructor" may refer to the same role in sentences S3 and S5 in Table 3. Other cases may arise when a writer use a subset of title to refer to the same role as in "lab technician" and "technician". To resolve the first situation, the process utilizes WordNet to see if two words are within the same synset. If so, the process considers them synonym candidates. The process also examines whether or not one term is contained within another. If so, those two terms are considered possible synonyms for each other. We chose to leave the final determination to users as it may be infeasible to distinguish synonyms reliably versus those terms that are closely related, yet distinct.

4.3.3 Negativity

Negativity, denying users access to a specific action and/or resource, is a critical problem for access control. In certain cases, users may be granted access to part of a record, but then explicitly denied from reading other parts of that record. Negativity can appear in multiple ways within sentences: negative determiners (e.g., no, zero, neither), adjectives (e.g., unable), nouns (e.g., none, nothing), verbs with negative connotation (e.g., stop, prohibit), and adverbs (e.g., never) [Huddleston & Pullman, 2002]. All such possibilities should be considered, and the identified negativity should be tied back to the appropriate subject, action, or resource.

From the Dixon's negation concepts [Dixon, 2005], we utilize multiple methods to detect negation within a sentence. We can detect specific adjectives (unable), adverbs (not, never), determiners (no, zero, neither), and nouns (none, nothing). We also utilize specific negative verbs (stop, prohibit, forbid) as well as a pre-determined list of negative prefixes associated with English words.

4.3.4 Role Limitation

Within a system, privileges are often limited to one or more specific roles. For the sentence "only an administrator can maintain system lookup tables", we need to grant permission to the administrator role while restricting the permission from all other roles. Within English, restrictive focus modifiers [Huddleston & Pullman, 2002] express such limitations. As the position of such modifiers (e.g. just, only) affect the overall semantic meaning of the sentence, we restrict where such modifiers are located. For example, consider the sentence, "Doctors write prescriptions." We assume the doctor has the privilege to write prescriptions (possibly an insert permission on a "prescription" table within a relational database). However, if the word "only" is placed in the sentence, then the meaning of the sentence varies based upon the location of "only". With "only" as the first word, the sentence means that only doctors write prescriptions and no other roles. Alternatively, if "only" is the second word, it modifies "write" and implies that the only action doctors can do in the system is write prescriptions. Finally, if "only" is the third word, the sentence now means that prescriptions are the only things doctors write. To ensure the modifiers are in the correct location, we set the limit flag for a sentence if the modifier applies to the subject identified in the sentence. If the modifier exists elsewhere, we flag the sentence for having a potential problem.

4.3.5 Resolution

Resolution issues also appear in ACR extraction. One common situation is the appearance of pronouns in place of the actual subjects or resources. A specific case, anaphora resolution, exists when a pronoun or other phrase (e.g., that, there) refers to a previously occurring noun or entity (the antecedent). A similar resolution situation occurs when generic terms such as "data", "entries", and "records" appear in the text. Another situation occurs when the system appears to be the subject (S4 in Table 3) when the actual actor can be identified from a prior sentence. REDE adopt the same solution as Xiao et al. [Xiao et al., 2012]used with Text2Policy using Kennedy et al.'s anaphora resolution algorithm [Kennedy & Boguraev, 1996]. This algorithm replaces the word needing resolution with the last known actor or resource depending upon which part of an ACR is missing. However, such an approach is not without problems. Consider these two sentences from Caramazza et al.'s work [Caramazza et al., 1977]: (1) John telephoned Bill because he wanted some information. (2) John telephoned Bill because he withheld some information. In the first sentence, “he” refers to John while in the second sentence “he” refers to Bill. As the resolution process is not central to our research goal, we accept that the pronominal resolution process may result in false choices.

4.3.6 Missing Elements

Situations exist in which not all of the ACR elements may be present within a single sentence. REDE allows for only subjects to be missing. While users may manually identify ACRs with missing resources, REDE has no support to find such patterns. The process uses anaphora resolution to fill in missing elements. When such resolution is not possible, the process highlights ACRs with a missing element and developers would be pointed to the surrounding sentences to finish defining the ACRs outside of the automated process. Actors may often be missing in sentences from user and training manuals as steps often indicate the actions to be performed on resources; the actor is either assumed to the “user” or a specific role listed in early the surrounding text.

5 Classification Study

Just as with functional requirements, a system's success depends greatly upon adherence to non-functional requirements (NFRs). When NFRs are missed or ignored, significant, costly issues can arise. A recently deployed U.S. Army intelligence sharing application costing $2.7 billion has been labeled as useless due to capacity, performance, and usability issues [Hoskinson, 2011]. Electronic health record (EHR) systems have been severely criticized for lack of usability, which has been one of the prime reasons why some EHRs adoption have failed [Bertman & Skolnik, 2010].

Given the need to analyze and implement NFRs from a wide variety of available sources, system analysts need to identify and categorize NFRs quickly. Business analysts need to ensure completeness of their work. System architects need to understand constraints such as availability, performance, and recovery capabilities to design appropriate architectures to meet those constraints. System integrators, those who deploy large-scale software systems, need to be aware of NFRs in order to place systems into appropriate environments where NFRs such as capacity, security, and operational constraints are appropriately satisfied. Customers need usable applications they can immediately operate with minimal training and with appropriate user interface designs to minimize errors.

Software documentation continues to be a neglected best practice, despite persistent pleas from educators and practitioners alike [de Souza et al., 2005]. Open-source projects often lack formal requirements with developers asserting new requirements themselves. For many open-source projects, archived mailing lists and message boards are the only available development documentation [Noll & Liu, 2010]. However, open-source projects typically contain administrative, install, and user manuals because the contained information is necessary for others to utilize the open-source system. Government and industry-based standards contain critical NFRs for projects. Sifting through all of the possible sources for NFRs, though, is a tedious, time-consuming effort.

The goal of our research is to aid analysts in more effectively extracting relevant non-functional requirements in available unconstrained natural language documents through automated NLP.

To meet this goal, we define a tool-based approach, which we call NFR Locator, to classify and extract sentences in existing natural language texts into their appropriate NFR categories. The classification is necessary to determine what role a sentence has. From sentences marked as non-functional, we would then extract critical information specific to each NFR category. For example, access control NFRs would need subjects, resources, and actions extracted to create relevant access control policies. We categorized sentences into one of these 14 NFR categories: (1) access control, (2) audit, (3) availability, (4) capacity and performance, (5) legal, (6) look and feel, (7) maintainability, (8) operational, (9) privacy, (10) recoverability, (11) reliability, (12) security, (13) usability, and (14) other. With NFR Locator, individuals would utilize pre-defined classifiers to locate and extract NFRs from existing natural language documents. If necessary, they can use the tool in an interactive mode to train additional classifiers with a new domain or document types.

To evaluate our approach, we developed the following research questions:

RQ1.1: What document types contain NFRs in each of the 14 different categories?

RQ1.2: What characteristics, such as keywords or entities (time period, percentages, etc.), do sentences assigned to each NFR category have in common?

RQ1.3: What machine learning classification algorithm has the best performance to identify NFRs?

RQ1.4: What sentence characteristics affect classifier performance?

The problem domain of classifying sentences NFRs equates to classifying sentences for access control and database objects.

This study has the following contributions:

- Process and tool to identify NFRs by categories within available natural language documentation
- Distribution report containing the NFR categories and frequencies by document type
- Empirical performance results for machine learning classifiers on NFRs
- Sentence similarity algorithm to use with a *k*-nearest neighbor classifier or a *k*-medoids clustering algorithm.
- Publically-available labeled corpus of identified NFRs[26]

The rest of this chapter is organized as follows: Section 5.1 presents our approach. Section 5.2 describes our research methodology. Section 5.3 presents our evaluation. We conclude this chapter with a discussion and overview of the limitations in Section 5.4.

[26] https://github.com/RealsearchGroup/NFRLocator

5.1 Approach

While NFRs have existed since the early days of software engineering, consensus does not exist for the name or the definition of an NFR [Glinz, 2007]. Many simply refer to them as the "ilities," the quality aspects of a system. Others have taken to labeling NFRs as systemic requirements [Browne, 2008]. The IEEE Recommended Practice for Software Requirements terms NFRs as constraints [IEEE Recommended Practice for Software Requirements Specifications - IEEE Std 830-1998, 1998]. Our concern resides with how NFRs place different constraints on systems, how to quickly identify such constraints, and then to extract relevant information for the NFR. While the number of such constraint categories is rather large (Lawrence Chung et al. identified 156 NFR categories [Chung et al., 2000]), we chose to concentrate on 14 categories frequently appearing in literature and practical use.

To extract NFR sentences within existing unconstrained natural language documentation, we define a two-step process, NFR Locator. For input, the process takes any natural language document related to a project. The process parses the natural language into an internal representation and then based upon relevant features, classifies sentences into specific NFR categories or returns "not applicable" if the sentence does not specify an NFR.

5.1.1 Step 1: Parse Natural Language

The process begins by entering the text into the system, parsing the text and converting the parsed representation into NFR Locator's sentence representation (SR). The SR represents each sentence as directed graph where the vertices are words, and the edges are the relationships between words.

The tool parses text with the Stanford Natural Language Parser (NLP) and, for each sentence, outputs a graph in the Stanford Type Dependency Representation (STDR) [de Marneffe et al., 2006]. We choose the STDR as it incorporates the sentence's structural information in a concise and usable format.

From the STDR generated by the parser, we create our SR. Figure 22 shows the SR for the sentence "The system shall terminate a remote session after 30 minutes of inactivity." Although, in general, the SR can be considered a tree, situations exist (primarily due to conjunctions) in which a vertex has multiple parents. Vertices correspond to words in the sentence and contain the word, the word's lemma and collapsed part of speech. Edges correspond to the relationship between two words (unchanged from the STDR). For prepositions, edges also represent the corresponding word (e.g., "of" and "after" by "prep_of" and "prep_after"). Utilizing a pre-order traversal, the process creates the SR from the Stanford graph. As the process creates each vertex, two changes are made. First, to avoid multiple versions of the same word, we use the lemma of the original word. Second, to avoid differences in the part of speech, we collapse the parts of speeches for all nouns, verbs, and adjectives to their base category. For example, we treat all plural nouns and proper nouns as just nouns. Similarly, verbs with different tenses are treated collectively as a single group. We also utilize a very small stop word list to remove common determiners[27] from the SR as demonstrated in Figure 22 with the dashed lines.

[27] a, an, the

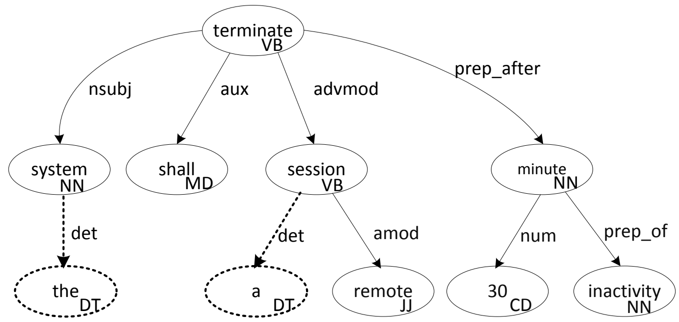


Figure 22. NFR Locator Sentence Representation

5.1.2 Step 2: Classify Sentences

Once the tool completes the parsing and initial analysis of a sentence, a k-NN classification algorithm classifies each sentence into one or more NFR categories. Sentences classified besides "not applicable" appear on generated reports from the tool for use outside of the system.

A k-NN classifier predicts a classification by taking a majority vote of the existing classifications of the k nearest neighbors to the item under test. Thus, in our situation, to classify a sentence into one of the 14 categories, the classifier needs to find which existing classified sentences are most similar to the current sentence under test. k-NN classifiers use a distance metric to find the closest neighbors. This metric is the sum of the differences among the attributes used to determine the classification. Typically, Euclidean distance serves as a metric for numerical attributes while for nominal values, the distance is generally considered to be zero if both attribute values are the same or one if they differ. Our situation is more complex as we have a variable number of attributes to consider for each sentence based upon the sentence length. Additionally, certain words may be more closely related (e.g., two words are synonyms). As such, we need to utilize a custom distance metric to compute a value representing the difference between two sentences.

Our distance metric is a modified version of Levenshtein distance [Levenshtein, 1966]. Rather than using the resulting number of edits to transform one string into another as the value as the Levenshtein distance does, our metric computes the number of word transformations to change one sentence into another. Rather than strictly using just zero or one as the difference between words, the metric uses the function defined in Figure 23. The function first checks the structure of graph around each vertex to ensure it corresponds to the other vertex. Next, the functions checks to see if the two vertices are the same (lemmas are equal). In line 7, we check if both words are numbers. Next line 8 checks to see if both words are the same type of named entity such as a person or an organization. Then in line 9, the process checks to see if the two words are related through sets of cognitive synonyms (synsets) within WordNet[28] via semantic relationships (hypernym or hyponym). If a relationship value is found, then a value between 0.1 and 0.4 is returned based upon the number of relationships traversed. Finally, a default value of 1 is returned if none of the other conditions are met.

Once the classification is complete, the user may review the predicted classifications and related sentences. If necessary, they can correct the classifications within the tool. As the user completes each classification, those classifications would be used within the k-NN classifier as additional sentences are processed.

[28] http://wordnet.princeton.edu/

```
computeVertexDistance(Vertex a, Vertex b)
if a = NULL or b = NULL return 1
if a.partOfSpeech <> b.partOfSpeech return 1
if a.parentCount <> b.parentCount return 1
for each parent in a.parents
if not b.parents.contains(parent) return 1
if a.lemma = b.lemma return 0
if a and b are numbers, return 0
if ner classes match, return 0
wnValue = wordNetSynonyms(a.lemma,b.lemma)
if wnValue > 0 return wnValue
return 1
```

Figure 23. Compute Vertex Distance Logic

5.2 Research methodology

This section describes the context regarding our problem and NFR categories. We also discussed how we collected relevant documents and prepared those documents for use in our evaluations.

5.2.1 Context

We chose the EHR domain to evaluate our research questions initially. A wide variety of open and closed source EHRs, various industry standards (HL7[29], CCHIT[30]), governmental regulations, and other document sources exist to elicit documentation. While not directly to healthcare, we included the PROMISE NFR Data Set [Menzies et al., 2012] in our evaluations to provide comparisons to prior research.

[29] http://hl7.org
[30] http://cchit.org

The nine initial NFR categories were chosen based upon existing NFR classification work [Cleland-Huang et al., 2007; Casamayor et al., 2010; Zhang et al., 2011b] with the PROMISE NFR Data Set. From industry experience, we added reliability and recoverability. We also combined performance and scalability into a single category as the two concepts are considered together in system architecture and design activities. To meet future research needs, we separated access control and audit from security. As there were specific NFRs that did not readily fall into any other existing categories, we added “other” to categorize these NFRs. By using the “other” category, we avoid placing NFRs into categories in which they do not belong. If the “other” NFRs were placed into existing categories, machine learning algorithms would perform less effectively due to irrelevant terms and sentences within a specific category.

5.2.2 Study Procedure

To perform this study, we first collected a series of 11 documents related to EHRs. We list the specific source locations within our available labeled corpus[31]. For requirement specifications, we utilized the CCHIT Ambulatory Requirements, iTrust [Meneely et al., 2011], and the PROMISE NFR Data Set. To represent documents from an open-source project without a formal requirements document, we utilized the OpenEMR[32] Install Manual and User Manual. We analyzed two data use agreements (DUAs) from the Centers for Medicare & Medicaid Services[33] and the North Carolina Department of Public Health. DUAs are legal contracts among two or more parties that specify what data is shared, who can access the data

[31] https://github.com/RealsearchGroup/NFRLocator
[32] http://www.open-emr.org/
[33] http://www.cms.gov/

(authorizations), and for what purpose the data may be used (purposes) [Schmidt et al., 2011]. We also used two request for proposals (RFPs) from organizations within the state of California for EHR systems. Many organizations use RFPs to solicit vendor bids for software systems. The RFPs often contain detailed lists of requirements for vendors to provide a response as to their capabilities in meeting a particular requirement. To represent governmental regulations, we utilized three sections of the United States Code of Federal Regulations (CFRs) related to healthcare.

We then converted those 11 documents into a text-only format using the "save as" feature in the relevant document application. The only changes made to the resulting text files were to account for misplaced line breaks and to remove table-based information that the natural language processor cannot process. Next, we imported each document into NFR Locator. First, we performed cluster analysis on the imported document to look for common sentences and detect patterns among those sentences. Each sentence (or line) in the file was then manually classified into (1) one or more NFR categories; (2) functional (FT); or (3) not applicable (NA). We allowed a sentence to be placed into multiple categories with the exception of "not applicable." For example, we categorize "The system has capability to electronically provide patient reports on demand following and allow for hiding private information to comply with Health Insurance Portability and Accountability Act (HIPAA) Privacy and Security requirements" into functional and into two NFR categories (legal and privacy) due to the different sentence elements. Table 6 presents the criteria used to classify sentences manually into specific NFR categories.

Table 6. NFR Category Criteria

NFR Category	Criteria
Access Control	Selective access or restriction on a particular subject to a specific object to perform a defined object. Sentence deals primarily with authorization. Authentication is considered part of "security".
Audit	Ability to track what has occurred within a system. Such sentences will have the capability to answer who, what, when, where, how, and why questions.
Availability	Does the sentence state when the system needs to be operational for use? Are certain time periods restricted for activities such as batch processing, system upgrades, backups, etc.? Would the system function in a limited functionality or capacity in such situations?
Legal	Does the sentence refer to a legal requirement?
Look and feel	Does the sentence specify how the application needs to appear to users?
Maintenance	Does the sentence references to standards? Are there references to flexibility and portability listed?
Operational	Does the sentence include references to execute on specific platforms or environments? To integrate with specific sentences? To comply with any standards for data formats for storage/transmission?
Performance and Scalability	Does the sentence contain references to capacity (volume of users and data), throughput, growth patterns, performance (timing characteristics of the software such response and processing times), efficiency based requirements, or scalability?
Privacy	Does the sentence relate to a privacy policy or related law? Does the sentence contain restrictions on the specific use, collection, and disclosure of information?
Reliability	Does the sentence refer to the acceptable frequency and severity of failures?
Recoverability	Does the sentence refer to the time needed to recover from issues? Are acceptable data losses described in the event of failures? Backup frequencies and types?
Security	Does the sentence refer to a security objective (confidentiality, integrity, authentication, non-repudiation etc.) not mentioned elsewhere in this set of NFRs?
Usability	Does the sentence contain factors that affect how an application is understood and operated by users? Does the sentence contain localization or internationalization factors?

We validated the classifications through several approaches. First, we computed clusters for all of the different sentences within a document and then compared the assigned labels and sentence content. We also generated listings by classification of the sentences to check for mislabeled sentences. As we labeled documents, we used already classified sentences in the k-NN classifier to examine similarities among the current sentence being labeled and those already classified. Any discrepancies in the predicted categories could easily be traced back to the source sentences and appropriate changes made. An internal validity threat may exist as the first author performed all of the initial sentence classifications. To check if this was a significant issue or not, we had six software developers classify a representative sample of 30 sentences. Utilizing Randolph's Online Kappa Calculator [Randolph, 2008], we had a free-marginal kappa of 0.714, indicating adequate inter-rater agreement, by comparing the first authors selections against the majority vote of the other raters.

Once we completely labeled the documents, we performed various experiments on the labeled data through the use of different classification algorithms and sentence features. To evaluate classifiers, we tested each classifier with a stratified n-fold cross-validation and computed the precision, recall, and F_1 measure. With the n-fold cross-validation, data is randomly partitioned into n folds based upon each fold of approximately equal size and equal response classification. For each fold, the classifiers are trained on the remaining folds and then the contents of the fold are used to test the classifier. The n results are then averaged to produce a single result. We follow Han et al.'s recommendation [Han et al., 2011] and use 10 as the value for n as this produces relatively low bias and variance. The cross-validation ensures that all sentences are used for training and each sentence is tested just once. For our evaluations, we utilized standard machine learning classifiers within Weka [Hall et al., 2009] suite in addition to the k-NN classifier and distance function presented in Section 4. To generate input data for the Weka classifiers, we exported the original sentences as text strings along with

Boolean flags for each of the NFR categories. We converted all strings to word vectors through a Weka filter. As the Weka classifiers do not natively support multi-label classifications, we train a Weka classifier for each NFR category. Each category is independently test for membership.

5.3 Evaluation

This section presents our evaluation of the research questions.

RQ1.1: *What document types contain NFRs in each of the 14 different categories?*

With RQ1.1, we look to see which document types are better sources for each of the 14 NFRs categories. Table 7 presents the documents with their classification breakdown. (Column abbreviations are listed in Table 8.) Our breakdown for the PROMISE NFR Data Set [Menzies et al., 2012] differs due to category changes and for classifying each sentence individually.

All evaluated document contained NFRs. RFPs had a wide variety of NFRs with the exception of look and feel NFRs. Due to their directed purpose, DUAs contained high frequencies of legal and privacy NFRs. Access control and/or security NFRs appeared in all of the documents. The low frequency of functional and NFRs with CFRs exemplifies why tool support is critical to extract requirements efficiently from those documents.

Table 7. Classified Documents by Requirements Category

Document	Document Type	Size	AC	AU	AV	LG	LF	MT	OP	PR	PS	RC	RL	SC	US	OT	FN	NA
CCHIT Ambulatory Requirements	Requirement	306	12	27	1	2	0	10	0	0	1	5	2	28	4	8	228	6
iTrust	Requirement, Use Case	1165	439	44	0	2	2	18	2	9	0	9	9	55	2	0	734	376
PromiseData	Requirement	792	164	20	36	10	50	26	89	7	75	4	12	71	101	19	340	0
Open EMR Install Manual	Installation Manual	225	3	0	0	0	0	0	5	1	0	6	1	25	0	0	2	184
Open EMR User Manual	User Manual	473	169	0	0	0	14	0	0	0	0	0	0	8	4	0	286	95
NC Public Health DUA	DUA	62	1	0	0	20	0	0	0	4	0	0	0	1	0	0	0	41
US Medicare/Medicaid DUA	DUA	140	1	0	0	26	0	0	0	17	0	0	0	0	0	5	2	108
California Correctional Health Care	RFP	1893	94	120	9	85	0	133	94	52	13	16	13	193	14	38	987	409
Los Angeles County EHR	RFP	1268	58	37	8	3	2	28	19	3	11	8	13	108	21	10	639	380
HIPAA Combined Rule	CFR	2642	28	8	3	0	0	78	0	213	0	9	0	41	1	0	317	2018
Meaningful Use Criteria	CFR	1435	0	0	0	0	0	0	0	0	0	0	0	8	0	0	116	1311
Health IT Standards	CFR	1475	10	20	0	0	0	119	0	1	0	2	2	71	1	2	164	1146
Total		**11876**	**979**	**276**	**57**	**152**	**68**	**413**	**207**	**300**	**100**	**50**	**43**	**563**	**148**	**82**	**3568**	**6076**

RQ1.2: *What characteristics, such as keywords or entities do sentences assigned to each NFR category have in common?*

To evaluate RQ1.2, we performed two tasks. First, we extracted the top 20 keywords for each category based upon their ability to predict whether a sentence belongs to a specific category. Using Equation (10), we ranked the probability of every keyword for each NFR category. The first and second parts of the equation are the term frequency-inverse document frequency (TF-IDF) commonly used for information retrieval. The third term applies a selectivity factor to the probability to improve results where the term resides primarily in one classification versus multiple categories. $N_{K,C}$ represents the number of sentences containing the keyword for a particular category. N_C is the total number of sentences in the category. N

denotes the total number of sentences. N_K is the number of sentences containing the keyword. $tf - idf_C$ equals the TF-IDF for a particular category and term. We eliminated any results in which there were not at least three occurrences of a keyword within a class.

$$P_k = \frac{N_{K,C}}{N_C} \times \log(\frac{N}{N_K}) \times \frac{tf - idf_C}{\sum_{i \in C} tf - idf_i} \tag{10}$$

Table 8 presents the top 20 keywords found for each category. The terms extracted for the maintenance category primarily represent specific standards healthcare system must implement. In future work, we will break these requirements into their own category, "data standards." Also note that document specific words do appear in Table 8. This issue could be resolved by adding an additional term to Equation (10) to account for document specificity as Cleland-Huang et al. performed [Cleland-Huang et al., 2007] or by adding more documents to the corpus.

Table 8. Top 20 Keywords by Category

NFR Category	Keywords
Access Control (AC)	choose, lhcp, hcp, visit, privilege, read, office, add, representative, sort, name, administrator, personal, dlhcp, view, status, accessor, edit, role, list
Audit (AU)	authorship, trail, arise, worksheet, auditable, exclusion, reduction, deletion, examine, editing, stamp, non-repudiation, inclusion, id, alteration, finalize, disable, summarize, attestation, log
Availability (AV)	achieve, 24, availability, 98, addition, available, 99, hour, day, online, schedule, confidentiality, resource, technical, year, transmit, integrity, maintenance, %, period
Legal (LG)	infeasible, custodian, hipaa, breach, dua, discovery, iihus, publication, iihi, recipient, delay, secretary, definition, harm, scope, jurisdictional, affect, derive, vocabulary, reuse
Look & Feel (LF)	appearance, scheme, tree, radio, appeal, color, look, navigation, sound, feel, ship, left, shot, menu, ccr, button, corporate, page, openemr, employer
Maintenance (MT)	4010, washington, ibr, x12n, asc, 2002, addenda, 837, september, 1999, 1.1, tele-communication, 5.1, astm, draft, february, january, 2010, context-aware, infobutton
Operational (OP)	mysql, microsoft, euhr, soms, letter, infrastructure, interoperability, connect, cchcs, machine, browser, platform, cardmember, central, cdcd, extraction, cchc, model, registry, interchange
Privacy (PR)	health, protected, entity, disclose, covered, use, disclosure, individual, such, purpose, law, permit, other, section, require, plan, person, paragraph, care, request
Recoverability (RC)	restore, credentials, backup, back, recovery, disaster, previous, emergency, establish, copy, state, need, implement, loss, plan, event, failure, organization, business, hour
Performance & Scalability (PS)	fast, simultaneous, 0, second, scale, capable, increase, peark, longer, average, acceptable, lead, handle, flow, response, capacity, 10, maximum, cycle, distribution
Reliability (RL)	reliable, dependent, validate, validation, input, query, accept, loss, failure, operate, alert, laboratory, prevent, database, product, appropriate, event, application, capability, ability
Security (SC)	cookie, encrypted, ephi, http, predetermined, strong, vulnerability, username, inactivity, portal, ssl, deficiency, uc3, authenticate, certificate, session, path, string, password, incentive
Usability (US)	easy, enterer, wrong, learn, word, community, drop, realtor, help, symbol, voice, collision, training, conference, easily, successfully, let, map, estimator, intuitive

In the second task for RQ1.2, we looked at selected NFR categories to find commonality. Within availability NFRs, we found that 30 sentences contained some form of the word "available", 20 sentences contained a time period, nine expressed a percentage, six stated some form of a maintenance window, and five mentioned replication. Only four of the 57 sentences did not contain one of these features. (Note: a sentence could express more than one of the features.)

Depending upon the presence of those features in other requirements, they may be suitable candidates for use in classifiers for their ability to discern availability NFRs. Alternatively, a set of hand coded rules could be used to process availability NFRs as there appears to be a relatively small number of patterns for this NFR category.

RQ1.3: What machine learning classification algorithm has the best performance to identify NFRs?

In addition to the k-NN classifier, we used the multinomial naïve Bayes and the sequential minimal optimization (SMO)[34] classifiers in Weka [Hall et al., 2009]. We also used two random methods, weighted and 50%. The weighted random model assumes that the classifier knows the population proportions within the training set and randomly returns answers proportionally to the existing classifications. The 50% model has no knowledge of the training set and randomly returns answers with equal likelihood between the two classifications values. As we allow for each sentence to have one or more classifications, we test each category individually for each classifier and produce the precision, recall, and F_1 values. We then

[34] SMO is a Support Vector Machine classifier

average the results of all of the categories for each classifier to produce the results displayed in Table 9. This "macro-averaging" allows for each category to be equally represented in the result. If we used "micro-averaging" and allowed the categories to be weighted in terms of their size, the "functional" category would have dominated the results[35]. For each classifier, we repeat this process five times. The average results for precision, recall, and the F_1 measure are displayed in Table 9. The standard deviation (SD) for the F_1 measure is presented as well. For the k-NN classifier, we report $k = 1$ as it had the best performance for k in our experirment.

Table 9. Stratified 10-Fold Cross Validation

Classifier	Precision	Recall	F_1	F_1 SD
Weighted Random	.047	.060	.053	.0042
50% Random	.044	.502	.081	.0016
Naïve Bayes	.227	.347	.274	.0043
SMO	.728	.544	.623	.0132
NFR Locator k-NN	.691	.456	.549	.0047

Although, the NFR Locator k-NN classifier did not perform as well as the SMO classifier, future work will utilize an ensemble classifier as described in Section 4.2.2. The k-NN classifier will be used to display closely related sentences. Comparing our classifier to the initial work performed by Cleland-Huang et al. [Cleland-Huang et al., 2007] for just the 15

[35] Excluding the random classifiers, all classifiers had F_1 values above .8 for the "functional" category.

documents in the PROMISE NFR Data Set, our F_1 measure was 0.382 and theirs was 0.239. Table 10 displays the detailed comparison results.

Table 10. Performance Comparison between Cleland-Huang, et. al and REDE

	[Cleland-Huang et al., 2007]				[Slankas & Williams, 2013a]		
NFR Category	**Precision**	**Recall**	**F_1**		**Precision**	**Recall**	**F_1**
Availability	0.11	0.89	0.20		0.47	0.70	0.56
Legal	0.16	0.70	0.26		0.40	0.40	0.40
Look & Feel	0.12	0.51	0.19		0.64	0.57	0.59
Maintainability	0.11	0.88	0.19		0.38	0.10	0.16
Operational	0.11	0.72	0.20		0.63	0.52	0.57
Performance	0.27	0.63	0.38		0.61	0.79	0.69
Scalability	0.11	0.72	0.19		Combined w/ performance		
Security	0.18	0.81	0.30		0.64	0.56	0.60
Usability	0.14	0.98	0.25		0.55	0.68	0.61

We also examined how well the “Combined SL” classifier found database related sentences. Using 10-fold cross-validation technique, we had a precision of .84, a recall of .82 and a F_1 score of .83.

RQ4: What sentence characteristics affect classifier performance?

Classifiers use any number of sentence characteristics (features) to make decisions. Sentence features include number of words, the words themselves, words represented as distance vectors, stems, lemmas, parts of speech, and known semantic entities. From a classification perspective, we need to utilize the features most useful and ignore other features providing little or no value. In Table 11, we present the results of using different word forms and stop words with the naïve Bayes and SMO classifiers. The “original” word form represents the word as it appeared in the sentence. "Lemma" is the lemma of the original word as produced by the Stanford NLP. "Porter" is the stem of the word produced by the Porter stemming algorithm [Porter, 1980]. "Casamayor" signifies the pre-processing steps Casamayor et al. performed [Casamayor et al., 2010]. Stop words are lists of words to exclude from use in the classifiers due to the commonality of the words. "Determiners" are "a", "an", and "the". "Frakes" [Frakes, 1992] contains 400 words and "Glasgow" contains 319 words[36]. By just eliminating the determiners, naïve Bayes' F_1 measure increased by 0.027 than with just the original sentence. Using additional stop words, lemmas, or stems had no more than 0.01 change in effectiveness. With SMO, effectiveness decreased by 0.019 using lemmas. Longer stop word lists appeared to have no effect for SMO. Each test was repeated five times with the average F_1 measure reported along with the standard deviation.

Table 11. Word Type and Stop Words

Model	Word Form	Stop Words	F_1	F_1 SD
Naïve Bayes	Original	Determiners	.291	.0022
Naïve Bayes	Porter	Determiners	.287	.0021
Naïve Bayes	Lemma	Determiners	.292	.0032
Naïve Bayes	Lemma	Frakes	.297	.0021
Naïve Bayes	Casamayor	Glasgow	.327	.0018
SMO	Original	Determiners	.603	.0044
SMO	Lemma	Determiners	.584	.0039
SMO	Lemma	Frakes	.586	.0042

[36] http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words

We examined the use of named entities (dates, durations, numbers, person, time, etc.) as sentence features. By representing the presence of a named entities as binary flags, we slightly increased the F_1 measure of the SMO classifier for the availability and reliability categories by 3.5%.

5.4 Discussion and Limitations

Several limitations exist for our work. As NFR Locator works on textual sentences, the process cannot extract information contained in images and tables. Within text documents, the process loses document structural information contained within the original Microsoft Word and Adobe PDF files. In several cases, there were lists of items that when examined as a whole would have been classified into a particular NFR category. To mitigate this limitation, we can directly parse document files in their native formats or utilize document structural analysis to discover titles, sections, lists and other document elements. Another limitation exists in that we do not know how our process and tool generalizes to other systems and domains.

This study presented a new process, NFR Locator, and a tool to assist analysts in effectively extracting relevant NFRs from a wide variety of documents. We demonstrated that NFRs exist in a wide variety of documents. In analyzing specific NFR categories, we found specific features unique to those categories that could be used by programs to identify other NFRs in the same category. We evaluated multiple classifiers to identify NFRs in specific categories and found SMO had the highest effectiveness. Our k-nearest neighbor classifier with a unique distance metric had a F_1 Measure of 0.54, outperforming in our experiments the optimal naïve Bayes classifier which had a F_1 Measure of 0.32. We also found that stop word lists beyond common determiners had no little performance effect.

6 Access Control Extraction Study

With over forty years of use and refinement, access control, often in the form of access control rules (ACRs), remains a significant and widely-used control mechanism for information security. ACRs regulate who can perform specific actions on specific resources within a software-intensive system and are considered a critical component to ensure both confidentiality and integrity [Samarati & Vimercati, 2001]. Despite access control's widespread usage, systems continue to have vulnerabilities of incorrectly implementing ACRs. In the 2011 CWE/SANS Top 25 Most Dangerous Software Errors [2011 CWE/SANS Top 25 Most Dangerous Software Errors, 2011], 30% of the errors directly relate to access control.

To meet security requirements, ACRs need to be complete, consistent, and correct [Zowghi & Gervasi, 2003]. Analysts must manage ACRs such that they can be evaluated for these three attributes. Existing project artifacts for systems such as use cases, requirements documents, and user manuals often capture the intended ACRs for the systems. However, manually sifting through these existing artifacts to extract the buried ACRs can be a tedious, time-consuming, and error-prone endeavor.

To address this issue, various researchers have proposed approaches to extracting ACRs from natural language (NL) artifacts. These approaches leverage techniques such as controlled natural languages (CNLs) [Brodie et al., 2006; Inglesant et al., 2008; Schwitter, 2010; Shi & Chadwick, 2011], manual analysis [Fernandez et al., 1997; He & Antón, 2009], and NLP [Xiao et al., 2012; Slankas & Williams, 2013b]. Each of these techniques has its own strengths and weaknesses. For example, manual analysis typically produces the most accurate results but at the cost of requiring more skilled human evaluators and time. Using CNLs produces comprehensive results as CNLs are designed to minimize the inherent ambiguity and complexity of NL [Schwitter, 2010]. However, using CNLs typically requires specialized authoring tools and conversion of pre-existing documents [Shi & Chadwick, 2011]. NLP techniques often require less manual effort than other techniques and can work on existing documents. However, NLP techniques tend to produce less accurate results than other techniques [Jurafsky & Martin, 2009].

Our research goal is to aid developers who implement ACRs by inferring ACRs from NL artifacts.

We define a novel approach, Access Control Rule Extraction (ACRE), to allow organizations to use existing, unconstrained NL texts such as requirements documents for inferring ACRs. ACRE combines NLP, information extraction (IE), and machine learning (ML) techniques. Internally, ACRE represents NL sentences as a type dependency parse graph [de Marneffe et al., 2006] that shows words as vertices and relationships between words as edges. Figure 24 shows the graph for the sentence "The user enters a grade for each student".

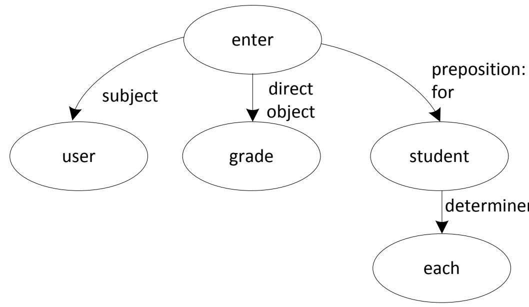


Figure 24. Type Dependency Graph

ACRE then searches these graphs for a basic pattern of *noun (subject)–verb–noun (direct object)*. ACRE creates a frequency table of all of the discovered subjects and direct objects. Any subject or direct object that occurs more times than the median count is assumed to a legitimate subject or resource (direct object) for the system. ACRE then searches all combinations of the discovered subjects and resources within the graphs. For each found combination, the approach extracts the minimal spanning tree (MST) and assumes that any verb within the tree corresponds to the action. ACRE adds the extracted MST to a set of valid ACR patterns. ACRE then expands the pattern set by allowing a subject or resource vertex within the pattern to match any word of the same part of speech in other sentence graphs (i.e., "wildcarded"). ACRE then searches all of the sentence graphs for instances of the ACR patterns. The found instances are assumed to represent ACRs, and the elements of the ACR are then extracted. ACRE iterates these steps, searching for additional patterns until no more patterns can be found in the sentences. To minimize incorrect patterns from being created and applied, ACRE constructs a naïve Bayes classifier to accept or reject patterns based upon domain-independent features found in other documents.

ACRE extends the various concepts in the Text2Policy approach proposed by Xiao et al. [Xiao et al., 2012]. However, ACRE incorporates IE and ML techniques to learn new ACR patterns automatically, whereas Text2Policy is limited by a fixed set of manually predefined ACR patterns.

This study makes the following main contributions:

- Bootstrapping mechanism to seed and iteratively discover ACR patterns in NL text.
- Approach and supporting tool (built upon the combination of NLP, IE, and ML techniques) to extract ACRs.
- Labeled data set of identified ACRs and sentences[37].
- Evaluation of ACRE against documents from systems in three domains: conference management, education, and healthcare.

The rest of this chapter is organized as follows: Section 6.1 presents our approach. Section 6.2 describes our research methodology. Section 6.3 presents our evaluation. We conclude this chapter with a discussion, a comparison with Text2Policy [Xiao et al., 2012], and overview of the limitations in Section 6.4.

6.1 Approach

This section details ACRE – our approach to extracting ACRs from NL text.

[37] https://sites.google.com/site/AccessControlRuleExtraction

6.1.1 Access Control Policy Representation

Internally, ACRE represents sentences with a dependency graph [de Marneffe et al., 2006] as depicted in Figure 25 for the sentence “a nurse can order a lab procedure for a patient.” In Section 6.1.4, we discuss how these graphs are produced. Each vertex represents a word from the sentence along with the word’s part of speech. In the figure, “NN” represents a noun, “VB” represents a verb, and “MD” represents a modal verb. Edges represent the grammatical relationship between two words. For instance, “nurse” functions as the nominal subject (nsubj) for “order”, and “lab procedure” is the direct object (dobj) to be ordered. The indicators correspond to the subject (“S”), action (“A”), resource (“R”) typically defined within an ACR. Dependency graphs can be considered as trees in most situations and are typically rooted by the sentence’s main verb. When conjunctions are present, vertices may have multiple parents, and thus the structure needs to be treated as a graph.

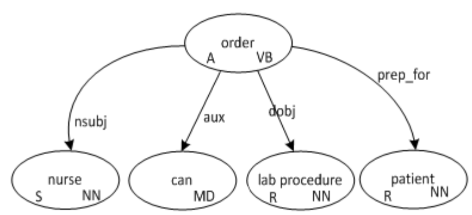


Figure 25. ACRE Sentence Representation

To represent an ACR, we use the pattern presented in Figure 26.

$$A(\{s\},\{a\},\{r\},[n],[l],\{c\},H,p)$$

Figure 26. ACR Representation

- A defines the overall ACR;
- s contains an order set of vertices that comprise the subject of a rule;
- a represents the action;
- r represents the resource;
- n contains the vertex representing negativity if required for the rule;
- l contains the indicating vertex if the rule should be limited to a particular subject s;
- c contains additional vertices required to provide context to a given action for a set of permissions;
- H represents the subgraph of a sentence's dependency graph that contains the vertices and necessary edges to connect all of the vertices listed in s, a, r, n, l, c; and
- p represents the permissions typically associated with an action.

We limit permissions to have the values of "create", "retrieve", "update", and "delete" as we are primarily concerned with controlling the ability to view and manipulate data in systems. We do use "execute" for permissions that do not map to one or more of the four preceding permissions. From the example in Figure 25**,** we define the two rules in Figure 27.

$A((nurse),(order),(lab\ procedure),(\ ),(\ ),(V{:}\ nurse,order,lab\ procedure;$
$E{:}\ (order,nurse,nsubj);(order,lab\ procedure,dobj)\ ),create)$
$A((nurse),(order),(patient),(\ ),(\ ),(V{:}\ nurse,order,patient;$
$E{:}\ (order,nurse,nsubj);(order,patient,prep\_for)\ ),read)$

Figure 27. Extracted ACRs

Situations exist in which not all of the ACR elements may be present within a single sentence. ACRE allows for only subjects to be missing. While users may manually identify ACRs with missing resources, ACRE has no support to find such patterns. Developers would be pointed to the surrounding sentences to finish defining the ACRs outside of the automated process.

6.1.2 Access Control Relation Extraction Process

The ACRE approach consists of five main steps:

1. Preprocess text documents
2. Produce dependency graphs
3. Classify each sentence as access control or not
4. Extract access control elements
5. Validate access control rules

6.1.3 Step 1: Preprocess Text Documents

In the approach, we first read the entire text into the tool built to support the approach. We separate the input into lines by either a carriage return or by periods at the end of sentences. Next, we apply a concise grammar [Slankas & Williams, 2013b] to label each token to a specific type (title, list element, list start, normal). By identifying specific types of sentences, we can increase the classification performance. Specifically, we found that section titles typically do not contain ACRs, and we design the mechanism of scoring sentence similarity to compare titles with only other titles within the document. We also found that list items require more context than just the specific item itself and process each list item combined with the start of the list.

6.1.4 Step 2: Produce Dependency Graphs

In our approach, after identifying the different sentence types, we parse each line (sentence) using the Stanford Natural Language Parser[38] and output a graph in the Stanford Type Dependency Representation (STDR) [de Marneffe et al., 2006]. While the Stanford Parser has several output formats available, we choose the STDR because it incorporates the sentence's syntactic information in a concise and usable format and captures the grammatical relationship between words. Dependency parse trees have become a critical component for many semantic role labeling systems as the NLP community evolved in the early 2000s from a strict constituent-based (i.e., shallow parsing) systems [Surdeanu & Johansson, 2008]. From the STDR generated by the parser, we create the sentence representation (SR) since we need to track additional attributes for the sentence and each word.

6.1.5 Step 3: Classify Each Sentence as Access Control or Not

Next, a k-NN classifier classifies whether a sentence contains an ACR. If the sentence does not express an ACR, we perform no further analysis on it. The k-NN classifier works by taking a majority vote of the existing classifications of the k nearest neighbors to the item under

[38] http://www-nlp.stanford.edu/software/corenlp.shtml

consideration. The classifier uses an adapted version of the Levenshtein distance [Levenshtein, 1966] as the distance metric. Rather than using the resulting number of edits to transform one string into another as the Levenshtein distance does, our adapted distance metric computes based on the number of word transformations to change one sentence into another.

Although other machine learning algorithms can provide similar performance to a k-NN classifier, the k-NN classifier provides easier interpretation of the results for analysts since they can see a ranked list of similar sentences and associated classifications.

Once we determine that the sentence contains an ACR, the user may review the determination and correct it if necessary within the tool.

Figure 28 shows a screenshot of the tool's user interface. The top table contains the document with individual columns to display the line number, sentence type, assigned classification, and completion status, assigned cluster (groups of similar sentences, optional functionality), and the sentence themselves. The dialog in the lower left allows users to review and manually enter or correct ACRs (discussed in the next section). The area in the lower right displays the SR.

6.1.6 Step 4: Extract Access Control Elements

Next, we need to extract the subject, action, and resource elements from the SR. We construct a relation extraction algorithm for the identification of ACR elements and subsequent extraction of the ACR. The algorithm follows a well-known bootstrapping technique [Jurafsky & Martin, 2009] but has been adapted specifically for ACR extraction. The basic concept is to start with a small, well-known set of ACR patterns and then expand those patterns to find other, closely related patterns.

To initialize the algorithm (presented in Figure 30), we seed a set of ten basic ACR patterns with each pattern consisting of just three vertices as shown in Figure 29. Each pattern is the same, except a different verb[39] is used for a "Specific Action". Wildcards are used to match any noun in sentences containing the pattern. We initially chose the words "create", "retrieve", "update", and "delete" because the words are commonly associated with viewing and manipulating data. We then examined the frequencies of all verbs within the test documentation and chose to add more verbs associated with data and appearing with high frequencies within the document. Based on the application domain or other documents, users may choose a different set of starting actions. From these patterns, we match all occurrences of the subjects and resources within the document along with their associated frequency counts. From the counts, we compute the median values for the subjects and resources. We then assume any word that occurs more than the median belongs to the application domain. Without a threshold, the potential for misidentified subjects and resources is much greater as any word matching the pattern would be accepted.

We then store these the subjects and resources in a list of known subjects and resources. From this listing, we then search the documents to see whether any subject exists along with any resource. For each sentence that does match the condition, we extract the dependency pattern between subject and resource vertices. We then assume that any verbs existing in that pattern are the actions. If more than one verb exists in the shortest path from the subject to the object, we combine the verbs but use the last appearing verb when defining permissions. In the

[39] create, retrieve, update, delete, edit, view, modify, enter, select

sentence, “the administrator chooses to create a new patient”, we combine “choose” and “create” to “choose create” for the action. The subject would be “administrator” and the object would be “patient”. We derive permissions for each pattern by finding the closest synonym (via WordNet[40]) that has an already defined permission.

Once we extract the pattern, we apply a series of transformations to extract additional patterns that may locate additional ACRs. Specifically, we transform patterns that have an active voice into passive voice and vice versa. We also transform the patterns to assume conjunctions may exist for two or more subjects, two or more actions, and two or more resources. To find additional subjects and resources, we apply wildcards to the identified subject or resource vertices. Only one the subject or only the resource vertex is wildcarded to minimize semantic drift in bringing in unrelated sentences or patterns to access control.

[40] http://wordnet.princeton.edu/

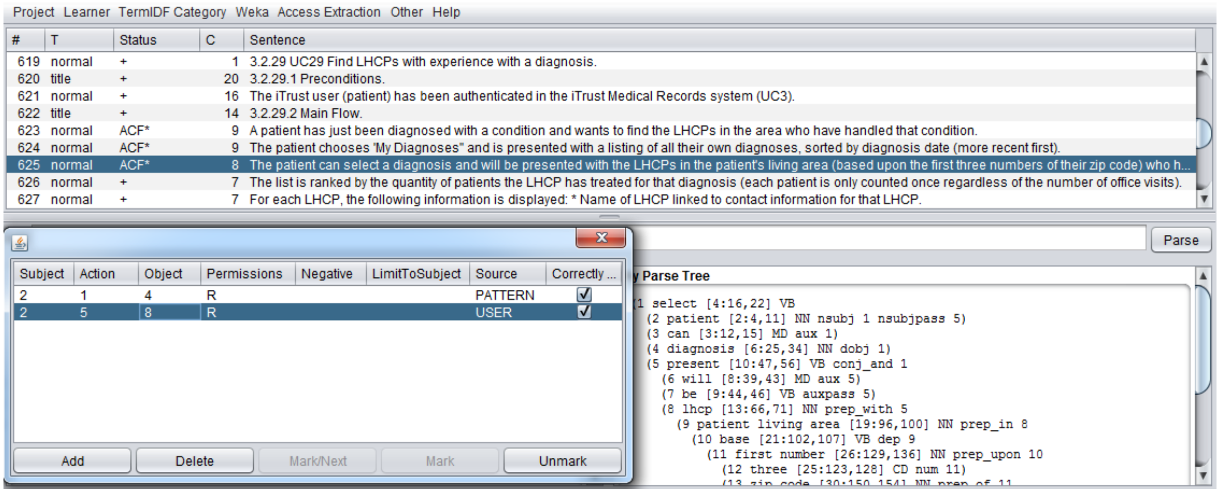


Figure 28. Tool Screenshot of Access Control Rule Extraction (ACRE)

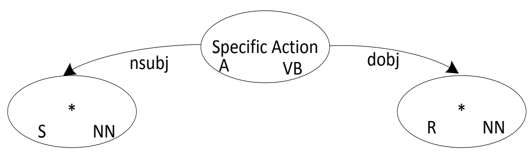


Figure 29. Basic ACR Seed Pattern

Furthermore, to ensure that the patterns are appropriate, we apply a naïve Bayes classifier to judge whether an identified pattern is appropriate. To detect the features used by the classifier, we use a forward, stepwise selection model [Han et al., 2011]. From this model, we found the pattern itself (the POS tags and relationships between vertices) to have the biggest influence. After that, the relationships between the resources and subjects had the next largest influence. After that, the identified parts of speech for the resources and subjects improved classification performance slightly. We did not use any further features (size, specific words, use of wildcards) since we found those features began to decrease the classifier's performance.

From the pattern set, we then search the documents for sentences matching one or more patterns. Once we find any match, we check to see whether other patterns match the same sentence. If more than one pattern matches and one pattern is contained within another pattern, we discard the former as the latter pattern provides a more specific match. Additionally, we check the matched sentences for any children vertices (of the matched pattern) that imply negativity or subject limitation (i.e., we check whether a relevant indicator exists just outside of the matched pattern).

We then store the extracted ACR in a list for validation and output to the user. Any new subjects or resources are added to the list of known subjects and resources. If newly discovered subjects or resources exist, then the algorithm iterates until no new items or patterns are discovered. Once the algorithm converges, the user may inject two additional patterns into the process to find more ACRs. Similar to the Basic ACR Seed Pattern in Figure 29, the first injected pattern allows any pronoun to be a subject within an ACR pattern. The second pattern searches for only actions and resources, leaving resolution of the subjects to another algorithm. The injection occurs after the algorithm initially converges to avoid spurious matches that would occur if the patterns were injected at the start of the algorithm. Additionally, the user may manually identify ACR patterns through identifying ACRs in the sentences. The information from these patterns is fed into the algorithm to search for additional extracted elements.

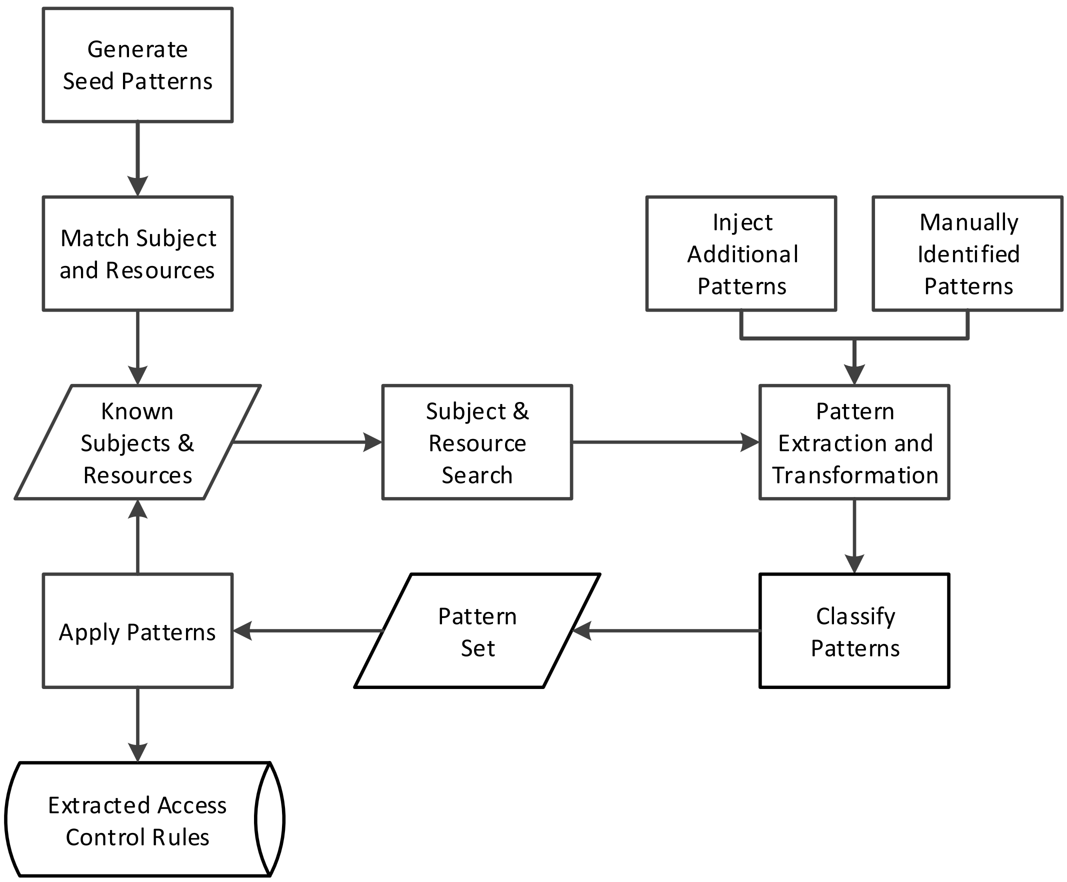


Figure 30. ACR Extraction Overview

6.1.7 Step 5: Validate Access Control Rules

In this step, the tool checks for coverage and conflicts within the extracted ACRs. Coverage is reported as the measurement for each subject as to the number of identified resources that it has ACRs identified. As we assume a default of no-access, 100% coverage is not required. However, low coverage values may indicate a need for further ACRs. Conflicts occur within ACRE when a specific subject has been both granted permission to a specific resource and restricted for the same permission on the same resource. Such conflicts may arise due to rule extraction in multiple locations or the use of a limiter to restrict access to a specific subject.

6.2 Research methodology

This section presents the methodology for collecting the study documents, creating the study oracle, and running the analysis.

6.2.1 Study Documents

As access control widely exists in numerous domains for software systems, we chose multiple domains for the evaluation. We selected documents from the electronic healthcare, educational, and conference management domains. Additionally, to compare results to prior work, we included Xiao et al.'s study documents [Xiao et al., 2012]. Table 12 lists the study documents.

For the electronic healthcare domain, we selected iTrust[41] [Meneely et al., 2011]. The requirements consist of 40 use cases plus additional non-functional requirements, constraints, and a glossary. We used two versions of this document. The first (iTrust for ACRE) was extracted directly from the project's wiki[42] while the second (iTrust for Text2Policy) was taken from the documentation[43] used by Xiao et al. [Xiao et al., 2012]. The first version more closely matches specifications used in industrial settings in that it has separate sections for the introduction, glossary, non-functional requirements, and other materials. The second version includes only the use cases themselves and simplifies complex sentences to be more consistent

41 http://agile.csc.ncsu.edu/iTrust
42 http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=requirements
43 https://sites.google.com/site/text2policy/

with the rules of their parser [Xiao et al., 2012]. For the educational domain, we took the eight use cases from the IBM Course Registration System [IBM, 2004] used in a prior research study [Vidya Sagar & Abirami, 2014]. For the conference management system, we used documents from CyberChair[44] [Stadt, 2012], which has been used by over 475 different conferences and workshops. We also included a combined document of 114 sentences with ACRs that Xiao et al. [Xiao et al., 2012] collected.

6.2.2 Study Oracle

To train the classifiers used within ACRE and to evaluate its performance, we constructed a study oracle. To create the oracle, the author of this dissertation followed the following steps:

1. Convert the document into a "text-only" format.
2. Correct the resulting text file to account for improper line breaks and other formatting issues.
3. Import the document into the ACRE tool.
4. Mark each sentence as to whether or not it is an ACR sentence.

After the classification was complete, we validated the classification through multiple ways. First, we created clusters of related sentences. We then compared the classifications within each cluster and investigated further those sentences that did not have the same classification as other sentences in the group. Additionally, as we classified each sentence, we had access to the neighbors contained within the k-NN classifier. This way allowed for more

[44] http://www.borbala.com/cyberchair/

rapid manual classification by suggesting initial classification that we could then verify or correct as deemed necessary. Additionally, any discrepancies in the predicted classification could be easily traced back to the source sentences.

We then had two other researchers manually classify each document. We then computed the Fleiss' kappa [Fleiss, 1971] for each document (see Table 12). From Landis' and Koch's guidelines [Landis & Koch, 1977], scores between 0.41 and 0.60 indicate moderate agreement while those between 0.61 and 0.80 indicate substantial agreement. 0.81 to 1.00 is considered almost perfect agreement. The three individuals then discussed the differences and collectively choose the appropriate classification for each sentence with disagreements.

Table 12. Study Documents

Document	Abbreviation	Domain	Number of Sentences	Number of ACR Sentences	Number of ACRs	Fleiss' Kappa
iTrust for ACRE	iTrust_acre	Healthcare	1160	550	2274	0.58
iTrust for Text2Policy	iTrust_t2p	Healthcare	471	418[45]	1070	0.73
IBM Course Management	IBM_cm	Education	401	169	375	0.82
CyberChair	Cyberchair	Conference Mgmt	303	139	386	0.71
Collected ACP Documents	Collected	Multiple	142	114	258	n/a

The author of this dissertation then manually identified each ACR within the documents. Once those rules were identified, another researcher then validated a random sampling of 20% of the ACRs from the document "iTrust for ACRE". To ensure that the

[45] Xiao et al. [Xiao et al., 2012] reported 448 sentences with 117 containing access control for the same document (https://sites.google.com/site/text2policy/).

reviewer diligently examined each set of tuples, we injected five “mistakes”. The reviewer made 29 corrections out of the 359 ACRs and found 80% of the injected “mistakes”.

To create the initial classifications of the five documents, the author of this dissertation spent six hours to classify the 2,477 sentences as access control or not. Identifying the specific access tuples took additional 62 hours for the five documents.

6.2.3 Study Procedure

Once the oracle has been created, we ran the ACRE Tool to produce several reports to pull out details to examine properties of sentences with access control. To evaluate how well we identify sentences with ACRs, we ran the k-NN classifier on a combined document of the iTrust ACRE requirements, IBM Course Management, and CyberChair. We also report results on each of the five documents being individually classified. Each document was tested with stratified n-fold cross-validation and computed the precision, recall, and F_1 measure. With the n-fold cross-validation, data is randomly partitioned into n folds based upon each fold of approximately equal size and equal response classification. For each fold, the classifier is trained on the remaining folds and then the contents of the fold are used to test the classifier. The n results are then averaged to produce a single result. We follow Han et al.'s recommendation [Han et al., 2011] and use 10 as the value for n as this value helps produce relatively low bias and variance. The cross-validation ensures that all sentences are used for training and that each sentence is tested just once. We also report the results of using the individual documents as folds within the combined document. As it is not necessarily feasible to have a trained classifier readily available, we also evaluate the amount of work necessary to classify a document from scratch in terms of the performance when classifying the rest of the document.

In the final phase of the study, we created the naïve Bayes classifier using a document-fold way (i.e., training the classifier with all of the documents being evaluated except for one and then repeating the experiment until all documents have been tested). We then ran the ACRE process to extract the ACRs. The extracted ACRs were compared against the study oracle to determine the accuracy of the results.

6.3 Evaluation

We present and answer our research questions in this section.

6.3.1 Access Control Sentence Analysis

RQ2.1: What patterns exist among ACR sentences?

For this research question, we explore different patterns that ACR sentences have. Xiao et al. [Xiao et al., 2012] reported four common sentence patterns for sentences with ACRs (see Table 17). In Table 13, we present the number of times we found these sentence patterns within our study documents. As we did not have access to their tool, we identified how often ACR sentences met one of their four patterns based on when certain conditions were met. For their "Modal Verb in Main Verb Group" pattern, we checked whether a modal verb existed in an ACR sentence. For the "Passive Voice" pattern, we checked whether the sentence was in the passive voice with "allow" and "to" appearing in the sentence. For the "access expression" pattern, we checked whether an ACR sentence had some form of "access" within it. Finally, for the "Ability Expression" pattern, we whether there existed some form of "access" within the sentence. Xiao et al. found that 85% of the ACRs followed these patterns in the document of iTrust for Text2Policy. The updated results derived by us showed that these patterns cover only 57% of the ACR sentences that we identified.

Table 13. Metrics by Document

Metric	iTrust_acre	iTrust_t2p	IBM CM	CyberChair	Collected
Text2Policy Pattern – Modal Verb	210	130	46	71	93
Text2Policy Pattern – Passive voice w/ to Infinitive	66	21	10	39	9
Text2Policy Pattern – Access Expression	32	7	5	1	18
Text2Policy Pattern – Ability Expression	45	21	14	11	3
Number of sentences with multiple types of ACRs	383	146	77	105	36
Number of patterns appearing once or twice	680	173	162	184	97
ACRs with ambiguous subjects (e.g. "system", "user", etc.)	193	119	139	1	13
ACRs with blank subjects	557	206	29	187	5
ACRs with pronouns as subjects	109	28	5	11	11
ACRs with ambiguous objects (e.g., entry, list, name,etc.)	422	228	45	47	34

We also examined how frequently extracted ACRs pattern appear. Table 14 shows the top ACR patterns for all documents. The most common pattern (equivalent to ACRE's basic seed pattern) occurs approximately 14% for all ACRs. We also found a high occurrence of ACR patterns with a preposition in them.

Table 14. Top ACR Patterns

Pattern	Num. of Occurrences	
`(VB root(NN nsubj)(NN dobj))`	465	(14.1%)
`(VB root(NN nsubjpass))`	122	(3.7%)
`(VB root(NN nsubj)(NN prep))`	116	(3.5%)
`(VB root(NN dobj))`	72	(2.2%)
`(VB root(NN prep_%))`	63	(1.9%)

We then examined how frequently multiple types of ACRs could occur in one sentence. For example, S5 in Table 3 has multiple forms of ACRs to be extracted. We found multiple ACRs in approximately 33% of all ACR sentences. Multiple forms of access being tied to the same action can also cause issues with permissions since different permissions may be necessary.

RQ2.2: How frequently do different forms of ambiguity occur in ACR sentences?

We also examined how frequently different forms of ambiguity (i.e., when a subject or object is not clearly defined in an extracted ACR), occur in the documents. We found that pronouns occurred as subjects in only 3.2% of the identified ACRs. "System" and "user" occurred as subjects in 11% of all ACRs. We also found that a subject could not be explicitly identified as a subject in 17.3% of the ACRs. We found very few instances of pronouns as objects. However, we did find that ambiguous terms (e.g., "list", "name", "record") occurred 21.5% in all of the ACRs. From these observations, a strong need exists to track actors from one sentence to another sentence to complete blank subjects or replace "system" and "user" occurrences. Similarly, an effective approach needs to be able to resolve ambiguity issues with objects. In most cases, this ambiguity resolution can be done by examining prepositions that relate directly to the ambiguous word. We also found that missing objects within ACRs only occurred in four times in all of the documents.

6.3.2 Identification of ACR Sentences

RQ2.3: How effectively does ACRE detect ACR sentences in terms of precision and recall?

Table 15 presents the results of running the classifier against each document individually with a ten-fold cross validation. We then evaluate the documents of iTrust for

ACRE, IBM Course Management, CyberChair, and Collected ACP using both a 10-fold cross-validation and a document-fold validation. For general use, the document-fold validation would most represent the performance as the user would use an existing classifier against a new document. For this result, ACRE had a precision of 81% and a recall of 65%. The scores for the documents of iTrust for Text2Policy and Collected ACP were abnormally high due to the high concentration of ACR sentences.

We also evaluated how ACRE would perform classifying sentences in which the user did not have a pre-trained classifier available. Figure 31 shows the classification performance for the documents based on what percentage of the sentences in a document has been correctly classified by a user. For the document of iTrust for ACRE, the F_1 score is already above 80% after just 25% of the sentences in the document have been classified as having ACRs or not. For the sentences in the document of IBM Course Management, the performance shows two irregularities where performance decreases (at 10% and 35% completion). Based on examining the document, the first situation contains the glossary. The sentences contained here tend to have high similarity to other sentences later in the document, but do not have any corresponding user-based actions. As such, they do contain ACRs. They then cause false negatives as later sentences are parsed. In the second situation, the User Login Use Case was evaluated. As these sentences deal primarily with authentication, they do not contain ACRs. The documents for iTrust for Text2Policy and Collected ACP were not placed into the graph since the overall percentages of ACR sentences are so high that no practical change to the classification performance exists.

Table 15. Identification of ACR Sentences

Document	Precision	Recall	F_1
iTrust for Text2Policy	96%	99%	98%
iTrust for ACRE	90%	86%	88%
IBM Course Management	83%	92%	87%
CyberChair	63%	64%	64%
Collected ACP	83%	96%	89%
10-fold validation	81%	84%	83%
Document-fold validation	81%	65%	72%

6.3.3 Access Control Rule (ACR) Extraction

RQ2.4: How effectively can the subject, action, and resource elements of ACRs be extracted from ACR sentences?

Table 16 presents the results of the bootstrapping algorithm on each of the documents. We train the naïve Bayes classifier by using the identified patterns in the other documents. The document of iTrust for Text2Policy is not used for training. Also, neither of the iTrust documents are used in the training set when evaluating one of those documents.

Table 16. ACR Extraction Results

Document	Precision	Recall	F_1
iTrust for Text2Policy	80%	75%	77%
iTrust for ACRE	75%	60%	67%
IBM Course Management	81%	62%	70%
CyberChair	75%	30%	43%
Collected ACP	68%	18%	29%

ACRE performed best on "iTrust for Text2Policy". This document contained a number of subjects and resources repeated throughout the document. As complex sentences were split into multiple, simpler sentences, there were less complex patterns to discover. ACRE performed worst on the document of Collected ACP. This document contains ACR sentences extracted from 19 sources. As little repetition exists in sentence structure, subjects, and resources, ACRE performs poorly in finding the initial set of known subjects and resources as well as in expanding the patterns. For the "Collected ACP" document, if we start the algorithm with a known list of resources and subjects (which can be obtained from a glossary or similar section), ACRE has a precision of 85%, a recall of 70%, and a F_1 of 77%. The document of CyberChair also demonstrated a very low recall. As this document was a combination of an introductory page as well as a conference paper, little repetition exists in the sentence structure. Bootstrapping from a known list of subjects and resources only slightly improves the recall to 39% as 117 of the missing 240 ACRs were covered by patterns with more than three vertices containing missing subjects.

When examining which features to use for the naïve Bayes classifier, we found that the patterns themselves produced 80% of the possible performance. The relationships between the subjects and the objects then made the next most noteworthy performance improvements. After these three features, adding more features into the classifier reduced the classifier's performance.

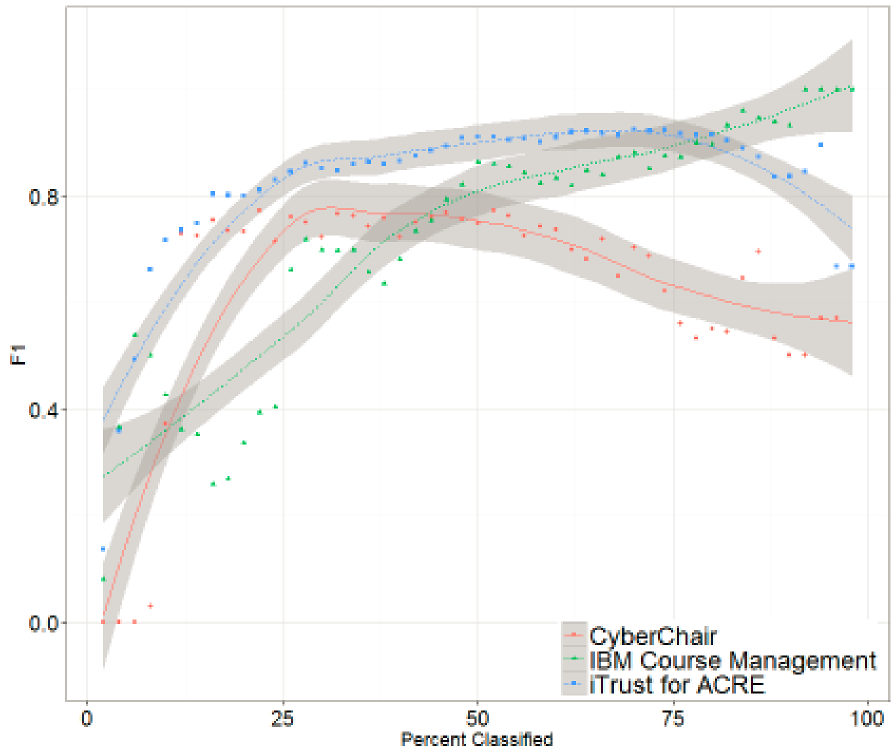


Figure 31. Classification Performance (F_1) by Completion %

6.4 Discussion

This section discusses the limitations of the approach and this study, a detailed comparison with Text2Policy, and our conclusion.

6.4.1 Limitations

Since our ACRE approach uses NLP techniques, ACRE and its supporting tool cannot extract information contained in images. With regards to ACRs, the bootstrapping mechanism does not take into account the presence of contextual information or conditions that may affect the generated ACRs. However, the user can manually enter such information. The ACRE approach also requires that subjects and resources be identified as nouns and actions as verbs unless the user manually enters a rule. The approach also assumes that all necessary information for an ACR is contained within the same sentence. However, it is feasible for elements of an ACR to exist in surrounding sentences of the corresponding ACR sentence.

Our approach currently does not handle resolution issues. These issues occur when a pronoun or generic term such as “system” or “data” is used in place of a descriptive term. In future work, we will incorporate resolution techniques to address such issues. We will also investigate how to search for larger ACR patterns in which the subject is missing.

The threat to external validity is mainly due to the representativeness of the subjects. To reduce the threat, we evaluate our approach on documents from three domains. However, to further reduce the threat, additional evaluation needs to occur across multiple domains and applications. We surmise that the approach can work for other narrative-based texts, but “task/step-oriented” documents such as test scripts and user manuals would be less effective as the subject is often assumed throughout a series of steps. For such documents, we would need to study the use of “action–resource” pairs to generate patterns.

ACRE does require manual effort to set up the classifiers. We were able to identify whether or not a sentence contained an ACR at an average rate around one sentence per 9 seconds. However, this identification process is just making a simple yes or no decision, and we had optimized an interface such that the user only had to press a single button per sentence. Considerably more effort is required to identify each ACR within a sentence. We identified the ACRs at an average rate of one ACR per 50 seconds. However, we found that the naïve Bayes classifier for patterns could be used effectively on our documents and domains. In future work, we will study ways to reduce the workload in human annotation for NL classification tasks.

6.4.2 Comparison with Text2Policy

Xiao et al. proposed Text2Policy [Xiao et al., 2012], for automated extraction of ACRs. Text2Policy accepts use cases in NL text as input and outputs the extracted ACRs in the eXtensible Access Control Markup Language (XACML) format. Text2Policy first uses shallow parsing techniques with finite state transducers to annotate sentences with "phrases, clauses, and grammatical functions of phrases such as subject, main verb, and object." From those annotations, Text2Policy matches a sentence into one of four possible access control patterns (Table 17). If such a match can be made, Text2Policy classifies the sentence as an ACR.

Table 17. Text2Policy - Semantic Role Patterns in Access Control Sentences

Semantic Pattern	Examples
Model Verb in Main Verb Group	A nurse$_{\text{[subject]}}$ can view$_{\text{[action]}}$ the patient's records$_{\text{[resource]}}$. An admin$_{\text{[subject]}}$ should not update$_{\text{[action]}}$ the patient's records$_{\text{[resource]}}$.
Passive Voice followed by To-infinitive Phrase	A nurse$_{\text{[subject]}}$ is disallowed to update$_{\text{[action]}}$ the patient's records$_{\text{[resource]}}$. A nurse$_{\text{[subject]}}$ is allowed to view$_{\text{[action]}}$ the patient's records$_{\text{[resource]}}$.
Access Expression	A nurse$_{\text{[subject]}}$ has read$_{\text{[action]}}$ access to the patient's records$_{\text{[resource]}}$. A patient's records$_{\text{[resource]}}$ is accessible$_{\text{[action]}}$ to a nurse$_{\text{[subject]}}$.
Ability Expression	A nurse$_{\text{[subject]}}$ is able to read$_{\text{[action]}}$ patient's records$_{\text{[resource]}}$. A nurse$_{\text{[subject]}}$ has the ability to read$_{\text{[action]}}$ access to the patient's records$_{\text{[resource]}}$.

Adapted from [Xiao et al., 2012]

Once Text2Policy classifies sentences as ACRs, Text2Policy uses the annotated portions of the sentences to extract the subject, action, and object from the sentence.

Text2Policy uses a pre-defined domain dictionary to associate the action with specific semantic classes such as "UPDATE" and "DELETE" to determine appropriate permissions for the ACR.

While ACRE and Text2Policy both target the problem of ACR extraction from NL, they differ in their basic approaches. Fundamentally, the differences between ACRE and Text2Policy can be summarized by an inductive versus a deductive approach. ACRE applies inductive reasoning to find and extract ACRs. Text2Policy applies deductive reasoning based on existing rules (sentence patterns) to find and extract ACRs.

ACRE identifies sentences containing ACRs through a supervised learning approach. As such, the approach requires a labeled dataset similar in structure and content to the document being analyzed. Text2Policy identifies sentences containing ACRs based upon whether or not the shallow parser can parse the sentence into one of its four required patterns. In this regard, Text2Policy has an advantage since Text2Policy does not require labeled data set to train a classifier. However, Text2Policy can miss ACRs that do not follow one of its four patterns. In our analysis of the documents, we found that only 34.4% of the identified ACR sentences followed one of Text2Policy's patterns. Additionally, Text2Policy's NL parser required splitting longer sentences as the parser could not handle complicated sentence structures.

Text2Policy can extract only one ACR per sentence. For example, ACRE would extract two rules from S5 in Table 3 (*instructor, enter*$_{create}$*, grade*) and (*instructor, enter*$_{read}$*, student*). Text2Policy would find only the former ACR. From our evaluation in Section 6.3.1, we found that sentences containing multiple ACRs account for 33% of the examined sentences.

6.4.3 Conclusion

In this study, we have presented an approach, ACRE, to assist developers in automatically extracting ACRs from NL documents. ACRE provides a way for developers to generate an initial set of ACRs quickly with traceability back to the originating sentences. Developers can apply the approach to detect conflicts in generated rules as well as evaluating the coverage of the generated rules to the identified subjects and resources. We demonstrated how effectively a bootstrapping algorithm can extract rules from a very small initial set of patterns.

We found that ACRs exist in 47% of the examined sentences. Our ACRE approach correctly identifies ACR sentences with a precision of 81% and recall of 65%. The approach extracts ACRs from those identified ACR sentences with an average precision of 76% and an average recall of 49%. Due to the bootstrapping mechanism, the approach works better with a longer document with similar sentences structures, subjects, and resources repeated throughout the document.

7 Database Model Extraction Study

One of the limitations of the prior work in extracting access control policies from natural language product artifacts is the lack of implementation of those policies directly into a system. Developers need solutions that can be implemented efficiently in the system's environment. In Xiao et al.'s paper [Xiao et al., 2012], the Text2Policy algorithm generated eXtensible Access Control Markup Language (XACML) policies as its output. However, a system would need to implement an XACML-based policy control point. The software would need to be altered such that access checks would utilize policy control. The developers would be responsible for mapping between terms used in the code base and the subject, predicates, and resources defined within the XACAML policy. As of March 2015, no major DBMS providers provide XACML-based policy control points within their database product nor is XACML part of the SQL Standard [ISO Standards Committee JTC 1/SC 32, 2011].

El-Azia and Kannan [El-aziz & Kannan, 2014] demonstrated an approach by using database triggers, relational tables, and modified queries. Database triggers are only valid for insert, update, and delete queries [ISO Standards Committee JTC 1/SC 32, 2011]. Further, as there method would need to be called explicitly, developers would need to alter their code to performance access control check first and then retrieve the existing record. A two-step process still leaves the access control implementation at only the application layer as the first step can be bypassed. Utilizing database views to join the original query and the access control query into a single query is possible, but would require writing a separate view for every possible select query. As the view would also have been created from multiple tables, the view could not have been used for update, insert, or delete queries. As such, database triggers would be required.

The ideal solution would implement additional access control mechanisms in the persistence layer utilizing the well-established and standardized methods already built-in into most relational database systems [ISO Standards Committee JTC 1/SC 32, 2011]. We define an approach to implementing RBAC for systems at the persistence layer. Having access control implemented in both the application and persistence layers strongly supports a defense in depth strategy. The approach mitigates potential loss or misuse of data when individuals bypass application controls or the ACRs are inconsistently implemented.

Our research goal is to improve security and compliance by ensuring access control rules explicitly and implicitly defined within unconstrained natural language product artifacts are appropriately enforced within a system's relational database.

We extend our prior studies to specify a tool-based process, which will call Role Extraction and Database Enforcement (REDE) to allow organizations to utilize existing, unconstrained natural language product artifacts to generate RBAC policies for databases. To assist users in following this process, we implemented an open-source tool[46]. REDE extends Access Control Rule Extraction (ACRE) with a novel approach to detect and extract database model elements (DMEs) from product artifacts based on natural language processing, information extraction, and machine learning techniques. REDE also includes steps for mapping the design found in the product artifacts to the physical database schema and

[46] https://github.com/RealsearchGroup/REDE

implementing a set of access control rules for an RBAC policy within a database. Chapter 4 provides the complete process details.

As this study examines the entire REDE process, we revisit prior research questions in addition to specifying new research questions for this study:

RQ3.1: What patterns exist among ACR sentences?

RQ3.2: What patterns exist among DME sentences?

RQ3.3: How effectively does REDE detect ACR sentences?

RQ3.4: How effectively does REDE detect DME sentences?

RQ3.5: How effectively can the subject, action, and resource elements of ACRs be extracted from ACR sentences?

RQ3.6: How effectively can the entities, entity-attributes, and relationship elements be extracted from DME sentences?

RQ3.7: How effectively are extracted DMEs mapped to a physical database schema?

RQ3.8: How effectively does REDE implement access control within a system's relation database?

This study makes the following contributions:

- Bootstrapping mechanism to seed and iteratively discover database model patterns in NL text.
- Approach and supporting tool (built upon the combination of NLP, IE, and ML techniques) to extract DMEs (entities, entity-attributes, and relationships).

- Labeled data set of identified ACRs, DMEs, and sentences[47].

The rest of this chapter is organized as follows: Section 7.1 refers readers to Chapter 4 for a complete presentation of our approach. Section 7.2 describes our research methodology. Section 7.3 presents our evaluation.

7.1 Approach

As this was the final study for this dissertation, the approach has been documented in Chapter 4.

7.2 Research methodology

This section presents the methodology for collecting the study documents, creating the study oracle, and running the analysis.

7.2.1 Study Artifacts

To evaluate REDE, we need to evaluate a sufficiently complex system that has documentation, source code, and other product artifacts. The system should be in production use to demonstrate both maturity and stability of the code base. For our past studies, we examined the documentation in a variety of fields: electronic healthcare records, educational, conference management, and finance. After evaluating multiple systems, we chose the Open Conference Systems(OCS) [Public Knowledge Project, 2015] as a case study to evaluate REDE. OCS was first released in 2000 as an open-source project and has been continually maintained since that time. Table 18 presents a variety of information for OCS. In addition, to

[47] https://github.com/RealsearchGroup/REDE/tree/master/data

OCS having sufficient size and maturity, we can also use our existing analysis of the Cyberchair Conference Management [Stadt, 2012] to function as training data for the classifiers used within the REDE.

Table 18. Open Conference System Meta Details

System	Open Conference System
Version	2.3.6, released May 28th, 2014
Language	PHP
Supported DBMSs	MySQL, PostgreSQL
Architecture	Web-based application
Number of PHP files	1557
Number lines in PHP files	22198
Number of application defined roles	7
Number of database tables	52
Number of fields in database tables	369

7.2.2 Study Oracle

To train the classifiers used within REDE and to evaluate its performance, we constructed a study oracle. To create the oracle, the author of this dissertation followed the following steps:

1. Convert the document into a “text-only” format.
2. Correct the resulting text file to account for improper line breaks and other formatting issues.
3. Import the document into the ACRE tool.
4. Mark each sentence as to whether or not it is an ACR sentence.
5. Mark each sentence as to whether or not it is a DME sentence.

For a sentence to contain an ACR, the subject (whether explicitly or implicitly) had to perform some activity with data stored within the persistence layer. While it is feasible and appropriate for access control to be defined for other situations (e.g., the ability of a user to go to a specific web page in an application), such access control was beyond the scope of this project. For a sentence to contain a DME, we examined whether or not if was feasible for the resource being viewed or manipulated in some way to be stored within the persistence layer.

After the classification was complete, we validated the classification through multiple ways. First, we created clusters of related sentences. We then compared the classifications within each cluster and investigated further those sentences that did not have the same classification as other sentences in the group. Additionally, as we classified each sentence, we had access to the neighbors contained within the k-NN classifier. This way allowed for more rapid manual classification by suggesting initial classification that we could then verify or correct as deemed necessary. Additionally, any discrepancies in the predicted classification could be easily traced back to the source sentences.

Table 19. Open Conference Systems User Manual Metrics

Metric	Count
Number of sentences	708
Number of ACR sentences	327
Number of ACRs	630
Number of DME sentences	329
Number of DMEs	1002
Number of Entity DMEs	748 (287 unique)
Number of Entity-Attribute DMEs	99 (75 unique)
Number of Relationship DMEs	155 (82 unique)
Number of DME sentences with no ACRs	2

The author of this dissertation then manually identified each ACR and DME within the documents. For ACRs, we examined how the subject or actor within a sentence views or other manipulate potential resources within the document. For DMEs, we considered a word within the sentence as an entity if it could be stored in the persistence layer, and the word was not “part of” another word within the same sentence. If a word was part of another word and there could only be one instance of the first word, we marked the first word as an attribute of the second word, which would have also been marked as an entity. If that first word could have multiple instances, then we considered the two words in an aggregation relationship. If one word was a specified type of another word, we considered that an inheritance/specialization relationship. Finally, any transitive verb that tied together two entities would form an association relationship. Once those rules were identified, two other researchers validated a random sampling of 50 sentences from the OCS User Manual. To ensure that the reviewers diligently examined each set of tuples, we injected five “mistakes”. The first reviewer made six corrections within the 50 sentences and found three of the injected “mistakes”. The second reviewer made 13 corrections (eight of these corrections were permission settings) and found the same three injected mistakes as the first reviewer. The author of the dissertation and the two other reviewers were in complete agreement on 32 of the 50 sentences (the two other reviewers had three matching corrections, but they missed two injected errors: $50 - 13 - 6 + 3 - 2 = 32$).

The author of this dissertation spent two hours to classify the 708 sentences as access control or not as well whether or not the sentence contained database model elements. Identifying the specific ACRs and DMEs took approximately 58 hours.

7.2.3 Study Procedure

Once the oracle has been created, we ran the REDE Tool to produce several reports to pull out details to examine properties of sentences with access control. To evaluate how well we identify sentences with ACRs, we ran the k-NN classifier on a combined document of the iTrust ACRE requirements, IBM Course Management, and CyberChair. We also report results on each of the five documents being individually classified. Each document was tested with stratified n-fold cross-validation and computed the precision, recall, and F_1 measure. With the n-fold cross-validation, data is randomly partitioned into n folds based upon each fold of approximately equal size and equal response classification. For each fold, the classifier is trained on the remaining folds and then the contents of the fold are used to test the classifier. The n results are then averaged to produce a single result. We follow Han et al.'s recommendation [Han et al., 2011] and use 10 as the value for n as this value helps produce relatively low bias and variance. The cross-validation ensures that all sentences are used for training and that each sentence is tested just once. We also report the results of using the individual documents as folds within the combined document. As it is not necessarily feasible to have a trained classifier readily available, we also evaluate the amount of work necessary to classify a document from scratch in terms of the performance when classifying the rest of the document.

In the final phase of the study, we created the naïve Bayes classifier using the documents previously analyzed in the second study. We then ran the REDE process to extract the ACRs for OCS. The extracted ACRs were compared against the study oracle to determine the accuracy of the results. Similar, we performed the same steps to perform the DME extraction.

7.3 Evaluation

7.3.1 Sentence Analysis

RQ3.1: What patterns exist among ACR sentences?

For this research question, we examine the different patterns ACR sentences have. Xiao et al. [Xiao et al., 2012] reported four common sentence patterns for sentences with ACRs (see Table 17). In Table 20, we report the number of times we found these sentence patterns within our study documents. As we did not have access to their tool, we identified how often ACR sentences met one of their four patterns based on when certain conditions were met. For their "Modal Verb in Main Verb Group" pattern, we checked whether a modal verb existed in an ACR sentence. For the "Passive Voice" pattern, we checked whether the sentence was in passive voice with "allow" and "to" appearing in the sentence. For the "access expression" pattern, we checked whether an ACR sentence had some form of "access" within it. Finally, for the "Ability Expression" pattern, we whether there existed some form of "access" within the sentence. Xiao et al. found that 85% of the ACRs followed these patterns in the document of iTrust for Text2Policy, which is very much line with the OCS user manual in which the patterns occurred 84.7%. In the previous chapter, we found the patterns had only occurred 57% of the time in those documents we examined. Given the number of ACRs with blank subjects and pronouns as subjects (71.9% of all ACRs), solutions to extract access ACRS must implement resolution methods when processing similar types of documents. Similarly as 18% of all ACRs have ambiguous resources, resolution must also occur in those situations.

Table 20. OCS Document Metrics

Metric	Count
Number of ACR Sentences	327
Number of ACRs	630
Text2Policy Pattern – Modal Verb	203
Text2Policy Pattern – Passive voice w/ to Infinitive	26
Text2Policy Pattern – Access Expression	15
Text2Policy Pattern – Ability Expression	33
Number of sentences with multiple types of ACRs	136
Number of patterns appearing once or twice	318
ACRs with ambiguous subjects (e.g. "system", "user", etc.)	9
ACRs with blank subjects	228
ACRs with pronouns as subjects	225
ACRs with ambiguous resources (e.g., entry, list, name,etc.)	115

We also examined how frequently extracted ACRs pattern appear. Table 14 shows the top ACR patterns for all documents. The most common pattern (equivalent to REDE's basic seed pattern) occurs approximately 15.1% for all ACRs, just slightly higher than the documents from the previous study. As the document we examined was a user manual, the pronoun "you" appeared quite often; this is reflected in the second most pattern in which a pronoun served as the subject for a sentence.

Table 21. Top ACR Patterns for OCS

Pattern	Num. of Occurrences	
`(VB root(NN nsubj)(NN dobj))`	95	(15.1%)
`(VB root(PRP nsubj)(NN dobj))`	57	(9.0%)
`(VB root(NN nsubj)(NN dobj))`	26	(4.1%)
`(VB root(NN dobj))`	21	(3.3%)
`(VB root(NN nsubjpass))`	18	(2.9%)

RQ3.2: What patterns exist among DME sentences?

For this question, we first performed some exploratory data analysis on the DME patterns manually identified. Figure 32 presents a histogram of the DME pattern sizes. As no patterns had more than 10 vertices (and only six patterns were larger than seven nodes), a reasonable assumption can be made to place an upper limit on pattern sizes. Within REDE, this limit can either be implemented as a hadrd-coded rule, or the pattern size could be a factor used in the naïve Bayes classifier for evaluating patterns.

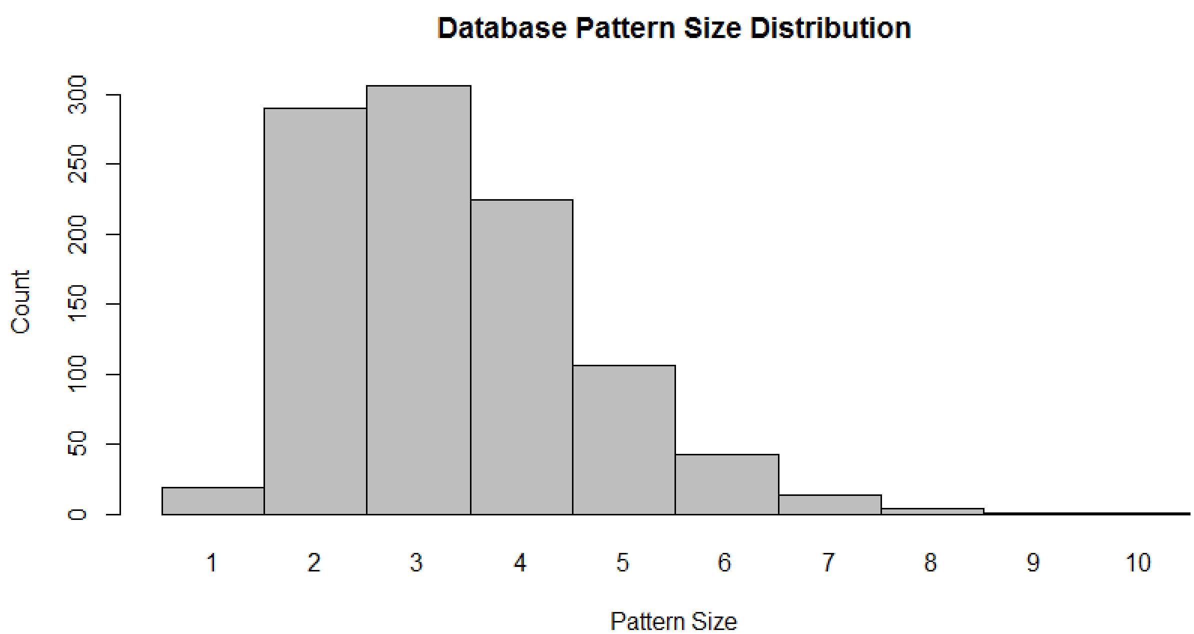


Figure 32. Database Pattern Size Distribution

Figure 33 presents a histogram of the maximum distance between words in the originating sentence for the graph patterns. While the histogram shows approximately half

(322 out of 630) of ACRs have a maximum distance between words of 4 or less. The right-tail shows one of the advantages of using the STDR graph as, even though, the words appeared far apart in the text; the resulting graph sizes were comparatively small.

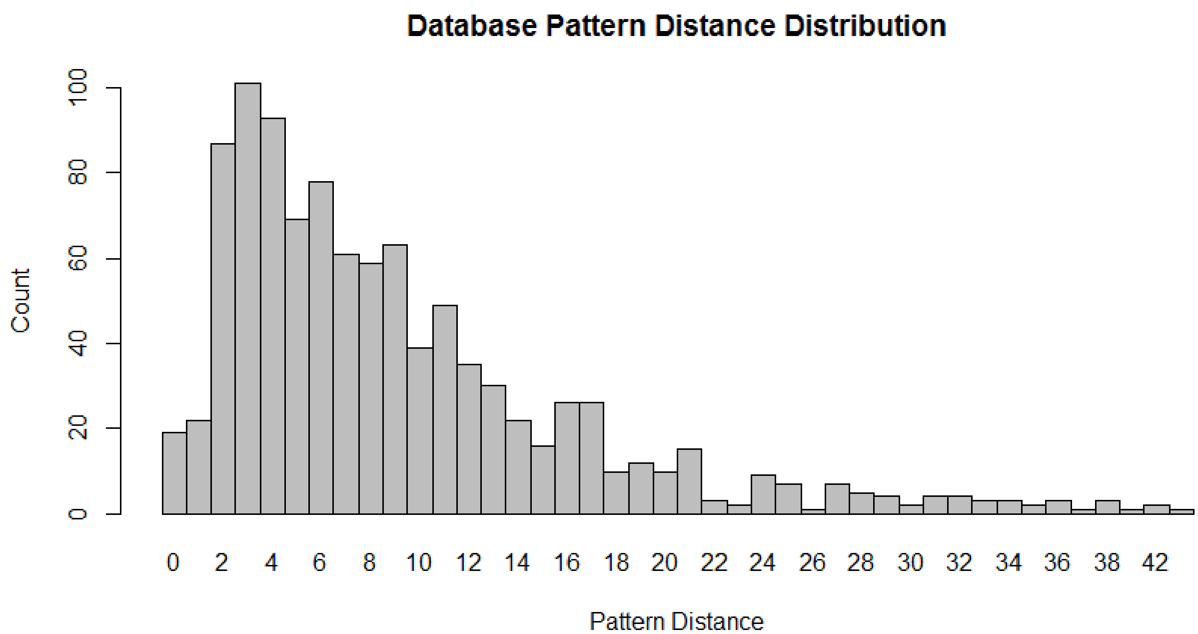


Figure 33. Database Pattern Distribution

7.3.2 Sentence Classification

RQ3.3: How effectively does REDE detect ACR sentences?

By using our classifier trained on the CyberChair data, we had a precision of 0.82, a recall of 0.29 and a F_1 score of 0.42 when classifying the OCS User Manual for ACR sentences. If we reversed the test, we had a precision of 0.75, a recall of 0.61 and a F_1 score of 0.67 when classifying the CyberChair paper for ACR sentences. Using a 10-fold evaluation on the OCS

User Manual, the classifier had a precision of 0.81, a recall of 0.78, and a F_1 score of 0.79. As a trained classifier may not be readily available, we also evaluated how well the classification would perform by working through the document. Figure 34 shows the results of this test. Classifier performance climbs to over a 0.7 F_1 score with less than 25% of the documents evaluated. However, users need to be wary of the distribution of sentences within the document. As CyberChair was an academic paper, dropped significantly when the author began discussing the formal implementation of the system along with prior work.

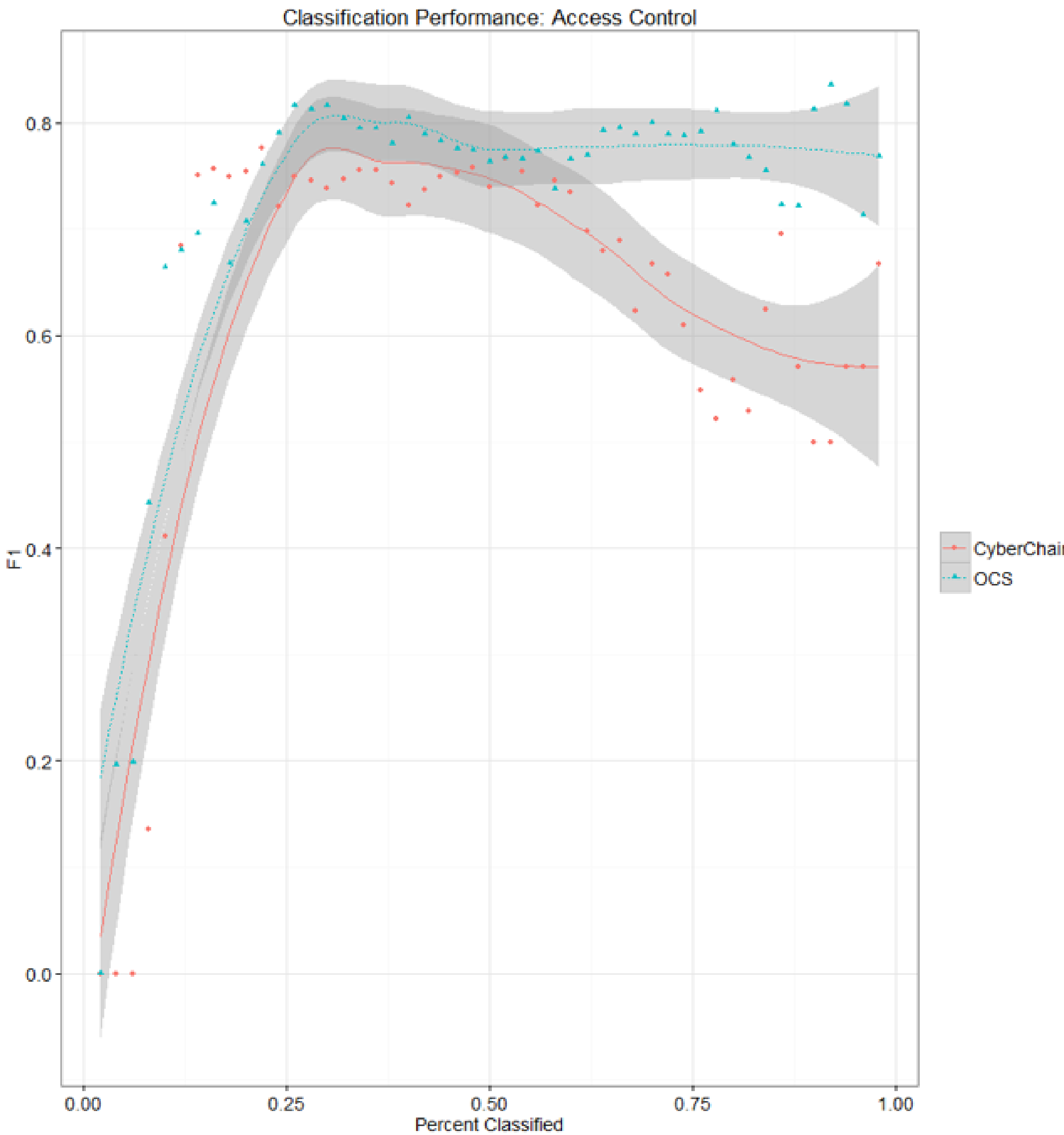


Figure 34. Classification Performance for ACRs by Document Completion %

RQ3.4: How effectively does REDE detect DME sentences?

REDE's performance for classifying DME sentences is nearly the same as for ACRs. Given the almost complete overlap between the categories (ACR sentences and DME sentences), the comparable performance is expected. By using our classifier trained on the

CyberChair data, we had a precision of 0.82, a recall of 0.29 and a F_1 score of 0.42 when classifying the OCS User Manual for DME sentences. If we reversed the test, we had a precision of 0.75, a recall of 0.61 and a F_1 score of 0.67 when classifying the CyberChair paper for DME sentences. Using a 10-fold evaluation on the OCS User Manual, the classifier had a precision of 0.83, a recall of 0.79, and a F_1 score of 0.80. Figure 35 shows the performance for classifying DMEs when a trained classify is not readily available. The increased variability scores as the document becomes more complete is due to the increased relativity that right or wrong answers affect the score as there are less instances being tested.

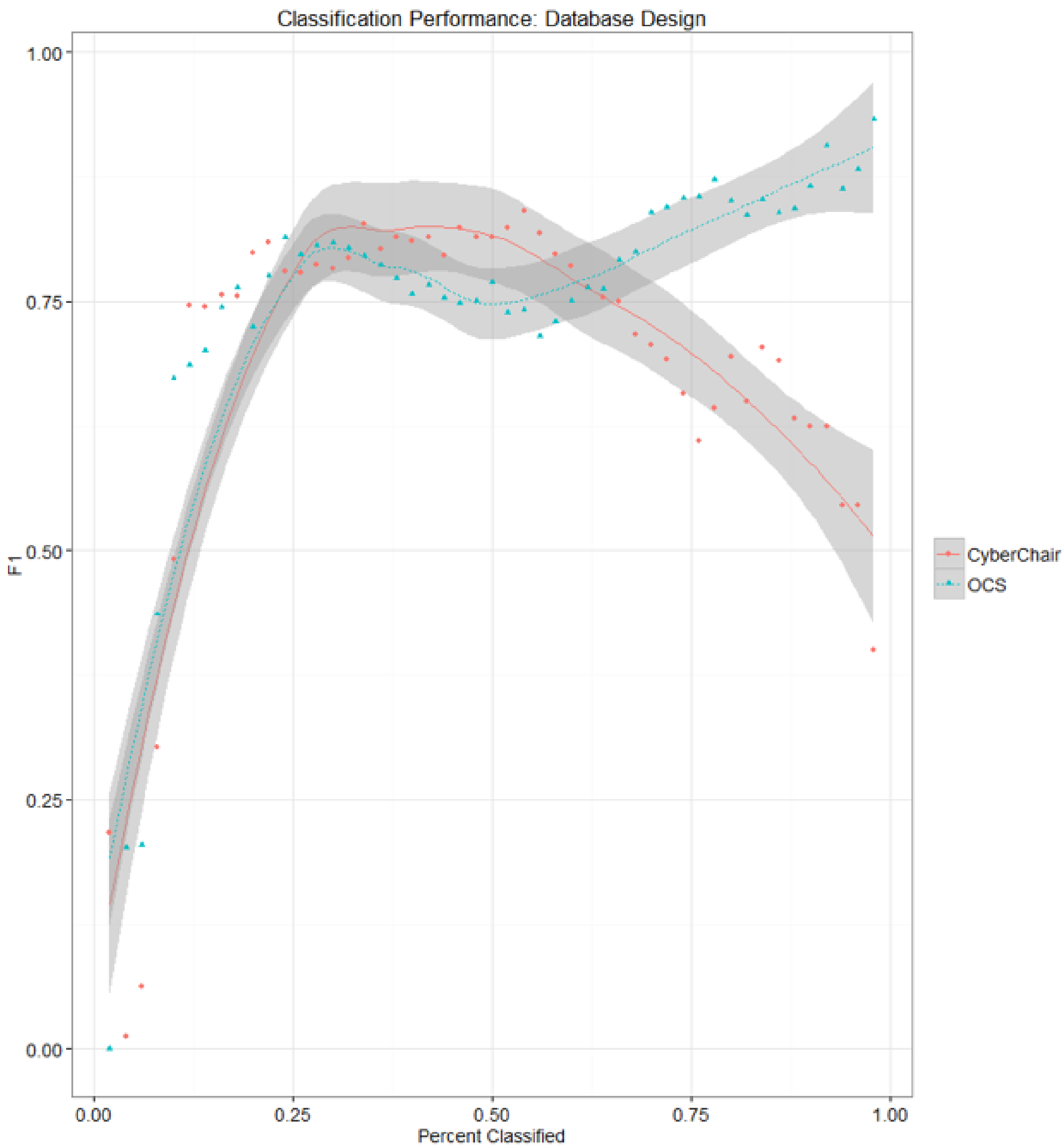


Figure 35. Classification Performance for DMEs by Document Completion %

7.3.3 Relation Extraction

RQ3.5: How effectively can the subject, action, and resource elements of ACRs be extracted from ACR sentences?

REDE initially had a performance scores of 0.52 for precision, 0.16 for recall, and 0.24 for the F_1 score. If we allowed actions, resources, and prepositions to be wildcard, the performance scores would increase slightly to 0.53 for precision, 0.27 for recall, and 0.35 for the F_1 score. By adding the subject to be wildcarded as well, the recall would increase to 0.40, but the precision would drop by half to 0.26.

Table 22 lists the top 10 most common errors for ACR extraction. The initial error occurs as the REDE process has not yet recognized enough words in the problem domain to substitute values effectively for the direct object. 61 of the other errors occurs due to the presence of pronouns in the pattern. As the REDE process was designed, no seed pattern existed with pronouns.

Table 22. Top 10 ACR Extraction Errors

Number Occurences	Error Type	Pattern
89	FN	(% VB root (% NN dobj))
36	FN	(% VB root (% PRP nsubj)(% NN dobj))
20	FN	(% VB root (% NN prep_%))
18	FN	(% VB root (% NN nsubj)(% NN dobj))
17	FP	(% VB root (% NN nsubjpass))
12	FN	(% VB root (% PRP nsubj)(% NN prep_%))
8	FP	(% VB root (% PRP nsubj)(% NN dobj))
5	FN	(allow VB root (% PRP dobj)(% VB dep (% NN dobj)))
5	FN	(% VB root (% NN nsubj)(% NN prep_%))
5	FN	(% VB root (% NN nsubjpass))

RQ3.6: How effectively can the entities, entity-attributes, and relationship elements be extracted from DME sentences?

To evaluate DME extraction process, we assumed that we started with perfect knowledge of the ACRs. From completing the eight template patterns with the known subjects, actions, and resources, we had a precision of 1.0, a recall of .89 and a F_1 score of .94. Starting from the ACR process as executed, we again had a precision of 1.0, but the recall fell to .81 as there were not specific entities and relations already defined from the ACRs.

7.3.4 Database Mapping

RQ3.7: How effectively are extracted database model elements mapped to a physical database schema?

To evaluate this question, we identified the 21 discovered subjects from the extracted ACRs, the 223 discovered entities from the DMEs, and the 83 discovered relationships to the 7 application roles and 52 physical database tables. Utilizing the process, we missed 10 of the 223 entities identified within the documentation.

7.3.5 Role Implementation and Validation

RQ3.8: How effectively does REDE implement access control within a system's relation database?

From the extracted ACRs, we identified 686 out of 730 grant statements to be created and executed to implement the RBAC for OCS.

7.4 Limitations

As this work builds upon our prior studies presented in Chapters 5 and 6, the limitations presented in those chapters carry forward into this study. Additionally, we have several limitations added from this chapter. We utilized a relatively naïve mechanism to resolve ambiguity with pronouns and missing subjects. In future work, we could utilize resolution methods contained within the natural language parsers.

From a database perspective, we only considered access at the table level. We could create specific rules for the identified attributes rather than converting their access to the entity level. Additionally, we manually mapped the discovered roles and entities to the actual database tables. Future work can bring in schema-matching technologies to automate the mapping. Implementations may differ as well based upon specific system needs. For instance, effective dated records could be used to replace updates and deletes. Additionally, the implemented system may not match documentation.

This study may not be generalizable as we evaluated just one system within a specific problem domain for the end to end resolutions. Future studies need to examine different systems and different types of documents.

8 Conclusions

This chapter presents the conclusion to this dissertation. Section 8.1 summarizes the dissertation. Section 8.2 lists our contributions. In Section 8.3, we describe future work.

8.1 Summary

To improve security and compliance, we have proposed and evaluated a method to find and access control rules from natural language product artifacts and then implement those rules within the persistence layer for an application. The tool-support process consists of six steps: 1) parse existing, unconstrained natural language product artifacts; 2) classify whether or not a sentence in the product artifact implies access control and whether or not the sentence implies database model; and, as appropriate, 3) extract ACR elements; 4) extract DMEs; 5) map extracted data model to a database schema; and 6) implement role-based access control (RBAC) within a relation database. In recalling Yue et al.'s five characteristics that approaches should have to transform requirements into analysis models (see Section 3.1), our approach either meets or measures all five characteristics. We parse existing, unaltered natural language documents. We provide measurements as towards the completeness and correctness of the generated access control model. Our approach uses a minimal amount of transformations to generate the access control model. The approach is largely automated, but allows the user to correct identified mistakes. Finally, our approach provides traceability from the source documents to the generated access control statements.

In our first study, we demonstrated a technique to parse natural language texts (step 1) and how to most effectively classify statements for particular properties (step 2). Specifically, we classified which sentences explicitly or implicitly contain non-functional requirements (NFRs) among 14 NFR categories (e.g. capacity, reliability, and security). Our k-nearest neighbor classifier with a unique distance metric had a F_1 measure of 0.55, outperforming in our experiments the optimal naïve Bayes classifier which had a F_1 measure of 0.27.

In our second study, we identified sentences containing ACRs (step 2) and then inferred those rules from the text (step 3). To infer the rules, we extract relations (i.e., the relationship among two or more items) from natural language (NL) artifacts such as requirements documents. Unlike existing approaches, our approach combines techniques from information extraction and machine learning. Our evaluation results show that ACRs exist in 47% of the sentences, and our approach effectively identifies those ACR sentences with a precision of 81% and recall of 65%. Our approach extracts ACRs from those identified ACR sentences with an average precision of 76% and an average recall of 49%.

In our final study, we followed the process from start to finish with a focus on identifying sentences containing database model elements (step 2), extracting the database model (step 4), mapping the extracting database elements to the physical database schema (step 5), and implementing RBAC (step 6) within an applications database. Our approach extends existing work to extract conceptual database models from NL artifacts using relations based upon heuristics defined in those works, but then adopts machine learning techniques to learn new rules to extract database entities, attributes, and relationships. We then presented guidance and tool support for mapping the extracted data model to a system's physical database schema. We performed system testing on the application to ensure the database appropriately enforces access control as defined. Our approach extracts database elements from the text with a F_1 score of 0.94 assuming the ACR extraction was correct.

8.2 Contributions

Through this dissertation, we provide the following contributions:

- Process to identify NFRs by categories within available natural language documentation;
- Distribution report containing the NFR categories and frequencies by document type;
- Empirical performance results for machine learning classifiers on NFRs;
- Sentence similarity algorithm to use with a k-nearest neighbor classifier or a k-medoids clustering algorithm;
- Bootstrapping mechanism to seed and iteratively discover ACR patterns in NL text;
- Bootstrapping mechanism to seed and iteratively discover database model elements patterns in NL text;
- Approach and supporting tool (built upon the combination of NLP, IE, and ML techniques) to extract ACRs and database model elements;
- Evaluation of ACRE for ACRs against product artifacts from systems in three domains: conference management, education, and healthcare;
- Start to finish evaluation of REDE for an open-source conference management system; and
- Publically-available labeled corpus of identified NFRs, a corpus of identified ACRs and sentences and a corpus of identified database elements.

For practitioners, we developed an open-source tool to incorporate into any software engineering methodology our process to generate database access control from unconstrained natural language text. Document writers can also use the tool to measure completeness as well as receive reports on possible ambiguity within documents.

8.3 Future Work

While we have demonstrated an effective process to extract both ACRs and DMEs from sentences, more work in this area. The ACR extraction process can be extended to supporting conditions for ACRs as well as temporal logic for them. The database extraction process can be extended to pull out additional information about the fields. What is the data type for a field? What are the possible ranges or values for a field?

While the performance of NLP-based systems has made substantial progress over the past fifteen years, work remains to improve the effectiveness of such system. One of the primary roadblocks is the inherent ambiguity and complexity of natural language [Jurafsky & Martin, 2009]. As these challenges may never be overcome directly, we should also look towards other solution methods. Following the lead of Norman [Norman, 2013], we can study the interplay between humans and computers in this area. Each group has positives and negatives. Computers can process text and numbers faster than any human could. However, computers lack the cognitive insight humans bring to solving problems. We can study how computers can detect situations such as ambiguity and then provide effective information and interfaces for humans to provide the necessary corrections.

REFERENCES

2011 CWE/SANS Top 25 Most Dangerous Software Errors. 2011. Available from: http://cwe.mitre.org/top25/

Ackerman WB, Plummer WW. 1967. An implementation of a multiprocessing computer system. In: ACM Symposium on Operating System Principles. Gatlinburg. p 1–10.

Akbik A, Broß J. 2009. Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns. In: Workshop on Semantic Search. Vol. 491. Madrid, Spain. Available from: http://ceur-ws.org/Vol-491/semse2009_7.pdf

Ambriola V, Gervasi V. 2006. On the systematic analysis of natural language requirements with CIRCE. Automated Software Engineering [Internet] 13:107–167. Available from: http://www.scopus.com/inward/record.url?eid=2-s2.0-29244483736&partnerID=40&md5=eff61578d596de4dda13f1020abea81e

Ambriola V. 2006. On the Systematic Analysis of Natural Language Requirements with CIRCE. Automated Software Engineering [Internet] 13:107–167. Available from: http://www.springerlink.com/index/F30GX355P452G406.pdf

ANSI/ISO/IEC 9075-2:1999, Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation). 1999. ISO.

Batini C, Lenzerini M, Navathe SB. 1986. A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys [Internet] 18:323–364. Available from: http://portal.acm.org/citation.cfm?doid=27633.27634

Bell D, LaPadula LJ. 1973. Secure computer systems: Mathematical foundations. Data Base [Internet] 1:513–523. Available from: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0770768\nhttp://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0770768

Bellman R. 1957. Dynamic Programming. Princeton: Princeton University Press.

Bertman J, Skolnik N. 2010. EHRs Get a Failing Grade on Usability. Internal Medicine News [Internet] 43:50. Available from: http://linkinghub.elsevier.com/retrieve/pii/S1097869010705862

Boehm BW. 1984. Verifying and Validating Software Requirements and Design Specifications. IEEE Software [Internet] 1:75–88. Available from:

http://ieeexplore.ieee.org/xpl/articleDetails.jsp;jsessionid=qqvsPB2bhGh2N85q1lQ1ZW2vnc3Yh7tgdXyNLYsRG4vF6G7mjw16!1226760293?arnumber=1695100&contentType=Journals+&+Magazines\nhttp://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1695100

Borg M, Runeson P, Ardö A. 2013. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering:1–52.

Brewer E. 2012. CAP Twelve Years Later: How the“ Rules” Have Changed. Computer [Internet] 45:23–29. Available from: http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

Brodie C a., Karat C-M, Karat J. 2006. An Empirical Study of Natural Language Parsing of Privacy Policy Rules Using the SPARCLE Policy Workbench. In Proc. SOUPS [Internet]:8–19. Available from: http://portal.acm.org/citation.cfm?doid=1143120.1143123

Browne J. 2008. Systemic Requirements. Available from: http://www.julianbrowne.com/article/viewer/systemic-requirements

Buckland M, Gey F. 1994. The relationship between Recall and Precision. Journal of the American Society for Information Science 45:12–19.

Budgen D, Turner M, Brereton P, Kitchenham B. 2008. Using mapping studies in software engineering. Proceedings of PPIG [Internet] 2. Available from: http://www.ppig.org/papers/20th-budgen.pdf

Caramazza A, Grober E, Garvey C, Yates J. 1977. Comprehension of anaphoric pronouns. Journal of Verbal Learning and Verbal Behavior 16:601–609.

Casamayor A, Godoy D, Campo M. 2010. Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. Information and Software Technology [Internet] 52:436–445. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0950584909001918

Casamayor A, Godoy D, Campo M. 2012. Mining textual requirements to assist architectural software design: A state of the art review. Artificial Intelligence Review 38:173–191.

Chen PP-S. 1983. English sentence structure and entity-relationship diagrams. Information Sciences 29:127–149.

Chinchor N, Sundheim B. 1996. Message Understanding Conference - 6: A Brief History. In: In Proc. Coling. Association for Computational Linguistics. p 466–471. Available from: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Message+Understanding+Conference#6

Choi N, Song IY, Han H. 2006. A Survey on Ontology Mapping. ACM Sigmod Record [Internet] 35:34–41. Available from: http://dl.acm.org/citation.cfm?id=1168097

Chomsky N. 1956. Three models for the description of language. Information Theory, IRE Transactions on [Internet] 2:113–124. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1056813

Chung L, Nixon BA, Yu E, Mylopoulos J. 2000. Non-functional Requirements in Software Engineering. Norwell, MA, USA: Kluwer Academic Publishers Group.

Clark DD, Wilson DR. 1987. A Comparison of Commercial and Military Computer Security Policies. In: In Proceedings of the 1987 IEEE Symposium on Security and Privacy. . p 184–194.

Cleland-Huang J, Settimi R, Zou X, Solc P. 2007. Automated classification of non-functional requirements. Requirements Engineering 12:103–120.

Codd EF. 1970. A relational model of data for large shared data banks. Communications of the ACM [Internet] 13:377–387. Available from: http://dx.doi.org.prox.lib.ncsu.edu/10.1145/362384.362685

Cohen J. 1960. A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement [Internet] 20:37–46. Available from: http://epm.sagepub.com/cgi/doi/10.1177/001316446002000104

CoNLL-2003. 2003. Language-Independent Named Entity Recognition. Available from: http://www.cnts.ua.ac.be/conll2003/ner/

Data Breaches: A Year in Review. 2011. Privacy Rights Clearinghouse [Internet]. Available from: https://www.privacyrights.org/top-data-breach-list-2011

DB-Engines. 2013. Available from: http://db-engines.com/en/

Defense D of. 1985. Trusted Computer System Evaluation Criteria. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.118.9902

Diaz I, Pastor O, Matteo a. 2005. Modeling interactions using role-driven patterns. 13th IEEE International Conference on Requirements Engineering (RE’05).

Digital Data on Patients Raises Risk of Breaches. 2011. NY Times [Internet]. Available from: http://www.nytimes.com/2011/12/19/technology/as-patient-records-are-digitized-data-breaches-are-on-the-rise.html

Dixon RMW. 2005. A Semantic Approach to English Grammar. Second. Oxford University Press, USA. Available from: http://books.google.com/books?id=19PDGPGqcjAC&printsec=frontcover&dq=A+Semantic+Approach+to+English+Grammar&hl=en&ei=UcLXTqSdA8Hlggf0m8mHDw&sa=X&oi=book_result&ct=result&resnum=1&ved=0CDAQ6AEwAA#v=onepage&q&f=false

Doan AH, Halevy AY. 2005. Semantic Integration Research in the Database Community : A Brief Survey. AI magazine [Internet] 26:83. Available from: http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1801

Donston-miller D. 2011. Ensuring Secure Database Access. Available from: http://twimgs.com/darkreading/databasesecurity/

Du S. 2008. On the Use of Natural Language Processing for Automated Conceptual Modelling. :201.

Duan C, Cleland-Huang J. 2007. Clustering support for automated tracing. Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering - ASE '07 [Internet]:244. Available from: http://portal.acm.org/citation.cfm?doid=1321631.1321668

El-aziz AEAA, Kannan A. 2014. XML Access Control : Mapping XACML Policies to Relational Database Tables. The International Arab Journal of Information Technology 11:532–539.

Elbendak M, Vickers P, Rossiter N. 2011. Parsed use case descriptions as a basis for object-oriented class model generation. Journal of Systems and Software [Internet] 84:1209–1223. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0164121211000537

Elmasri R, Navathe S. 2010. Fundamentals of Database Systems. 6th ed. Addison-Wesley Publishing Company.

Falessi D, Cantone G, Canfora G. 2010. A comprehensive characterization of NLP techniques for identifying equivalent requirements. Proceedings of the 2010 ACM-IEEE … [Internet]:18:1–18:10. Available from: http://doi.acm.org/10.1145/1852786.1852810\nhttp://dl.acm.org/citation.cfm?id=1852810

Fernandez EB, Hawkins JC, Raton B. 1997. Determining Role Rights from Use Cases. In: In Proc. ACM Workshop on RBAC. . p 121 – 125.

Ferraiolo DF, Kuhn DR. 1992. Role-Based Access Controls. In: Proceedings of the 15th National Computer Security Conference. Baltimore. p 554–563.

Fleiss JL. 1971. Measuring nominal scale agreement among many raters. Psychological Bulletin [Internet] 76:378–382. Available from: http://content.apa.org/journals/bul/76/5/378

Frakes WB. 1992. Information retrieval. In: Frakes WB, Baeza-Yates R, editors. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. p 1–12. Available from: http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=129687.129688

Gesmundo A, Samardžić T. 2012. Lemmatisation as a Tagging Task. In Proce. ACL [Internet]:368–372. Available from: http://dl.acm.org/citation.cfm?id=2390748

Glinz M. 2007. On Non-Functional Requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007). Ieee. p 21–26. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4384163

Hall M, National H, Frank E, et al. 2009. The WEKA Data Mining Software : An Update. SIGKDD Explorations 11:10–18.

Han J, Kamber M, Pei J. 2011. Data Mining: Concepts and Techniques. 3rd ed. Morgan Kaufmann.

Harmain HM., Gaizauskas R. 2003. CM-Builder: A natural language-based CASE tool for object-oriented analysis. Automated Software Engineering [Internet] 10:157–181. Available from: http://www.scopus.com/inward/record.url?eid=2-s2.0-0037385209&partnerID=40&md5=a7a19e7d1d72ddc7be9c7ddf027c565a

Hartmann S, Link S. 2007. English sentence structures and EER modeling. In: Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling-Volume 67. Vol. 67. Ballarat, Australia: Australian Computer Society, Inc. p 27–35. Available from: http://dl.acm.org/citation.cfm?id=1274460

He Q, Antón AI. 2009. Requirements-based Access Control Analysis and Policy Specification (ReCAPS). Information and Software Technology [Internet] 51:993–1009. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0950584908001699

Hearst M. 1992. Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th conference on Computational Linguistics. Nantes, France. p 539–545. Available from: http://dl.acm.org/citation.cfm?id=992154

Helms ED, Hwang J, Williams L, Xie T. 2014. A Systematic Literature Review on Extensions to the Role-Based Access Control Reference Model. Work in Progress.

Hirschman L, Thompson HS. 1997. Overview of Evaluation in Speech and Natural Language Processing. In: Survey of the State of the Art in Human Language Technology. . p 409–414.

Hirschman L. 1998. The Evolution of evaluation: Lessons from the Message Understanding Conferences. Computer Speech & Language [Internet] 12:281–305. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0885230898901024

Hoskinson C. 2011. Army's Faulty Computer System Hurts Operations. Politico [Internet]. Available from: http://www.politico.com/news/stories/0611/58051.html

Huddleston R, Pullman G. 2002. The Cambridge Grammar of the English Language. First. Cambridge University Press.

IBM Watson: Ushering in a new era of computing. 2012. IBM [Internet]. Available from: http://www-03.ibm.com/innovation/us/watson/

IBM. 2004. Course Registration Requirements.

IEEE Recommended Practice for Software Requirements Specifications - IEEE Std 830-1998. 1998. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574&isnumber=15571

Inglesant P, Sasse MA, Chadwick D, Shi LL. 2008. Expressions of Expertness: The Virtuous Circle of Natural Language for Access Control Policy Specification. In: In Proc. SOUPS. ACM. p 77–88. Available from: http://dl.acm.org/citation.cfm?id=1408675

ISO Standards Committee JTC 1/SC 32. 2011. ISO/IEC 9075-1:2011,- Information technology -- Database languages -- SQL -- Part 1: Framework (SQL/Framework).

ISO/IEC/IEEE. 2010. Systems and software engineering -- Vocabulary. ISO/IEC/IEEE 24765:2010(E) [Internet]:1–418. Available from: http://ieeexplore.ieee.org/servlet/opac?punumber=5733833

Joachims T. 1998. Text categorization with support vector machines: Learning with many relevant features. Machine Learning: ECML-98 [Internet]. Available from: http://www.springerlink.com/index/drhq581108850171.pdf

Jurafsky D, Martin J. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Second. Pearson.

Kalfoglou Y, Schorlemmer M. 2003. Ontology mapping: the state of the art. The Knowledge Engineering Review [Internet] 18:1–31. Available from: http://www.journals.cambridge.org/abstract_S0269888903000651

Kennedy C, Boguraev B. 1996. Anaphora for everyone: pronominal anaphora resoluation without a parser. In: In Proc. Coling. Vol. 1. . p 113–118. Available from: http://dl.acm.org/citation.cfm?id=992651

Kitchenham B, Charters S. 2007. Guidelines for performing systematic literature reviews in software engineering.

Ko Y, Park S, Seo J, Choi S. 2007. Using classification techniques for informal requirements in the requirements analysis-supporting system. Information and Software Technology 49:1128–1140.

Lampson BW. 1971. Protection. In: Proceedings of the 5th Princeton Conference on Information Sciences and Systems. Vol. 8. Princeton. p 437–444.

Landis JR, Koch GG. 1977. The Measurement of Observer Agreement for Categorical Data Data for Categorical of Observer Agreement The Measurement. Biometrics [Internet] 33:159–174. Available from: http://www.jstor.org/stable/2529310

Levenshtein VI. 1966. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics Doklady [Internet] 10:707–710. Available from: http://www.mendeley.com/research/binary-codes-capable-of-correcting-insertions-and-reversals/

Li Y, McLean D, Bandar Z a., O'Shea JD, Crockett K. 2006. Sentence similarity based on semantic nets and corpus statistics. IEEE Transactions on Knowledge and Data Engineering 18:1138–1150.

Luisa M, Mariangela F, Pierluigi NI. 2004. Market research for requirements analysis using linguistic tools. Requirements Engineering 9:40–56.

Manning C, Raghavan P, Schütze H. 2008. Introduction to Information Retrieval. Cambridge University Press. Available from: http://nlp.stanford.edu/IR-book/

Manning C. 2011. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In: In Proc. CICLing. . p 171–189. Available from: http://link.springer.com/chapter/10.1007/978-3-642-19400-9_14

De Marneffe M-C, MacCartney B, Manning C. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In: In Proc. LREC. . p 449–454. Available from: http://nlp.stanford.edu/manning/papers/LREC_2.pdf

Meneely A, Smith B, Williams L. 2011. iTrust Electronic Health Care System: A Case Study. In: Software System Traceability.

Menzies T, Caglayan B, Kocaguneli E, et al. 2012. The PROMISE Repository of Empirical Software Engineering Data. West Virginia University, Department of Computer Science [Internet]. Available from: http://promisedata.googlecode.com/

Meth H, Brhel M, Maedche A. 2013. The state of the art in automated requirements elicitation. Information and Software Technology [Internet] 55:1695–1709. Available from: http://dx.doi.org/10.1016/j.infsof.2013.03.008

MongoDB. 2014. MongoDB Manual - Security Concepts. Available from: http://docs.mongodb.org/manual/core/authorization/

Mu Y, Wang Y, Guo J. 2009. Extracting software functional requirements from free text documents. 2009 International Conference on Information and Multimedia Technology, ICIMT 2009:194–198.

Niu N, Reddivari S, Mahmoud A, Bhowmik T, Xu S. 2012. Automatic labeling of software requirements clusters. 2012 4th International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation, SUITE 2012 - Proceedings:17–20.

Noll J, Liu W-M. 2010. Requirements elicitation in open source software development. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development - FLOSS ’10. Cape Town, South Africa: ACM Press. p 35–40. Available from: http://portal.acm.org/citation.cfm?doid=1833272.1833279

Norman D. 2013. The Design of Everyday Things: Revised and Expanded Edition. New York: Basic Books.

Okman L, Gal-Oz N, Gonen Y, Gudes E, Abramov J. 2011. Security Issues in NoSQL Databases. In: Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM ’11). IEEE. p 541–547. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6120863

Oliva J, Serrano JI, Del Castillo MD, Iglesias Á. 2011. SyMSS: A syntax-based measure for short-text semantic similarity. Data and Knowledge Engineering [Internet] 70:390–405. Available from: http://dx.doi.org/10.1016/j.datak.2011.01.002

Omar N, Hassan R, Arshad H. 2008. Automation of database design through semantic analysis. In: Proceedings of the 7th WSEAS International Conference on Computational Intelligence, Man-machine Systems and Cybernetics. Cairo, Egypt. p 71–76. Available from: http://dl.acm.org/citation.cfm?id=1569519

Omar N. 2004. Heuristics-based entity-relationship modelling through natural language processing. :247.

Osborn S, Sandhu R, Munawer Q. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Transactions on Information and System Security 3:85–106.

Park S, Kim H, Ko Y, Seo J. 2000. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. Information and Software Technology 42:429–438.

Pascual A, Miller S. 2015. 2015 Identity Fraud – Protecting Vulnerable Populations. Available from: https://www.javelinstrategy.com/brochure/347

Petersen K, Feldt R. 2008. Systematic mapping studies in software engineering. … in Software Engineering [Internet]:1–10. Available from: http://robertfeldt.net/publications/petersen_ease08_sysmap_studies_in_se.pdf

Peterson A. 2015. 2015 is already the year of the health-care hack — and it ’ s only going to get worse . Washington Post [Internet]. Available from: http://www.washingtonpost.com/blogs/the-switch/wp/2015/03/20/2015-is-already-the-year-of-the-health-care-hack-and-its-only-going-to-get-worse/

Piskorski J, Yangarber R. 2013. Information Extraction: Past, Present, and Future. In: Poibeau T, Saggion H, Piskorski J, Yangarber R, editors. Multi-source, Multilingual Information Extraction and Summarization. Theory and Applications of Natural Language Processing. Berlin, Heidelberg: Springer Berlin Heidelberg. p 23–50. Available from: http://link.springer.com/10.1007/978-3-642-28569-1

Po L, Sorrentino S. 2011. Automatic generation of probabilistic relationships for improving schema matching. Information Systems [Internet] 36:192–208. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0306437910000980

Porter M. 1980. An algorithm for suffix stripping. Program [Internet] 14:130–137. Available from: http://www.lirmm.fr/~mroche/Recherche/Articles/Porter/porter.pdf

Potts C, Recasens M. 2012. The Life and Death of Discourse Entities : Identifying Singleton Mentions. 0.

Public Knowledge Project. 2015. Open Conference Systems. Available from: https://pkp.sfu.ca/ocs/

Rahm E, Bernstein P a. 2001. A survey of approaches to automatic schema matching. The VLDB Journal [Internet] 10:334–350. Available from: http://www.springerlink.com/index/10.1007/s007780100057

Randolph JJ. 2008. Online Kappa Calculator. Available from: http://justusrandolph.net/kappa/

Riaz M, King J, Slankas J, Williams L. 2014. Hidden in Plain Sight: Automatically Identifying Security Requirements from Natural Language Artifacts. In: Proceedings of the 22nd IEEE International Requirements Engineering Conference (to appear). Karlskrona, Sweden.

Rumbaugh J, Blaha M, Lorensen W, Eddy F, Premerlani W. 1991. Object-Oriented Modeling and Design. 1st ed. Prentice Hall.

Sadalage P, Fowler M. 2012. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional.

Salton G, McGill MJ. 1986. Introduction to Modern Information Retrieval. New York, NY, USA: McGraw-Hill, Inc.

Saltzer JH, Schroeder MD. 1975. The Protection of Information in Computer Systems A . Considerations Surrounding the Study of Protection. Proceedings of the IEEE [Internet] 63:1278 – 1308. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1451869

Samarati P, Vimercati S de. 2001. Access control: Policies, models, and mechanisms. Foundations of Security Analysis and Design [Internet]:137–196. Available from: http://www.springerlink.com/index/80wrewj7j1a716wb.pdf

Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. 1996. Role-based access control models. Computer 29:38–47.

Santorini B. 1995. Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision, 2nd printing).

Schmidt JY, Antón AI, Williams L, Otto PN. 2011. The Role of Data Use Agreements in Specifying Legally Compliant Software Requirements. In: Fourth International Workshop on Requirements Engineering and Law. . p 15–18.

Schneider K, Knauss E, Houmb S, Islam S, Jürjens J. 2012. Enhancing security requirements engineering by organizational learning. Requirements Engineering 17:35–56.

Schwitter R. 2010. Controlled Natural Languages for Knowledge Representation. In: In Proc. CICLing. Association for Computational Linguistics. p 1113–1121. Available from: http://dl.acm.org/citation.cfm?id=1944694

Shi L, Chadwick D. 2011. A Controlled Natural Language Interface for Authoring Access Control Policies. In: In Proc. SAC. . p 1524–1530. Available from: http://dl.acm.org/citation.cfm?id=1982510

Slankas J, Williams L. 2012. Classifying Natural Language Sentences for Policy. 2012 IEEE International Symposium on Policies for Distributed Systems and Networks [Internet]:33–36. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6267998

Slankas J, Williams L. 2013a. Automated extraction of non-functional requirements in available documentation. In: 2013 1st International Workshop on Natural Language Analysis in Software Engineering, NaturaLiSE 2013 - Proceedings. . p 9–16. Available from: http://www.scopus.com/inward/record.url?eid=2-s2.0-84886694829&partnerID=40&md5=ac38fc70a97a4032d293dc5b1e9e904d

Slankas J, Williams L. 2013b. Access control policy extraction from unconstrained natural language text. In: Proceedings - SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013. . p 435–440. Available from: http://www.scopus.com/inward/record.url?eid=2-s2.0-84893585590&partnerID=40&md5=3138f9c8565c51e9d7f9a6ce2149ff18

Slankas J, Williams L. 2015. Relation Extraction for Inferring Database Models from Natural Language Artifacts. In: in progress.

Slankas J, Xiao X, Williams L, Xie T. 2014. Relation Extraction for Inferring Access Control Rules from Natural Language Artifacts. In: Proceedings of the 30th Annual Computer Security Applications Conference. New York, NY, USA: ACM. p 366–375.

Snow R, Jurafsky D, Ng AY. 2004. Learning Syntactic Patterns for Automatic Hypernym Discovery. In: Advances in Neural Information Processing Systems 17. Vol. 17. Vancouver, British Columbia, Canada. p 1297–1304. Available from: http://ilpubs.stanford.edu:8090/665

Socher R, Bauer J, Manning C, Ng A. 2013. Parsing with Compositional Vector Grammars. In Proc. ACL [Internet]. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.387.6840

De Souza SCB, Anquetil N, de Oliveira KM. 2005. A Study of Documentation Essential to Software Maintenance. In: Proceedings of the 23rd annual international conference on Design of communication documenting & designing for pervasive information - SIGDOC '05. New York, New York, USA: ACM Press. p 68–75. Available from: http://portal.acm.org/citation.cfm?doid=1085313.1085331

Stadt R Van De. 2012. Cyberchair: A web-based groupware application to facilitate the paper reviewing process. arXiv arXiv:1206.1833 [Internet]. Available from: http://arxiv.org/abs/1206.1833

Stanford Natural Language Processing Group. 2015. Stanford Parser FAQs. Available from: http://nlp.stanford.edu/software/parser-faq.shtml#z

Stonebraker M. 2011. New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps. Communications of the ACM, Blog [Internet]. Available from: http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext

Storey VC. 1993. Understanding semantic relationships. The VLDB Journal [Internet] 2:455–488. Available from: http://www.springerlink.com/index/10.1007/BF01263048

Surdeanu M, Johansson R. 2008. The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In: In Proc. CoNLL. Association for Computational Linguistics. p 159–177. Available from: http://dl.acm.org/citation.cfm?id=1596352

Vidya Sagar VBR, Abirami S. 2014. Conceptual modeling of natural language functional requirements. Journal of Systems and Software [Internet] 88:25–41. Available from: http://dx.doi.org/10.1016/j.jss.2013.08.036

Witten I, Frank E, Hall M. 2011. Data Mining: Practical Machine Learning Tools and Techniques. 3rd ed. Morgan Kaufmann.

Xiao X, Paradkar A, Thummalapenta S, Xie T. 2012. Automated Extraction of Security Policies from Natural-Language Software Documents. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering.

Xie TXT, Thummalapenta S, Lo D, Liu CLC. 2009. Data Mining for Software Engineering. Computer 42:55–62.

Yue T. b, Briand LC., Labiche Y. 2011. A systematic review of transformation approaches between user requirements and analysis models. Requirements Engineering [Internet] 16:75–99. Available from: http://www.scopus.com/inward/record.url?eid=2-s2.0-79955933734&partnerID=40&md5=ee46e7ff9ffdb0cebdafdbe82e65c0ad

Zhang J, Sun Y, Wang H, He Y. 2011a. Calculating Statistical Similarity between Sentences. Journal of Convergence Information Technology 6:22–34.

Zhang W, Yang Y, Wang Q, Shu F. 2011b. An Empirical Study on Classification of Non-Functional Requirements. In: The Twenty-Third International Conference on Software Engineering and Knowledge Engineering (SEKE 2011). . p 190–195. Available from: http://www.ksi.edu/seke/Proceedings/seke11/51_Wen_Zhang.pdf

Zhou G, Su J, Zhang J, Zhang M. 2005. Exploring Various Knowledge in Relation Extraction. In: Proceedings of the 43rd Annual Meeting of the ACL,. Ann Arbor. p 427–434.

Zowghi D, Gervasi V. 2003. On the interplay between consistency, completeness, and correctness in requirements evolution. Information and Software Technology [Internet] 45:993–1009. Available from: http://linkinghub.elsevier.com/retrieve/pii/S0950584903001009