A Refined Production Rule Model for Aiding in Regulatory Compliance

Jeremy C. Maxwell and Annie I. Antón
{jcmaxwe3, aianton}@ncsu.edu

Abstract—Software engineers are being asked to develop software for increasingly regulated environments. When systems are not dependably compliant, companies must pay the high cost of non-compliance, including the cost of lost reputation and brand damage. Regulations represent the minimum level of security and dependability with which systems must comply. We develop a methodology for creating production rule models to aid developers in specifying legally compliant software requirements. By querying production rule models, software engineers can gain valuable knowledge of the legal text. They can perform an initial compliance analysis and obtain preliminary compliance requirements that can be further refined in consultation with a lawyer. We model the law using the legal concepts of rights, obligations, privileges, no-rights, powers, liabilities, immunities, and disabilities. Herein, we develop heuristics for specifying production rules that model legal texts. We refined our methodology within the context of a case study in which we model the Privacy Rule, Part E, of the Health Insurance Portability and Accountability Act (HIPAA).

Index Terms—Healthcare, Logic Programming, Regulatory Compliance, Requirements Engineering

1 INTRODUCTION

Company and brand reputation are increasingly driving information privacy and security concerns, as companies are becoming more mindful of the negative press that often results from privacy and security breaches [9]. Eighty-five percent of the respondents to the 2008 Ernst & Young Global Information Survey reported that lost reputation and brand damage are key drivers for information security in their companies [9]. Laws and regulations represent the minimum level of privacy and security with which companies must comply; regulatory compliance is the starting point of protecting company and brand reputation.

When software systems are not dependably compliant, companies must pay the high cost of non-compliance, including the cost of lost reputation and brand damage. For example, consider the ChoicePoint data breach case [27]. Identity thieves fraudulently accessed 163,000 accounts resulting in at least 800 accounts of identity theft [27]. ChoicePoint paid over 27 million dollars in penalties, including fines, legal fees, and victim restitution, as well as complying with government audits for 20 years [27]. Recently, one of these audits uncovered additional breaches compromising 13,750 records, for which ChoicePoint paid an additional $275,000 in penalties [19].

Complying with laws and regulations is challenging, because legal texts contain ambiguities, cross-references to sections of the same or different legal texts, and possibly conflicting definitions and domain-specific terminology [26]. In addition, laws and regulations undergo frequent updates and amendments, requiring software engineers to manage and track these changes [26]. Cross-references to external legal texts should be explored to obtain additional software requirements. Engineers unfamiliar with the laws governing a domain need tools and techniques to help identify compliance requirements.

In this paper, we develop a methodology for creating production rule models of legal texts. We model the law using the legal concepts of rights, obligations, privileges, no-rights, powers, liabilities, immunities, and disabilities [16]. In addition, we introduce heuristics to aid in specifying production rule models. By querying our model, software engineers can gain valuable knowledge of the legal text. They can perform an initial compliance analysis and obtain preliminary compliance requirements that can be further refined in consultation with a lawyer. We do not seek to replace lawyers; instead, we propose supplementing interactions with lawyers to make these interactions more efficient and reduce the cost of complying with relevant laws and regulations.

Production rules are stated using Horn clauses connected by logical operators [3]. Each rule is an if-then statement. Many such rules combine to create a knowledge base, also called a rules base. To interact with this rules base, a query is presented as a top-level goal. An inference engine then uses a reasoning strategy, usually backwards chaining, to execute the rules in the rules base. The result is an affirmation or a refutation of the original query [28].

We developed the methodology and heuristics by modeling a portion of the Health Insurance Portability and Accountability Act[1] (HIPAA). We modeled the Privacy Rule, which regulates the use of protected health information (PHI) by certain organizations called covered entities. Failure to comply with the HIPAA can result in civil penalties of $25,000 per individual per violation per year and criminal penalties of a quarter million dollars and 10 years imprisonment.

Regulatory compliance in the healthcare domain is timely; in February 2009, President Obama signed the American Recovery and Reinvestment Act of 2009[2] (ARRA), a stimulus package which appropriates 17 billion dollars for developing electronic health record (EHR) systems [36]—systems that will be regulated by HIPAA. The ARRA provides incentives to healthcare providers to adopt EHRs, with the goal of providing an EHR for each U.S. citizen by 2014 [36]. This is a significant

[1] 45 CFR Parts 160, 162, and 164
[2] Pub. L. No. 111-5. (2009)

challenge, given that 90% of hospitals in the U.S. currently lack even basic EHR systems [17].

The remainder of this paper is outlined as follows: Section 2 reviews related work; Section 3 provides an introduction to Prolog; Section 4 presents our methodology and heuristics for developing production rule models; Section 5 discusses our HIPAA case study; Section 6 considers threats to validity; and Section 7 provides summary remarks and outlines future work.

2 Related Work

In this section, we discuss related work in requirements engineering, logic programming, and expert systems. We model the law using Hohfeld's eight legal concepts. A legal theorist, Hohfeld developed the concepts to clarify the meaning of the term "right" [16]. The eight Hohfeld concepts are:

right – A claim an actor makes that places obligations on other actors [16]. For example, an individual has a right to be notified of an organization's privacy practices.

obligation – An action an actor is required, by law, to carry out. Hohfeld calls these duties [16]. For example, a covered entity is obligated to disclose PHI to the US Department of Health and Human Services (HHS) so that HHS can verify the covered entity's compliance with HIPAA.

privilege – An actor is free from an obligation. A privilege is an action an actor is allowed to perform but not required to perform [16]. For example, a covered entity has the privilege to include a patient in its directory of individual's in its facility.

no-right – Explicitly states that an actor does not have a right [16]. For example, an inmate does not have the right to be notified of a covered entity's privacy practices.

power – The capability of an actor to change a legal relation and imply liabilities on others [16]. For example, covered entities have the power to enter into contracts with business associates.

liability – A responsibility to perform some action [16]. For example, business associate of a covered entity are liable to safeguard PHI, but are not obligated to do so until they receive PHI from the covered entity.

immunity – Just as a privilege signifies that an actor is free from an obligation, an immunity expresses that an actor is free from a legal power [16]. In other words, an immunity is a no-liability. For example, in the US, non-profit organizations are immune from the liability of paying taxes.

disability – Whereas a no-right express that another party does not hold a right, a disability expresses that a party does not hold a legal power [16]. For example, a covered entity does not have the power to authorize a business associate to use or disclose PHI beyond what HIPAA allows.

The concepts are paired through the opposite and correlative relationships. A party cannot hold concepts that are opposites, for example, individuals cannot have both a right to amend PHI about them as well as a no-right to amend PHI about them. Correlative concepts are concepts that imply each other. For example, if an individual has a right to be notified of an organization's privacy practices, the organization is obligated to notify the individual. We discuss Hohfeld's concepts further in Section 4.1.

2.1 Requirements Engineering and the Law

In our previous work, we presented a methodology for developing production rule models [23]. We used the legal concepts of rights, obligations, and permissions (privileges) to model the law, and developed the methodology based on a case study of four HIPAA Privacy Rule sections [23]. Herein, we refine this methodology to include the additional Hohfeldian concepts of no-rights, powers, liabilities, immunities, and disabilities. As discussed in Section 5, these additional concepts allow us to capture important legal requirements that our previous analysis missed. In addition, we specify heuristics to aid in the development of legal texts. We base our refined methodology and heuristics on fifteen sections of the HIPAA Privacy Rule. We have used our four-section model for checking the iTrust Medical Records Systems requirements for HIPAA compliance [24].

Breaux et al. use the Frame Based Requirements Analysis Method (FBRAM) to extract rights and obligations from regulatory texts [5, 6, 8]. They applied their methodology to the HIPAA Privacy Rule, but focused on access control rules [6], whereas we model both the access control rules as well as the procedural rules placed on covered entities. Our ontology differs from Breaux et al.'s in that we incorporate all eight of the Hohfeldian concepts in our model, whereas Breaux et al. solely focus on rights and obligations [5, 8]. We previously used the three concepts of rights, obligations, and permissions to model the legal text [23]. We rename permissions to privileges to disambiguate from the use of the term "permission" as used in prior work—Breaux et al. uses permissions to denote rights [5, 6, 8], whereas we denote privileges through our prior use of the term "permission." We modify two techniques from Breaux et al's methodology for use in our methodology presented in Section 4. First, in normative phrase analysis, legal text statements are classified based on the natural language phrases in the statement [5, 8]. As discussed in Section 4.1.1, we expand normative phrase analysis to include all eight of Hohfeld's concepts, not just rights and obligations. Second, we employ rights and obligations balancing to identify implied rights and obligations [8], that is, a right of one group imposes an obligation on others. In this paper, we expand rights and obligation balancing to accommodate additional legal concepts (see Section 4.1.1).

There have been two goal-oriented approaches to legal compliance in the requirements literature. The Ghanavati et al. approach [11, 12] is based on the User Requirements Notation (URN), which is composed of use cases and an i*-like goal notation [43]. They use URN to model business practices and legal texts, and compare the two models using traceability links to identify potential areas of noncompliance [11, 12]. In contrast, we go beyond traceability between models by maintaining traceability across all requirements artifacts back to the legal text, and forwards to the software design and implementation. Ghanavati et al. limit their examination to business practices rather than software requirements, and do not explain how to specify their goal model of the law. The Siena et al. framework, called Nomos [34, 35], is similar to our approach in that they model the law using Hohfeld's eight legal concepts [35], formalizing the concepts using deontic logic. They then extract the concepts from the law and model them using a goal-oriented approach based on i*. However, Siena et al. do not analyze the Hohfeldian concepts individually; instead, they combine each of the correlative concepts into a single predicate [35]. As we will discuss in Section 5, failing to analyze all the concepts can overlook legal requirements. Additionally, Siena et al. do not provide guidance to software engineers about how to extract the concepts from the law, and only demonstrate their methodology on a textbook example. In contrast, we provide prescriptive guidance for extracting legal concepts from the law and have successfully applied our preliminary approach to an existing system [24].

Hassan and Logrippo extract compliance requirements from law using an approach based on the Unified Modeling Language (UML) [15]. Once extracted, they model these requirements in Alloy, a logic-based modeling language. They specify categories of legal statements, such as declarative statements (declaring a fact) and procedural statements (if-then statements) [15]. Their only publication does not explain the extraction process [15] or how to classify legal statements. Using formal methods, like Alloy and description logic, is important for demonstrating that legal compliance methods are automatable, but they are inefficient and too rigid given how regulations evolve [7].

May et al. formalize a section of HIPAA (§164.506) using Privacy APIs and then use the Spin model checker to find loop holes and perform other queries on the law [25]. They use public comments from an earlier version of HIPAA to test their model [25]. Like production rule models, model is queryable. However, their model only covers one section of HIPAA, whereas we have modeled the entire HIPAA Privacy Rule, Part E. Additionally, they use simple flags to signify environmental variables and externally cross-referenced texts [25]. Thus, the software engineer is burdened with the task of manually following and resolving external cross-references.

2.2 Logic Programming and the Law

Logic programming and the law was a popular area of research in the eighties and nineties [1, 2, 31, 32, 33]. Among these efforts, our work is most similar to the ESPLEX project [2] in which the land leasing legislation used by Biagoli et al. is a general regulation impacting multiple domains. Similarly, the HIPAA impacts several domains: healthcare, law enforcement, the correctional system, educational institutions, etc. Biagioli et al. also outline a methodology, albeit by example only, for converting legal texts to production rules and for identifying obligations and permissions in the legal text. Research to date has failed to provide a repeatable, systematic methodology for translating legal texts into production rules, instead using a trial and error methodology [32], whereas others only present their methodology by example [1, 2, 31]. Finally, the methodologies used in legal knowledge based systems have been criticized as being too ad hoc to be reliable or maintainable [39].

Our work differs from prior work in logic programming in a few specific ways. First, the nature of the HIPAA legislation is inherently different from that of legislation used in prior work in production rule modeling of legal texts [1, 31, 32, 33]. The legal documents used in these efforts usually seek to answer a single question. For example, in the British Nationality Act effort [32], the query considered is whether or not an individual is a British citizen. In contrast, the HIPAA Privacy Rule does not have one query that unifies the document. Instead, its broad nature allows many possible queries; for example, queries about access control and the right of notice, the interactions between organizations, etc. Moreover, there are 31 different types of stakeholders mentioned in the HIPAA Privacy Rule alone. HIPAA's broader scope makes it challenging to predict potential queries. We employ Hohfeld's concepts to capture a broader range of potential queries on the model because all are required in order to provide a holistic view of a compliant software system [22]. Second, no prior work has used production rule models to create legally compliant software. Instead, they focused on: improving the understanding of law using production rules [2]; knowledge representation research rather than practical uses of production rule models [1, 32]; aiding law makers in drafting legislation [32]; and legal reasoning [2, 31, 33]. Third, we consider all the Hohfeldian concepts that are expressed in the regulations, not just obligations and privileges. Because rights impose obligations on other parties [8], they are an important source of legal requirements.

Lam et al.'s proof of concept compliance checker determines whether an email message complies with three HIPAA Privacy Rule sections before it is transmitted [20]. To accomplish this, they translated the three sections into a version of Prolog, focusing on paragraphs that address access to Protected Health Information (PHI) [20]. Based on this analysis, they develop a set of eight message characteristics, used to make compliance decisions, but do not justify why these message characteristics are sufficient to determine compliance [20]. In contrast, we specify a variety of legal preconditions beyond Lam et al.'s eight message characteristics [23, 24]. Additionally, they do not provide a means to verify that their formalization provides a correct interpretation of the law [20]. They claim that handling cross-references is easy but fail to show how [20], and researchers have repeatedly noted that cross-references are both challenging and critical in determining legal compliance with confidence [1, 6, 14, 25, 26].

2.3 Expert Systems and Law

An expert system captures the knowledge of a human expert in a particular domain and makes it accessible for non-experts [13, 21]. Like production rules, expert systems also have a knowledge base and an inference engine to make use of that knowledge [30]. Visser et al. provide a methodology, based on the CommonKADS methodology [30], for constructing legal knowledge-based systems that seek to replicate the legal reasoning performed by a lawyer [40, 41]. Bench-Capon describes expert systems as: interactive; based on one narrow domain of expertise; having the ability to reason under uncertain or incomplete information; and capturing "rules of thumb" used by domain experts. Popple defines a legal expert system as a "system that provides answers to legal questions in a form that one would expect from a lawyer" and that "the output from [a legal expert system] should be usable without further legal analysis" [29]. We do not seek to construct an expert system nor do we seek to replace lawyers because legal interpretation is complex, changes over time, and depends on factors beyond the text of a legal document, such as case law, industry standards and current practice, and administrative clarifications.

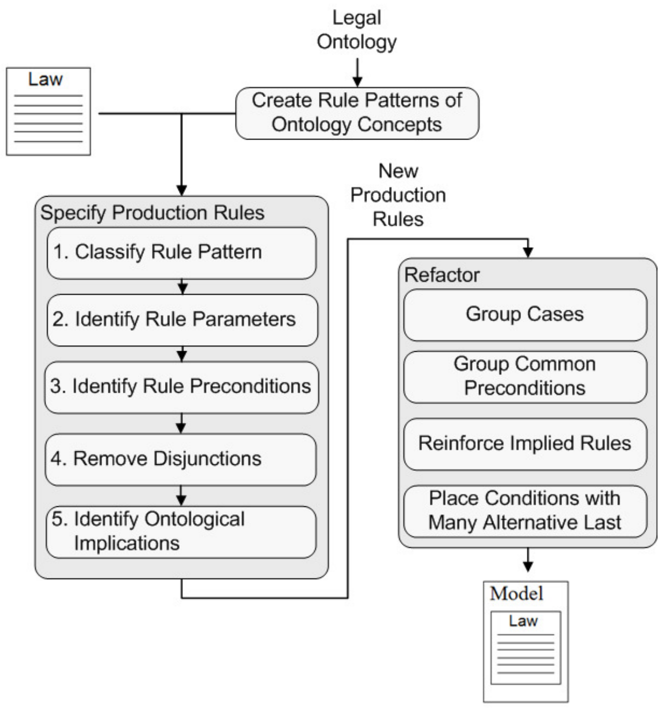
Legal Ontology
Law
Create Rule Patterns of Ontology Concepts
New Production Rules
Specify Production Rules
1. Classify Rule Pattern
2. Identify Rule Parameters
3. Identify Rule Preconditions
4. Remove Disjunctions
5. Identify Ontological Implications
Refactor
Group Cases
Group Common Preconditions
Reinforce Implied Rules
Place Conditions with Many Alternative Last
Model
Law

Fig. 1. Production Rule Modeling Methodology Overview

3 PROLOG PRIMER

We employ Prolog to encode production rules because of its relatively straightforward design and its prior use in the area of legal knowledge representation [1 2, 31, 32, 33]. The syntax of a Prolog rule is:

```
<result> :-
        <condition1>,
        <condition2>, ...
        <conditionN>.
```

Where the symbol `:-` is interpreted as the *if* conditional, the comma symbol is interpreted as logical-and, and the period symbol is interpreted as a full stop (the end of a rule). The result is evaluated to true only if {`condition1`, `condition2`,..., `conditionN`} are evaluated to true. The Prolog rule `father(X,Y) :- male(X), child(Y,X)` is read "X is the father of Y if X is male and Y is the child of X."

In Prolog, an atom is a quoted string, name, or a sequence of special characters (`:-` is one example). A variable signifies a single yet unspecified quantity and begins with a capital letter. A predicate is a relationship between atoms [37]. The production rule model makes use of two built-in Prolog predicates. The `assert(NewFact)` procedure adds a new fact to the knowledge base. Similarly, the `retract(Fact)` procedure removes the first occurrence of the specified fact from the rules base [37].

The strength of Prolog to answer queries comes from two concepts: unification and backtracking [28]. Unification occurs when the inference engine attempts to find a single value to bind to multiple occurrences of a variable. For example, unification would occur if a single value is found for the variable `Org` in the predicates `coveredEntity(Org)` and `healthPlan(Org)`. The inference engine uses backtracking to determine the result of a query. The initial query is treated as a top-level goal, then the engine searches the rules to determine the goal's value through trial and error.

The distinction between Prolog and production rules is the use of working memory, a temporary storage area where intermediate results are stored. With working memory, a result only needs to be computed once, and can then be reused at many stages of computation. Lacking working memory, Prolog recomputes each intermediate result, even if it has been computed before. We add temporary storage to Prolog, however, by adding a working memory predicate, `wm(X)`, where `X` is an element in working memory. Once an intermediate result is computed, we add it to the working memory. When an intermediate result is required, this memory is checked before the result is computed. At the completion of a query, the memory is erased.

4 METHODOLOGY WITH HEURISTICS

In this section, we describe our methodology for creating production rule models of legal texts. Figure 1 displays the Production Rule Modeling methodology. The inputs to the methodology are a legal text and a legal ontology. The methodology consists of a preparatory step, *Create Rule Patterns of Ontological Concepts*, and two activities, *Specify Production Rules* and *Refactoring*. In the preparatory step, the software engineer defines rule patterns for each concept in the inputted legal ontology (Section 4.1). The two activities are described below:

Activity 1: Specify Production Rules
- Step 1. Classify the rule pattern based on the language present in the legal text.
- Step 2. Identify the rule parameters for the pattern classified in step 1.
- Step 3. Identify which preconditions cause a rule to be applied.
- Step 4. Remove disjunctions using case splitting
- Step 5. Identify rules implied by the ontology.

Activity 2: Refactor – Identify patterns in the rules to reduce rule and condition count.

The output of the methodology is a production rule model of the legal text. In the remainder of this section, we describe the legal ontology we use in our case study, heuristics for specifying production rules, and heuristics for refactoring.

We define two terms that we use in this section. First, a *statement* is an independent clause in the legal text. Second, an *actor* is a person, organization, or other entity explicitly defined or referenced in the legal text. Example actors include a covered entity, a doctor, and a health insurance issuer.

4.1 Legal Ontology & Rule Patterns

In this subsection, we present the legal ontology we used to model the law and the Prolog pattern for each concept in the ontology. In our case study, we used the Hohfeldian concepts to model the law. We provide definitions for each of these concepts in Section 2.

Table 1 displays the eight Hohfeldian concepts, the production rule pattern that expresses that concept, the correlative concept, the opposite concept, and the normative phrases used to identify the concept in the Privacy Rule. For each of the Hohfeldian concepts, we specify a production rule pattern during the preparatory step *Create Rule Patterns of Ontological Concepts*. For example, if a covered entity is obligated by §164.512(f)(2)(ii) of the Privacy Rule to not disclose PHI to a third party, we express this obligation as:

```
must(CE,
    not(discloses(CE,
                ThirdParty
                PHI
    '164.512(f)(2)(ii)').
```

To maintain traceability from the legal text to the production rules, we adopt Sherman's solution [33]; we include an additional parameter, `Source`, with each pattern specifying the source of the rule. For instance, the obligation listed above is specified in §164.512(f)(2)(ii) of the HIPAA Privacy Rule.

Each of the Hohfeldian concepts has a correlative concept [16]—one concept implies another concept. For example, if an individual has a right to be notified of the uses and disclosures of his PHI by a covered entity, the covered entity is obligated to provide such a notice. Table 1 lists the concept implied by each concept. We discuss implied rules further in Section 4.2.5. Also, each concept has an opposite concept—an actor cannot hold both a concept and its opposite. Table 1 lists each concept's opposite concept. We use opposite concepts to express exception cases in Section 4.2.3. To classify sections of the legal text, we use normative phrase analysis [8]; we list the normative phrases in Table 1. Immunities lack normative phrases because we did not encounter either of these concepts expressed in the Privacy Rule. We discuss this normative phrase analysis further in Section 4.2.1.

4.2 Heuristics for Specifying Production Rules

In this section, we present heuristics for each of the five steps in the Specify Production Rules activity. We illustrate our methodology and heuristics using concrete examples from our HIPAA Privacy Rule case study. As done in related work [5], we do not include document meta-information such as table of contents, title pages, appendices, or paragraph and section headings in our analysis. Our methodology steps are sequential, but the heuristics for a particular step need not be applied sequentially.

When specifying production rules, one should keep the wording as close to that of the original legal text as possible. For example, in different contexts in the Privacy Rule, the terms "notify", "inform", and "alert" are used. These terms are nearly synonymous, so analysts might be tempted to use a single Prolog predicate to model them. However, there may be some legal nuance that makes these terms different. Therefore, we create the Prolog predicates `notify`, `inform`, and `alert` to model these three Privacy Rule terms, respectively. Consulta-

TABLE 1
HOHFELDIAN CONCEPTS, RULE PATTERNS, AND CLASSIFICATION PHRASES

Concept	Production Rule Pattern	Implied Concept	Opposite Concept	Normative Phrases
right	`right(Actor,Counterparty,Right,Source)`	obligation	no-right	has a/the right to retains the right to
obligation	`obligation(Actor,Obligation,Source)`	right	privilege	must is required to shall may not is prohibited is subject to
privilege	`privilege(Actor,Privilege,Source)`	no-right	obligation	may may elect not to is not required to requirement does not apply is permitted to at the election of is not subject to
no-right	`noRight(Actor,NoRight,Source)`	privilege	right	does not have a right to
power	`power(Actor,LegalRelation,Power,Source)`	liability	disability	authorize termination of must obtain an authorization may revoke may terminate
liability	`liability(Actor,Liability,Source)`	power	immunity	provide that <actor> will/must obtain assurances that
immunity	`immunity(Actor,Counterparty,Power,Source)`	disability	liability	(none found)
disability	`disability(Actor,Power,Source)`	immunity	power	may not authorize

tion with legal domain experts can clarify nuances in meaning of these terms.

Otto and Antón document the challenge of ambiguity in legal texts [26]. When modeling the law, we do not attempt to resolve ambiguous terms such as "reasonable," "sufficient," or "adequate". The law uses such language to allow legal domain experts to make determinations based on the context of a particular case. In our model, we introduce the predicates `reasonable`, `sufficient`, and `adequate` to model these terms, respectively , and do not attempt to determine what actions are reasonable, sufficient, or adequate. Case law, industry best practices, and Federal agency clarification documents can be used to clarify these terms, but are outside the scope of our model.

Throughout the remainder of this section, we will illustrate our heuristics using a concrete example from HIPAA, §§164.510(a)(1)(i)(A-D), displayed in Figure 2.

Preparatory Heuristic 1 (PH_1): Split continuations into separate rules

Oftentimes, a legal statement is broken across a list. For example, the legal statements in Figure 2 are split across multiple sections of the legal text. This is called a continuation and results in multiple legal rules [5]. The text in sections (1) and (i) of the legal text are prepended to the text in subsections (A), (B), (C), and (D), resulting in four legal rules:

1. "Except when an obligation is expressed in accordance with paragraphs (a)(2) or (a)(3) of this section, a covered healthcare provider may use the following PHI to maintain a directory of individuals in its facility: the individual's name."
2. "Except when an obligation is expressed in accordance with paragraphs (a)(2) or (a)(3) of this section, a covered healthcare provider may use the following PHI to maintain a directory of individuals in its facility: the individual's location in the covered health care provider's facility."
3. "Except when an obligation is expressed in accordance with paragraphs (a)(2) or (a)(3) of this section, a covered healthcare provider may use the following PHI to maintain a directory of individuals in its facility: the individual's condition described in general terms that does not communicate specific medical information about the individual."
4. "Except when an obligation is expressed in accordance with paragraphs (a)(2) or (a)(3) of this section, a covered healthcare provider may use the following PHI to maintain a directory of individuals in its facility: the individual's religious affiliation."

4.2.1 Classify Rule Pattern Heuristics

Classifying statements in the legal text as one of the rule patterns (presented in Section 4.1) requires using two classification heuristics.

Classification Heuristic 1 (Cla_1): Classify using normative phrase analysis

We use normative phrase analysis to classify rules according to the natural language phrases used in the legal text [8]. Table 1 in Section 4.1 displays the normative phrases we use to classify rule patterns. For example, we classify the legal statement in Figure 2 as a privilege because of the natural language phrase "may." When multiple normative phrases are identified, we use the most inclusive normative phrase. For instance, §164.502(a) states that "a covered entity may not use or disclose PHI…" We classify this statement as an obligation (using the phrase "may not") instead of a privilege (using only the phrase "may").

Cla_2: Add obligations for "only" phrase

When we encounter the term "only", we add an obligation to the model. For example, §164.502(d)(2)(ii) states that if deidentified information is somehow reidentified, a "covered entity may use or disclose such reidentified information only as permitted or required" by HIPAA. This statement expresses a covered entity's privilege to use or disclose reidentified information in the same way it uses or discloses other PHI. The statement also represents that a covered entity may only use the reidentified information in such a manner. Thus, we add the obligation "a covered entity must not use or disclose reidentified information in violation of HIPAA" to the model.

(1) Except when an objection is expressed in accordance with paragraphs (a)(2) or (3) of this section, a covered health care provider may:
(i) Use the following protected health information to maintain a directory of individuals in its facility:
(A) The individual's name;
(B) The individual's location in the covered health care provider's facility;
(C) The individual's condition described in general terms that does not communicate specific medical information about the individual; and
(D) The individual's religious affiliation

Fig. 2. §§164.510(a)(1)(i)(A-D) of the HIPAA Privacy Rule

After the *Classify Rule Pattern* step, the production rule expressing section (A) in Figure 2 is `privilege(_,_,_)`.

4.2.2 Identify Rule Parameters Heuristics

Identifying the parameters for the rule patterns classified in the previous step entails the use of four identification parameters heuristics.

Identify Rule Parameters Heuristic 1 (Par_1): Identify the actor who is subject to rule

We identify the actor who is subject to a rule by identifying the subject of the statement. For example, the actor subject to the privilege in section (1)(i)(A) in Figure 2 is a covered healthcare provider. Where possible, we identify the most specific actor. For example, HIPAA defines both healthcare providers and covered healthcare providers. We identify the actor in section (1)(i)(A) as a covered healthcare provider because it is the more specific than healthcare provider.

Par_2: Identify the legal relation affected by a power

A power grants an actor the ability to change a legal relation. We identify the relation the actor is able to change in this heuristic. For example, §164.504(e)(2)(iii) authorizes a covered entity to terminate a contract with a business associate if the associate has violated the agreed-upon contract. The legal relation the covered entity has the power to change is the legal contract between the covered entity and the business associate.

Par_3: Identify the rule action

The rule action is the action the actor has a right to perform, the action the actor is obligated to perform, etc. It is the portion of the legal statement immediately following the normative phrase identified in the Classify Rule Pattern step (Section 4.2.1). For example, in Figure 2, section (A), the action the covered healthcare provider is allowed to perform is "use the following PHI to maintain a directory of individuals in its facility: the individual's name."

Par_4: Identify the rule source

The source of each rule is the full section reference. In the case of continuations, the source is the lowest subsection in the legal document hierarchy that is a part of the legal statement. For example, the source for section (A) in Figure 2 is `'164.510(a)(1)(i)(A)'`.

After step 2, Identify Rule Parameters, the production rule expressing section (A) in Figure 2 is:

```
privilege(CHCP,
          for(uses(CHCP,name(Individual)),
              maintains(CHCP,directory)),
          '164.510(a)(1)(i)(A)')
```

4.2.3 Identify Rule Preconditions Heuristics

This step entails identifying the legal preconditions that cause a rule to be applicable, and uses four heuristics.

Identify Rule Preconditions Heuristics (Pre_1): Identify type-checking preconditions

Prolog is typeless; thus, we add preconditions to check the types of actors, objects, and relations used in the rule. For example, for the privilege in section (A) in Figure 2, we express that an organization is a covered healthcare provider using the Prolog predicate `isCHCP(CHCP)`.

Pre_2: Identify preconditions expressing exceptions

Exceptions are expressed using the natural language phrases "except", "is not effective under", "other than", "does not apply to", "notwithstanding", and "unless." For example, the privilege in section (A) of Figure 2 does not apply if the individual has expressed an objection according to (a)(2) or (a)(3). We perform two actions when we encounter an exception. First, a we add a precondition that is a negation of the exception condition to the production rule. For the example text, the precondition "if the individual did not express an objection according to (a)(2) or (a)(3)" is added to the privilege "a covered healthcare provider may use the following PHI to maintain a directory of individuals in its facility: the individual's location in the covered health care provider's facility." Second, we create a new rule expressing the exception (opposite) case. For the example text, a new production rule is created expressing that a covered healthcare provider may not (is obligated to not) use an individual's name for their directory if the individual has expressed an objection. Each concept's opposite is listed in Table 1 in Section 4.1.

Pre_3: Identify preconditions using precondition keywords

We identify preconditions in the legal statement that follow the phrases "if," "when," "whenever," "that", "who", "whose", "to the extent that", and "provided that", as well as temporal phrases such as "after", "prior", and "for as long as". For example, in section (a)(3) of §164.510, a covered healthcare provider may use an individual's name for their directory, "if the opportunity to object to uses or disclosures to uses or disclosures required by paragraph (a)(2) of this section cannot practicably be provided because of the individual's incapacity". We add this precondition to the production rule that expresses (a)(3).

Pre_4: Identify preconditions from cross-references

Cross-references must be carefully followed to obtain additional preconditions. For example, in Figure 2, we add the precondition "the objection is in accordance with (a)(2) or (a)(3)" to the rules for (A). The analyst must follow this cross-reference and determine what preconditions are relevant—which can introduce ambiguity, because cross-references may refer to only part of the referenced statement [5].

Preconditions may be added by a cross-referencing statement in the legal text, that is, an entirely separate legal statement may place preconditions on a legal statement. For example, consider HIPAA §164.512(f)(6)(ii):

> If a covered health care provider believes that the medical emergency described in paragraph (f)(6)(i) of this section is the result of abuse, neglect, or domestic violence of the individual in need of emergency health care, paragraph (f)(6)(i) of this section does not apply and any disclosure to a law enforcement official for law enforcement purposes is subject to paragraph (c) of this section.

This statement adds an exception precondition to paragraph (f)(6)(i). Thus, we must revisit the production rules associated with (f)(6)(i) and add an exception precondition to those rules using PrH_2.

We performed our analysis of the Privacy Rule sequentially, beginning with the first section. When we encountered a forward cross-reference, for example, (a)(1)(i)(A) in Figure 2 contains a forward reference to (a)(2) and (a)(3). We tabled these references until we modeled the referenced sections.

After step 3, *Identify Rule Preconditions Heuristic*, the production rules expressing section (A) in Figure 2 are:

```
privilege(CHCP,
          for(uses(CHCP,name(Individual)),
              maintains(CHCP,directory)),
          '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  not(s164_510a1_exception(CHCP,Individual)).
obligation(CHCP,
        not(for(uses(CHCP,name(Individual)),
            maintains(CHCP,directory)),
        '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  s164_510a1_exception.
```

4.2.4 Remove Disjunctions Heuristics

This step entails identifying and removing disjunctions through case splitting. By convention, Prolog rules are expressed using only logical-and.

Remove Disjunctions Heuristic 1 (Dis_1): Split "or"

We case split the natural language phrase "or" into multiple rules. For example, in section (A) of Figure 2, there two cases: (1) when an objection is expressed according to (a)(2), and (2) when an objection is expressed according to (a)(3).

Dis_2: Split logical-or's masquerading as "and"

As identified by Breaux, legal texts contain logical ambiguity [5], where the legal text will use the word "and," but in context, it is a logical-or. For example, HIPAA §164.504(e)(2)(i)(A) states:

> The contract [between the covered entity and business associate] may permit the business associate to use and disclose PHI for the proper management and administration of the business associate

This statement allows the business associate to "use and disclose PHI" for their administrative tasks. If we interpret this to mean a logical-and, then the business associate has no option to use the PHI without disclosing it as well. However, if we interpret this to be a logical-or, then the business associate may use the information for their own internal processes without disclosing it to a third party. In this heuristic we split this "and" using case splitting, just as we would an "or."

After step 4, *Remove Disjunctions*, the production rules expressing section (A) in Figure 2 are:

```
privilege(CHCP,
          for(uses(CHCP,name(Individual)),
              maintains(CHCP,directory)),
          '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  not(s164_510a2_objection).
privilege(CHCP,
          for(uses(CHCP,name(Individual)),
              maintains(CHCP,directory)),
          '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  not(s164_510a3_objection).
obligation(CHCP,
        not(for(uses(CHCP,name(Individual))),
            maintains(CHCP,directory)),
        '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  s164_510a2_objection.
obligation(CHCP,
        not(for(uses(CHCP,name(Individual))),
            maintains(CHCP,directory)),
        '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  s164_510a3_objection.
```

4.2.5 Identify Ontological Implications

Depending on the ontology used to model the production rules, the software engineer may infer additional facts to add to the database. Using the Hohfeldian concepts, each concept implies its correlative concept. The implied concepts are listed in Table 1 in Section 4.1. Implied rules are important to obtain, because codifying them increases requirements coverage and provides important traceability information to aid in establishing due diligence [8]. Breaux et al. introduced rights and obligations balancing identify implied rights and obligations in HIPAA. We expand rights and obligations balancing to balancing all implied rules and add them during this step. For example, the privilege in section (A) of implies a no-right on the individual, whereas the obligation expressed in section (A) implies a right on the individual.

The production rules after we complete the *Specify Production Rules* activity that express section (A) are displayed the Appendix.

4.3 Refactoring Heuristics

After translating each legal statement, we check the entire rule base for refactoring opportunities. Refactoring is an optional supplemental activity; the rules generated during the *Specify Production Rules* are a complete production rule model. Fowler identifies advantages of refactoring, including improving software design, making software easier to understand, aiding in debugging, and improving software engineer productivity [10]. When refactoring, we seek to identify common patterns to reduce rule and condition count. Fewer rules and conditions can lead to a more readable rules base. Additionally, we refactor some rules to increase performance. In this section, we review the refactoring techniques we employed in our case study.

Refactoring Heuristic 1(Ref$_1$): Group common cases

This heuristic reduces the rule count by grouping common cases that were split during the Remove Disjunctions step (Section 4.2.4). For example, in section (A) of Figure 2, the two exception cases, (a)(2) and (a)(3) can be grouped using one predicate, for example, `s164_510a2_or_a3_objection`, and the preconditions modified accordingly. We discuss this heuristic in our prior work [23].

Ref$_2$: List conditions with many alternatives last

To improve Prolog search efficiency, Bratko suggests to stop processing pointless alternatives and avoid needless backtracking [4]. Conditions with many alternatives can cause both pointless alternatives and needless backtracking; in this refactoring technique, we move conditions with many alternatives to the end of the conditions list to improve efficiency.

Ref$_3$: Reduce logic load for common cases

In this heuristic, we add additional rules to reduce the logic load for common cases, using transitivity. For example, consider a production rule model that contains the rules `A :- B` and `B :- C`. If `C` is a precondition that is used in many production rules, we reduce the number of subgoals the inference engine must prove by asserting the rule `A :- C`.

Ref$_4$: Group common preconditions

In our previous study, many rules shared a set of common preconditions [23]. To reduce precondition count, we group these preconditions into a single predicate that is true when the preconditions are true. We then replace the common preconditions with the new predicate.

5 Discussion

This section discusses our experiences in modeling the HIPAA Privacy Rule. The Privacy Rule comprises 45 CFR Part 160 and Part 164, Subparts A and E [38]. Our analysis focuses on Subpart E, which describes privacy requirements for covered entities. The only portion of Part 160 that we model is §160.103, which contains defines key terms used throughout the Privacy Rule. We do not model the remainder of Part 160, which describes enforcement, penalties, and other requirements for governmental agencies, or Part 164, Subpart A, which describes applicability and covered entities' legal options under HIPAA.

The HIPAA Privacy Rule, Part E comprises §§164.500-164.534, a total of 37 printed pages. Our model contains 2,258 rules and is written using the SWI-Prolog[3] implementation. Section 164.534 contains the compliance dates for various covered entities. Because all the compliance dates have passed, we

[3] http://www.swi-prolog.org/

did not model this section. It took approximately 294 man-hours to construct a production rule model of the Privacy Rule. To the best of our knowledge, based on a survey of over 150 publications [26], we are the first to model an entire Federal regulation. As mentioned in Section 2, Breaux and Antón analyzed the Privacy Rule, but focused solely on access control rules [6].

Table 2 displays the number of production rules expressed using each of the concept patterns. Disjunction splitting and refactoring impacts the number of production rules that express each of the concepts. The majority of the concepts identified in our case study were privileges and obligations. In the Privacy Rule, the majority of these privileges and obligations are placed upon covered entities, whereas, individuals hold rights and the majority of the implied no-rights. We did not encounter any immunities expressed directly in the Privacy Rule; instead, they are implied by disabilities.

The Hohfeldian concepts allow us to identify legal requirements that we failed to identify in our previous study [23]. For example, section 164.520 of the HIPAA Privacy Rule regulates a covered entity's notice of privacy practices. Section 164.520(a)(3) contains two legal statements: 1) "An inmate does not have a right to notice under this section", and 2) "the requirements of this section do not apply to a correctional institution that is a covered entity." Statement one is a no-right, indicated by the phrase "does not have a right to." The second statement is a privilege, indicated by the phrase "requirements…do not apply." In our previous study, we failed to classify the first statement as a no-right, instead classifying all of §164.520(a)(3) as a privilege, because our methodology did not include all the Hohfeldian concepts.

We classify and model all of the Hohfeldian concepts individually. Identifying the Hohfeldian concepts individually is important for software engineers to capture for three reasons. First, they establish exceptions and priorities between legal requirements, which Breaux and Antón found to add additional constraints and requirements to legal rules [6]. Second, capturing the concepts implied by the additional Hohfeldian concepts is important for legal compliance. For example, no-rights imply privileges on the counterparty. In the no-right from §164.520(a)(3), the implied privilege is "a correctional institution is not required to provide a notice of privacy practices to inmates." Identifying this privilege could save a correctional institution the expense of notifying their inmates and allow software developers to not have to build this functionality into their systems. On the other hand, software developers may have to build this functionality into their EHRs, if they are developing systems that will be deployed in other covered entities in addition to correctional institutions. Third, engineers can identify inconsistencies in the law using the opposite concepts. For example, the law is inconsistent if an individual can hold both a no-right and the opposite right.

In the Privacy Rule, liabilities are placed on third parties, e.g., business associates of covered entities. Business associates can include IT, legal, and accounting firms [18]. For instance, §164.504(e)(2)(ii)(B) states that business associates is liable to safeguard the information entrusted to them. It is important to model these liabilities, because the ARRA expanded the requirements of HIPAA for business associates, requiring them to comply with the Privacy Rule [18].

Breaux identifies various kinds of ambiguity in legal texts, such as logical ambiguity, attributive ambiguity, referential ambiguity, and under-specification. During our case study, we identified an addition kind of ambiguity: structural ambiguity. Structural ambiguity occurs when different levels of a hierarchical document use the same symbols to denote headings. For example, Figure 3 displays an example of structural ambiguity. There are two top-level sections in Figure 3, labeled (a) and (b). There are two subsections in (b), labeled (1) and (2). Section (2) also has subsections, labeled (a) and (b). The ambiguity is section (c), which has been bolded in the figure. Heading (c) could be interpreted as a top-level section or as a subsection of (b)(2). This ambiguity exists in the Privacy Rule because the Privacy

TABLE 2
CONCEPT AND RULE COUNTS

Concept	Explicit	Implied
Rights	9	258
Obligations	258	9
Privileges	177	2
No-Rights	2	177
Powers	7	29
Liabilities	29	7
Immunities	0	2
Disabilities	2	0

TABLE 3
MOST FREQUENT NORMATIVE PHRASES

must	215
may	154
may not	26
provide that <actor> will	25
has a/the right to	11
is permitted to	7
may revoke	6
is not required to	4

TABLE 4
HEURISTIC USE

Concept	Cla_1	Cla_2	Par_1	Par_2	Par_3	Par_4	Pre_1	Pre_2	Pre_3	Pre_4	Dis_1	Dis_2
right	12	0	12	0	12	12	20	12	4	10	5	1
obligation	249	4	249	0	249	249	356	35	100	154	88	6
privilege	175	13	175	0	175	175	358	25	165	129	120	7
no-right	1	0	1	0	1	1	5	0	4	4	0	0
power	5	0	5	5	5	5	7	3	5	0	2	0
liability	27	0	27	0	27	27	53	3	8	6	11	1
immunity	0	0	0	0	0	0	0	0	0	0	0	0
disability	2	0	2	0	2	2	5	0	1	1	3	0

Rule reuses numerals at different levels of the hierarchy, as well as the symbol i (the lower case letter) and i (the first Roman numeral). For example, §164.514(e)(4)(ii)(C) of the Privacy Rule contains subsections labeled (1)-(4). The next section is labeled (5) and could be interpreted as §164.514(e)(4)(ii)(C)(5) or §164.514(e)(5). To resolve this ambiguity, software engineers must use other context clues, such as continuations or surrounding section headings, to determine the correct interpretation. In the Privacy Rule example, section (5) is a level 5 heading, because the next section is (iii). If section (5) was a level 2 heading, the next section would be (i).

Recall we introduced thirteen heuristics for the Specify Production Rules activity. We applied the heuristics to the HIPAA Privacy Rule as follows. When we applied the preparatory heuristic, PH_1 *Split Continuations into Separate Rules*, we identified 157 continuations. The remaining heuristics are applied after we resolve these continuations. Table 3 displays the most frequent normative phrases in the Privacy Rule. The normative phrases not listed in Table 3 occurred only two times or less in the Privacy Rule. Table 4 lists the number of times we applied our heuristics, and the concept to which the heuristic was applied.

As a result of our analysis of the HIPAA Privacy Rule, we discovered that privileges are more complex than obligations. In Table 4, we see that we classify 74 fewer privileges than obligations. Despite of this, privileges have the same number of type-checking preconditions as obligations, 65 more preconditions identified using precondition keywords, only 25 fewer cross-reference preconditions, 32 more disjunctions, and one more disjunction masquerading as a logical-and. This suggests that the question "what must an organization do to comply with the law?" is less complex, though still challenging, than the question "what is an organization allowed to do under the law?"

6 Threats to Validity

For any case study, it is important to consider any threats to validity. Thus far, we have only examined one regulation. This threatens the external validity of our study; external validity addresses the ability of a case study's findings to be generalized to other studies [42]. Future studies examining legal texts in different domains will further validate the methodology presented herein. We make no causal inferences, there are no threats to the internal validity of our study [42].

The case study was performed by researchers who have prior knowledge of the HIPAA regulation. Allowing that previous knowledge of the regulation guide the modeling process is a threat to our study's reliability—the ability to repeat the case study and produce similar results [42]. To mitigate this threat to reliability, we carefully followed the methodology presented in Section 4.

Construct validity addresses the degree to which a case study is in accordance with the theoretical concepts used [42]. Three ways to reinforce construct validity are: use multiple sources of evidence, establish a chain of evidence, and have key informants review draft case study reports [42]. While we only used one source of evidence for this case study, the HIPAA Privacy Rule, future studies will validate and refine the heuristics presented herein. To establish a chain of evidence, we carefully documented the techniques we used when modeling the Privacy Rule; these techniques became the heuristics presented in Section 4. Finally, our draft case study report was reviewed by several ThePrivacyPlace[4] members.

7 Summary and Future Work

In this paper, we presented a methodology for developing production rule models, and a set of heuristics for specifying the models. We developed the methodology and heuristics through a case study modeling the HIPAA Privacy Rule.

Prior researchers have identified cross-referencing as a challenging problem for software engineering [1, 8, 26], because engineers must carefully follow cross-references to obtain additional legal requirements, exceptions, conditions, and priorities. A simple classification of cross-references is internal and external cross-references. An internal cross-reference is a citation from one portion of a legal text to another portion of the same text. An external cross-reference is a citation from one legal text to separate legal text. During the course of our study, we found 594 internal cross-references and 88 external cross-references to 46 distinct legal texts. Determining external cross-references' impact on legal compliance and developing methods to resolve these references is a major area of future work.

(a) Li Europan lingues es members del sam familie. Lor separate existentie es un
(b) Myth por sciente
(1) Musica, sport,
(2) Etc, litot Europa usa li sam vocabulary. Li lingues differe solemn in
(a) Li grammatical
(b) Li Pronunciation e li plu common vocabules.
(c) Omnicos directe al desirabilite

Fig. 3. Example of Structural Ambiguity

In addition to examining cross-references, we are planning a study to develop metrics and measure the accuracy of our models. We are eliciting EHR usage scenarios. Using these scenarios, we are going to perform two analyses. First, we will query our model for a covered entity's rights, obligations, privileges, etc. Second, legal domain experts will analyze the scenarios and provide us with the same analysis. Comparing these two analyses, we will be able to measure the accuracy of our production rule model.

Currently, we are using SWI-Prolog's command line interface. Future work includes development of a tool with a graphical user interface. Using this tool, we plan to measure the ability of engineers, who are unfamiliar with the law, to model legal texts using our methodology.

Appendix

```
privilege(CHCP,
          for(uses(CHCP,name(Individual)),
             maintains(CHCP,directory)),
          '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  not(s164_510a2_objection).
implied(noRight(Individual,
          right(Individual,
             CHCP
             not(for(uses(CHCP,
                    name(Individual)),
```

[4] www.theprivacyplace.org

```
                maintains(CHCP,directory))),
                _),
            '164.510(a)(1)(i)(A)')) :-
  isCHCP(CHCP),
  not(s164_510a2_objection).
privilege(CHCP,
            for(uses(CHCP,name(Individual)),
                maintains(CHCP,directory),
            '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  not(s164_510a3_objection).
implied(noRight(Individual,
            right(Individual,
                CHCP
                not(for(uses(CHCP,
                        name(Individual)),
                maintains(CHCP,directory))),
                _),
            '164.510(a)(1)(i)(A)')) :-
  isCHCP(CHCP),
  not(s164_510a3_objection).
obligation(CHCP,
            not(for(uses(CHCP,
                name(Individual))),
                maintains(CHCP,directory)),
            '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  s164_510a2_objection.
implied(right(Individual,
            CHCP
            not(for(uses(CHCP,
                name(Individual)),
                maintains(CHCP,directory))),
            '164.510(a)(1)(i)(A)')) :-
  isCHCP(CHCP),
  s164_510a2_objection.
obligation(CHCP,
            not(for(uses(CHCP,
                name(Individual))),
                maintains(CHCP,directory))),
            '164.510(a)(1)(i)(A)') :-
  isCHCP(CHCP),
  s164_510a3_objection.
implied(right(Individual,
            CHCP
            not(for(uses(CHCP,
                name(Individual)),
                maintains(CHCP,directory))),
            '164.510(a)(1)(i)(A)')) :-
  isCHCP(CHCP),
  s164_510a3_objection.
```

ACKNOWLEDGMENT

This work was partially funded by NSF ITR grant #0325269 and NSF Cyber Trust grant #0430166. We thank H. Gu, G. Kaur, A. Massey, M. Singh, A. Villanes, and J. Young for their comments and feedback.

REFERENCES

[1] T.J.M. Bench-Capon, G.O. Robinson, T.W. Routen, M.J. Sergot, "Logic Programming for Large Scale Applications in Law: A Formalisation of Supplementary Benefit Legislation", *Proc. of the 1st ACM Intl. Conf. on Artificial Intelligence and Law*, Boston, 1987, pp. 190-198.

[2] C. Biagioli, P. Mariani, D. Tiscornia, "Esplex: A Rule and Conceptual Model for Representing Statutes", *Proc. of the 1st ACM Intl. Conf. on Artificial Intelligence and Law*, Boston, 1987, pp. 240-251.

[3] R.J. Brachman, and Levesque, H.J., *Knowledge Representation and Reasonin*g, Elsevier, 2004.

[4] I. Bratko, *Prolog: Programming for Artificial Intelligence*, Addison-Wesley, 2001, 3rd ed.

[5] T.D. Breaux, "Legal Requirements Acquisition for the Specification of Legally Compliant Information Systems", Ph.D. Dissertation, North Carolina State University, 2009.

[6] T.D. Breaux, A.I. Antón, "Analyzing Regulatory Rules for Privacy and Security Requirements", *IEEE Trans. on Software Engineering*, 34(1), Jan.-Feb. 2008, pp. 5-20.

[7] T.D. Breaux, A.I. Antón, J. Doyle, "Semantic Parameterization: A Process for Modeling Domain Descriptions", *ACM Trans. on Software. Engineering Methodologies*, (In Press) 2009.

[8] T.D. Breaux, M.W. Vail, A.I. Antón, "Towards Regulatory Compliance: Extracting Rights and Obligations to Align Requirements with Regulations", *Proc. of the 14th IEEE Intl. Requirements Engineering Conf.*, Minneapolis, 2006, pp. 46-55.

[9] *2008 Global Information Survey*, Ernst & Young, 2008.

[10] M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 2000.

[11] S. Ghanavati, D. Amyot, L. Peyton, "Towards a Framework for Tracking Legal Compliance in Healthcare", *Proc. of the 19th Intl. Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, pp. 218-232.

[12] S. Ghanavati, D. Amyot, L. Peyton, "Compliance Analysis Based on a Goal-Oriented Requirement Language Evaluation Methodology", *Proc. of the 17th IEEE Intl. Conf. on Requirements Engineering*, Atlanta, 2009, pp. 133-142.

[13] G. Greenleaf, A. Mowbray, A.A. Tyree, "Expert Systems in Law: The Datalex Project", *Proc. of the 1st ACM Intl. Conf. on Artificial Intelligence and Law*, Boston, May 1987, pp. 9-17.

[14] M. Hamdaqa, A. Hamou-Lhadj, "Citation Analysis: An Approach for Facilitating the Understanding and the Analysis of Regulatory Compliance Documents," *Proc. of the 6th Intl. Conf. on Information Technology: New Generations*, Las Vegas, 2009, pp. 278-283.

[15] W. Hassan, L. Logrippo, "A Governance Requirements Extraction Model for Legal Compliance Validation", *Proc. of the 2nd Intl. Workshop on Requirements Engineering and Law*, Atlanta, 2009.

[16] W.N. Hohfeld, "Some Fundamental Legal Conceptions as Applied in Judicial Reasoning", *The Yale Law Journal*, 23(1), Nov. 1913, pp. 16-59.

[17] A.K. Jha et al, "Use of Electronic Health Records in U.S. Hospitals", *The New England Journal of Medicine*, 360(16), Apr. 16, 2009.

[18] G.M. Kastel, "ARRA Changes to HIPAA Include Requirements Restrictions for Business Associates", Feb. 20, 2009, <http://www.faegre.com/showarticle.aspx?Show=8969 >.

[19] B. Krebs, "ChoicePoint Breach, Exposed 13,750 Consumer Records", *The Washington Post*, Oct. 19, 2009, <http://voices.washingtonpost.com/securityfix/2009/10/choicepoint_breach_exposed_137.html>.

[20] P.E. Lam, J.C. Mitchell, S. Sundaram, "A Formalization of HIPAA for a Medical Messaging System", *Proc. of the 6th International Conference on Trust, Privacy & Security in Digital Business*, Linz, 2009.

[21] J. Liebowitz, "Expert Systems in Law: A Survey and Case Study", *Telematics and Informatics*, 3(4), 1986, pp. 263-271.

[22] A.K. Massey, P.N. Otto, L.J. Hayward, A.I. Antón, "Evaluating Existing Security and Privacy Requirements for Legal Compliance", In Press: *Requirements Engineering Journal*, Springer-Verlag, 2009.

[23] Maxwell, J.C., Antón, A.I., "Developing Production Rule Models to Aid in Acquiring Requirements from Legal Texts", *Proc. of the 17th Intl. IEEE Requirements Engineering Conf.*, Atlanta, 2009, pp. 101-110.

[24] Maxwell, J.C., Antón, A.I., "Validating Existing Requirements for Compliance with Law Using a Production Rule Model" *Proc. of the 2nd Intl. IEEE Workshop on Requirements Engineering and the Law*, Atlanta, 2009, pp. 1-6.

[25] M.J. May, C.A. Gunter, I. Lee, "Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies", *19th IEEE Computer Security Foundations Workshop*, pp. 85-97, 2006.

[26] P.N. Otto, A.I. Antón, "Addressing Legal Requirements in Requirements Engineering", *Proc. of the 15th IEEE International Requirements Engineering Conference*, New Dehli, 2007, pp. 5-14.

[27] P.N. Otto, A.I. Antón, D.L. Baumer, "The Choicepoint Dilemma: How Data Brokers Should Handle the Privacy of Personal Information", *IEEE Security and Privacy*, 5(5), Sep.-Oct. 2007, pp. 15-23.

[28] N.P. Padhy, *Artificial Intelligence and Intelligent Systems*, Oxford University Press, 2005.

[29] J. Popple, "Legal Expert Systems: The Inadequacy of a Rule-Based Approach", *The Australian Computer Journal*, 23(1), Feb. 1991, pp. 11-16.

[30] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga, *Knowledge Engineering and Management: The CommonKADS Methodology*, MIT Press, 2000.

[31] M.J. Sergot, A.S. Kamble, K.K. Bajaj, "Indian Central Civil Service Pension Rules: A Case Study in Logic Programming Applied to Regulations", *Proc. of the 3rd ACM Intl. Conf. on Artificial Intelligence and Law*, Oxford, 1991, pp. 118-127.

[32] M.J. Sergot, F. Sadri, A. Kowalski, F. Kriwaczek, P. Hammond, H.T. Cory, "The British Nationality Act as a Logic Program", *Comm. of the ACM*, 29(5), May 1986, pp. 370-386.

[33] D.M. Sherman, "A Prolog Model of the Income Tax Act of Canada", *Proc. of the 1st ACM Intl. Conf. on Artificial Intelligence and Law*, Boston, 1987, pp. 127-136.

[34] A. Siena, J. Mylopoulos, A. Perini, A. Susi, "The Nomos Framework: Modelling Requirements Compliant with Laws", Technical Report, TR-0209-SMSP, 2009.

[35] A. Siena, A. Perini, A. Susi, J. J. Mylopoulos, "A Meta-Model for Modelling Law-Compliant Requirements", *Proc. of the 2nd Intl. Workshop on Requirements and Law*, Atlanta, 2009.

[36] E. Singer, "A Big Stimulus Boost for Electronic Health Records", *Technology Review*, Feb. 20, 2009.

[37] L. Sterling, and E. Shapiro, *The Art of Prolog: Advanced Programming Techniques,* MIT Press, 1994, 2nd ed.

[38] U.S. Dept. of Health and Human Services, "Standards for Privacy of Individually Identifiable Health Information—Part 164, Subpart E," Federal Register, 68(34), Feb. 2003, pp. 8334-8381.

[39] A. Valente, *Legal Knowledge Engineering: A Modeling Approach*, in Frontiers in Artificial Intelligence and Applications, Vol. 30, IOS Press, 1995.

[40] P.R.S. Visser, T.J.M. Bench-Capon, and J. van den Herik, "A Method for Conceptualising Legal Domains: An Example from the Dutch Unemployment Benefits Act", *Artificial Intelligence and Law*, 5(3), Sep. 1997, pp. 207-242.

[41] P.R.S. Visser, R.W. van Kralingen, T.J.M. Bench-Capon, "A Method for the Development of Legal Knowledge Systems", *Proc. of the 6th ACM Intl. Conf. on AI and Law*, Melbourne, Australia, 1997, pp. 151-160.

[42] R.K. Yin, *Case Study Research: Design and Methods*, in Applied Social Research Methods Series, Vol. 5, 2003, 3rd ed.

[43] E. Yu, "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering", *Proc. of the 3rd IEEE Intl. Symposium on Requirements Engineering*, Annapolis, Jan. 1997, pp. 6-10.