Acquiring Software Compliance Artifacts from Policies and Regulations

Travis D. Breaux and Annie I. Antón
Department of Computer Science
North Carolina State University
Raleigh, NC, 27695-8206 USA
{tdbreaux,aianton}@ncsu.edu

Abstract

Policies and government regulations impose restrictions on information practices in healthcare and finance. These restrictions govern the use and disclosure of information that spans organizations and their business practices. To comply with policies and the law, organizations must demonstrate that they have verifiable procedures in-place to implement these restrictions. To this end, we present techniques that software engineers can use to systematically acquire software artifacts from natural language policies and regulations based on our in-depth analysis of the U.S. Health Insurance Portability and Accountability Act[1] (HIPAA). The techniques apply semantic primitives to regulatory statements to express class structures using the Z notation. From these structures, software engineers distinguish between necessary and discretionary software requirements and acquire the following software artifacts: specifications for transactions including interfaces between software and business processes; data schemas and data maintenance requirements; and event-based test cases for ensuring that systems comply with policies and regulations.

1. Introduction

In the United States, government regulations require organizations to develop policies and procedures that comply with the law. These regulations are specified in complex and ambiguous legalese (English) and generally prescribe business practices. The increasing reliance on software systems to support these practices presents software engineers and system administrators with the daunting challenge of interpreting regulations to determine if their systems comply. The fact that new privacy regulations that govern information practices are primarily specified by individuals who are domain experts in non-engineering fields, such as medicine, finance or law, further complicates the compliance landscape. These domain experts are often not familiar with accepted requirements and software engineering practices. In our experience, this contributes to the plethora of ambiguous and complex regulations that are difficult if not impossible to test, or that are potentially intractable in software systems.

To address these challenges, we are developing a methodology for analyzing policies and regulations with the aim of deriving software artifacts that demonstrate compliance to lawyers, business managers and auditors [8]. The methodology provides engineers with a systematic process for extracting two new and important requirements artifacts from regulatory documents: *rights*, which describe what actions stakeholders are permitted to perform, and *obligations*, which describe what actions stakeholders are required to perform. To support this endeavor, we developed several formal primitives and natural language patterns to model the deep semantics expressed in regulatory statements. These primitives allow us to identify ambiguities early-on and reason about consistency in regulatory requirements [6, 8].

In this paper, we discuss the implications that stakeholder rights and obligations have on software systems by demonstrating how our semantic primitives can be used to acquire software artifacts. We precisely define these primitives in the Z notation [11] using an object-oriented semantics to conceptually align the codified rights and obligations with corresponding interfaces, requirements and test cases. Together, the regulation text, codified rights and obligations, and the derived software artifacts provide auditors with a reproducible and certifiable chain of evidence that shows how software systems comply with the law.

To motivate and illustrate this research, we have applied this process to an extensive U.S. Federal regulation developed to comply with the Health Insurance Portability and Accountability Act (HIPAA). The regulation, called the HIPAA Privacy Rule [12], governs the use and disclosure of electronic patient information in the healthcare industry and was

[1] U.S. Public Law No. 104-191, est. 1996.

developed by the U.S. Department of Health and Human Services (HHS). Since the effective compliance date in April 2003 for large healthcare providers, enforcement of the Privacy Rule by HHS has been entirely complaint-driven, with a reported 18,000 complaints registered since March 2006 [20]. These complaints include improper use and disclosure of patient health information (PHI) and disclosing more information than the minimum necessary to complete a transaction. Until auditors have a reasonable mechanism for compliance auditing (as proposed in this paper), enforcement will continue to be complaint-driven. The final HIPAA Enforcement Rule requires increased accountability and imposes civil monetary penalties of up to $25,000 per violation for non-compliance; thus it behooves organizations to adopt methodologies that will facilitate accountability and compliance.

The remainder of this paper is organized as follows: in Section 2, we review related work; in Section 3, we show how to manage traceability when deriving rights and obligations from regulations; in Section 4, we introduce the semantic primitives used to codify rights and obligations as object-oriented class structures; in Section 5, we present techniques to acquire software artifacts from codified rights and obligations; in Section 6, we conclude with the discussion and summary.

2. Related Work

The problem at hand concerns aligning the environment, which includes stakeholders and the actions they perform, with the operations of software systems [21]. This section begins with an evolution of methodologies to codify objects and properties from natural language statements [1, 9, 18, 19, 16]. Our approach is distinguished from these by the use of activity models to organize domain knowledge [8], similar to goal-based requirements engineering methods [10, 2]. After discussing our progress in this area, we briefly compare our methodology [8] to another approach analyzing the HIPAA that grew out of formal methods research [17].

Early work to map natural language phrase structure to software artifacts includes Abbot's classification system for objects and data properties using English nouns, adjectives and verbs [1] and Chen's work with Entity-Relationship diagrams [9]. Recently, Overmyer et al. describe the Linguistic assistant for Domain Analysis (LIDA) that uses part-of-speech tagging to map nouns, adjectives and verbs to classes, attributes and operations in conceptual models [18]. Alternatively, others have developed natural language templates to help designers specify properties in finite state verification [19] and real-time requirements specification [16]. Following this trend, our work focuses on patterns for extracting activities from regulatory texts to specify system requirements.

In requirements engineering, goals are semi-structured statements that describe actions performed by actors within the environment. The KAOS framework provides a formal semantics for goals that includes goal refinement and temporal constraints between goals [10]. Goal refinement entails refining high-level organizational goals into low-level operational goals and requirements; goal refinement helps stakeholders realize their business practices in terms of software requirements [10, 2].

To add rigor to the goal-acquisition process, Antón proposed the Goal-Based Requirements Acquisition Methodology (GBRAM) to extract goals from natural language documents [2]. The GBRAM has since been applied to financial and healthcare privacy policies [3]. Goals extracted from these policies have been used to construct a privacy taxonomy [4] and to develop new semantics for reasoning about goals [5, 6]. These new semantics enable the expression of goals in policies [6] and regulations [7, 8] as rights and obligations. In Deontic Logic, rights and obligations are those actions that distinguish "what is permissible" from "what ought to be" [14]. Because some stakeholder rights impose implies obligations on other stakeholders, engineers can use these semantics to increase requirements coverage by identifying the implied obligations that fulfill those rights [8].

Goal-based requirements engineering and the new semantics from our previous studies [5. 6] inspired our methodology to extract rights, obligations and constraints from regulations [8]. The methodology was developed empirically using Grounded Theory [12] and applied in four studies: a formative study of a HIPAA fact sheet [7] that summarizes the Privacy Rule for patients; in the HIPAA Privacy Rule, a detailed case study of §164.520–§164.526 [8], regarding notification, amendment and restrictions to PHI and a broad study of all fourteen sections of the Privacy Rule to identify information use and disclosure rules. In this paper, we present the results of fifth study on §164.528 and show how our models of regulatory statements are used to acquire software compliance artifacts.

May et al. employ a methodology to derive formal models from regulations that they applied to §164.506 in the HIPAA Privacy Rule [17]. Several of their basic assumptions, including 1) each paragraph has exactly one rule; and 2) external and ambiguous references are satisfied by default [17], are contradicted by our own study of four other sections in the Privacy Rule [8]. In contrast, not only do paragraphs contain multiple rules (e.g., different cases *and* different stakeholder actions), we discovered that cross-references introduce

important constraints from other sections that restrict which rules apply. If unaccounted for, engineers are prone to make interpretations and inferences that are inconsistent with the law. To address these challenges, we show how to manage cross-references (see Section 3) and provide semantic primitives that guide engineers in identifying resolving ambiguities (see Section 4).

3. Traceability across Regulations

Laws and regulations like the HIPAA are written in dense legalese with numerous cross references to other paragraphs, sections and other laws. In the Privacy Rule alone, there are over 400 cross-references. These references refer to restrictions that are described in other paragraphs and require the engineer to interpret those restrictions across multiple contexts [8]. For example, consider the regulation text paraphrased, below. The text is adapted precisely from the HIPAA Privacy Rule; text was only omitted and not changed in any other way. The *italicized* text identifies rights and obligations; the **bold** text is a condition keyword (except) and the underlined text is a purpose or a numerical constraint.

Paraphrased from §164.528:

(a) *The individual has a right to receive an accounting of disclosures made by the CE, except for disclosures:* …

(iii) For notifications purposes as provided in §164.528; …

(b) *The CE must provide the individual with an accounting that*:

(i) **Except** as provided by paragraph (a) of this section, *the accounting must include disclosures* that occurred during the six years prior to the date of the request.

Paraphrased from §164.510:

(c) Permitted uses and disclosures: …

(ii) *The CE may disclose PHI* to notify a family member of the individual's location or general condition.

Applying our methodology and its associated natural language patterns (see Section 4) yields the activities A_1–A_6. Activities stated as nouns and purposes are separated and replaced by a reference to the separate activity index (in parenthesis) as shown below. Each activity is followed by the reference, in **bold,** to the regulation paragraph from which it was extracted; and inferred phrases are italicized. The complexity of these regulations makes maintaining this kind of traceability critical.

A_1: [Right] The individual may receive an accounting of disclosures excluding (A_2). **§164.528(a)**

A_2. The CE discloses PHI to an *entity* for the purpose of (A_3). **§164.528(a)(iii)**; **§164.510(c)(ii)**

A_3: The *entity* notifies a family member of the individual's location or general condition. **§164.510(c)(ii)**

A_4: [Obligation] The CE must provide the individual with an accounting of disclosures, excluding (A_2). **§164.528(b)**, **(b)(i)**, **(a)(iii)**

A_5: The disclosures occurred during the six years prior to the request (A_5). **§164.528(b)(i)**

A_6: The *individual* requests *an accounting of disclosures*. **§164.528(b)(i)**

Activity A_1 is a right and A_4 is an obligation. Both A_1 and A_4 are said to balance each other because each implies the other [8]. Also, note that activity A_3 is the purpose of the disclosure in A_2; the purpose is a frequent semantic primitive we formalize in Section 4. To appreciate the traceability challenge, observe how the disclosure A_2 is first referenced generally in paragraph (a) in the phrase "except for disclosures" and is then incrementally refined in paragraph (a)(iii) by the phrase "for notification purposes" that it is ultimately detailed in paragraph (c)(ii) by "to notify family members of the individual's location or general condition."

By isolating activities and tracing their references through cross-references, engineers can refine the specifications for stakeholder actions. Skipping these references, as others routinely do in their analyses of regulations [17], results in under-specifications that lead to unauthorized behavior or assumptions that contradict or violate the intent of the law.

4. Modeling Regulatory Semantics

Rigorously obtaining compliance artifacts from natural language regulations relies on consistently encoding governed business practices as formal structures. In Section 3, we showed how to informally extract discrete activities and constraints from regulations. To ensure consistency, we express these activities as class structures using the Z notation, pronounced *Zed*, based on Zermelo-Fraenkel set theory and first-order predicate logic [11], and published as the international standard ISO/IEC 13568:2002. These class structures are presented so that other engineers and researchers can reuse our approach. Other notations, such as the Unified Modeling Language[2] (UML) and Alloy [15], are insufficient to model *complete* regulatory statements: UML is ideal for visual presentation, whereas we require a concise formal semantics; and Alloy, developed to perform

[2] The UML 2.0, Object Management Group (OMG), http://www.uml.org

model checking, lacks semantics for linear arithmetic expressions commonly found in numerical constraints. After a brief introduction to Z, we discuss how to specify objects, properties and constraints in Z before illustrating our regulatory class structures referred to in our results in Section 5.

4.1. Brief Introduction to Z

In the Z notation, a quantified expression has the following syntax: ($Q\ v : R \mid c \bullet p$), where Q is the quantifier $\forall$ (for all) or $\exists$ (there exists), v is a bound variable in the range R, c is the constraint and p is the predicate. The expression is read "for all/ there exists v in R satisfying c such that p." We use $\Leftrightarrow$ (if and only if) and $\Rightarrow$ (implies) to make inferences. For example, the constraint c must be satisfied by all elements in R bound to v and is treated as a logical conjunction $\wedge$ or an implication $\Rightarrow$ as follows:

$$(\exists\ v : R \mid c \bullet p) \Leftrightarrow (\exists\ v : R \bullet c \wedge p)$$
$$(\forall\ v : R \mid c \bullet p) \Leftrightarrow (\forall\ v : R \bullet c \Rightarrow p)$$

Given two sets A, B of objects, we can define a binary relation R as a subset of the Cartesian product $A \times B$ that consists of ordered pairs (a, b), also written $a \mapsto b$. Using the expression syntax, we define this relation as follows: $R = \{a{:}A; b{:}B \bullet a \mapsto b\}$. Relations are classified as symmetric, reflexive and transitive, as well. Binary relations with unique mappings from every element in their domain to every element in their range are called total, bijective functions in Z. In Z, the total, bijective function $f = \{ z{:}\ \mathbb{Z} \bullet z \mapsto z \}$ maps every integer z in the set $\mathbb{Z}$ of integers to itself, called an identify function.

4.2. Formal Definitions

We present our primitives using an object-oriented semantics expressed in the Z notation with the intent that engineers can reuse these primitives to instantiate class objects from words in regulatory statements. These objects can then be used to identify the software artifacts presented in Section 5. In this paper, objects have two fundamental relationships: *classes* that organize objects into conceptual hierarchies with types and sub-types, and *properties* that define objects as either sets, called *aggregation*, or as constituent parts of a whole, called *composition*.

Definition 1: A *class* is a finite set of objects and a *sub-class* is a subset of a parent class. The root (most abstract) class is the set *Object* that contains all objects. Two additional classes are presumed: the class *Integer* $= \{ z{:}\ \mathbb{Z} \mid -n \le z \wedge z \ge n \}$ for some constant n is a sub-class *Integer* $\subseteq$ *Object*; and the sub-class *String* $\subseteq$ *Object* contains words consisting of printable characters.

Definition 2: A *property* is an anti-reflexive, anti-symmetric binary relation between the objects of two classes. For example, for two sub-classes *Actor*, *Information* $\subseteq$ *Object*, we can define the property *source* $= \{a{:}Information; b{:}Actor \bullet a \mapsto b\}$ such that the predicate *source*(a, b) is true if and only if the actor b is the source of the information a. If a property is required by a class, then excluding it from an object specification is an ambiguity.

Regulatory statements use numerical constraints to specify deadlines and conditions on numerical properties. For example, the phrase "children younger than 13 years of age" contains a numerical constraint "younger than" on the age of children [7]. Numerical constraints are defined over object properties that have ranges in the class *Integer*. The expression for numerical constraints is: *v Op w*; where *v* is in *Integer*, *Op* is one of (=, >, <, ≥, ≤) and *w* is in *Integer* or an arithmetic expression with operators in (+,–,∗) for addition, subtraction and multiplication .

4.3. Semantic Primitives and Class Structures

The semantics are summarized from five studies of policies and regulations, to analyze:

1. one hundred privacy goals in financial and healthcare policies [5, 6]; and
2. a fact sheet summarizing patient rights [7] under the HIPAA Privacy Rule;
3. information use and disclosure practices in fourteen sections §164.502–§164.532 of the Privacy Rule;
4. policy notification and requests for restrictions, access and amendment of PHI in four sections §164.520–§164.526 [8] of the Rule; and
5. in this paper, an individual's right to receive an accounting of disclosures of PHI in §164.528 of the Rule.

In the following sub-sections, we define the classes and properties to model regulatory statements. These formal relations are summarized in the UML diagram in Figure 1. Each sub-section begins with a sample regulatory statement, following by the general notation for expressing the primitive and concluding with the observed frequency for the primitives in the five studies as empirical evidence, suggestive of their importance.

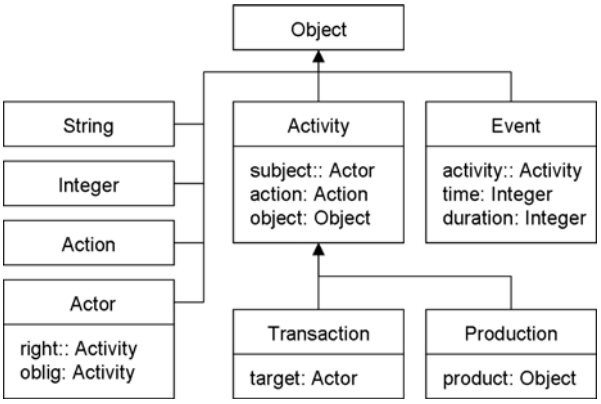

Figure 1: UML Class Diagram for Regulatory Semantics

Based on our experience in analyzing policies and regulations from healthcare and finance, it is reasonable to expect that these semantics are domain-independent and reusable in other domains.

4.3.1. Basic Activities

The basic activity is comprised of a subject (a noun) who performs an action (a verb) on some object. For example, "the covered entity (CE) documents an oral statement" has the subject "CE," action "document" and object "oral statement." More complex activities, such as transactions betweens stakeholders, extend this basic model. The sub-class *Actor* $\subseteq$ *Object* contains all subjects; the sub-class *Action* $\subseteq$ *Object* contains all actions; and the sub-class *Activity* $\subseteq$ *Object* contains all activities. We define the following required properties for all activities:

$$subject = \{v{:}Activity; s{:}Actor \bullet v \mapsto s\}$$

$$action = \{v{:}Activity; a{:}Action \bullet v \mapsto a\}$$

$$object = \{v{:}Activity; o{:}Object \bullet v \mapsto o\}$$

Natural language statements describing a basic activity can be mapped to activity objects following a simple procedure. Consider the activity statement S_1, followed by the class declarations C_1 for the noun and verb phrases and the corresponding Z expression Z_1:

S_1: The covered entity documents the oral statement.
C_1: *CoveredEntity* $\subseteq$ *Actor*; *OralStatement* $\subseteq$ *Object*; *Document* $\subseteq$ *Action*
Z_1: $(\exists v{:}Activity; s{:}CoveredEntity; a{:}Document, o{:}OralStatement \bullet subject\ (v,s) \wedge action\ (v,a) \wedge object\ (v,o)$

The statement S_1 is defined formally by first creating sub-classes for noun phrases *CoveredEntity*, $\subseteq$ *Actor*, *OralStatement* $\subseteq$ *Object* and the verb *Document* $\subseteq$ *Action*, if they do not exist. In the Z expression, the variables *v*, *s*, *a*, *o* bind the instances of these classes to the properties of the activity.

In regulation texts, most verbs indicate an activity and some nouns refer to activities. For example, the noun "disclosure" refers to the activity "to disclose" and "notification" refers to the activity "to notify." The engineer must identify these activity nouns in regulations and attempt to identify the range restrictions for the subject, action and object properties associated with each activity.

Sometimes, these activity nouns will appear before a preposition and another noun, such as "disclosures of PHI" or "request for suspension." These phrases are not indicative of composition, they are partitive phrases that cannot be inverted into possessive phrases (e.g., information's disclosure) and the engineer should exercise caution when considering these phrases as aggregations when the preposition "of" is concerned (e.g., accounting of disclosures). We further discuss this type of aggregation as description in Section 4.3.5.

In our analysis of §164.502–§164.532, we identified 306 separate activities that were information use and disclosure practices. In the more detailed analysis of §164.520–§164.528 in which we analyzed every statement in these sections, we identified 613 separate activities.

4.3.2. Transactions

Activities in which the action is performed between two parties are called *transactions;* for example, a disclosure of information from one party to another. Like general activities, we distinguish the subject, the actor who performs the action, from the target, the other actor with whom the action is performed. The sub-class *Transaction* $\subseteq$ *Activity* contains all transactions and the required property *target* accounts for the other actor with whom the action is performed:

$$target = \{\ v{:}Transaction; t{:}Actor \bullet v \mapsto t\ \}$$

Consider the example transaction S_2, a notification, class definitions C_2 and instance Z_2:

S_2: The covered entity notifies the individual about their privacy practices.
C_2: *CoveredEntity*, *Individual* $\subseteq$ *Actor*; *PrivacyPractice* $\subseteq$ *Object*; *Notify* $\subseteq$ *Action*
Z_2: $(\exists v{:}Activity; s{:}CoveredEntity; a{:}Notify, o{:}PrivacyPractice, t{:}Individual; \bullet subject(v, s) \wedge action(v, a) \wedge object(v, o) \wedge target(v, t))$

In S_2, the noun phrase for the other actor is modeled by the sub-class *Individual* in C_2, which is instantiated by the variable *t* of the *target* property in Z_2.

In English, many transactions have complementary actions in which the activity can be viewed from the perspective of either the subject or the target. For

example, the action "disclose" is complemented by the action "receive," in which the target of the disclosure is the subject in the activity "to receive." Observing these complements is important when evaluating which transactions become operationalized by the software interface (see Section 5).

4.3.3. Events and Temporality

Regulations restrict the scope of events to certain periods of time; these restrictions are called *temporal constraints*. For example, the statement "the disclosures occurred six years prior to the request" restricts the date of relevant disclosures to within six years of the date of the request. For simplicity, we reduce temporal constraints to linear arithmetic constraints over the smallest unit of time, in this case seconds, using total, bijective functions such as:

$$second = \{ z: \mathbb{Z} \bullet z \rightarrowtail z \}$$

$$minute = \{ z: \mathbb{Z} \bullet z \rightarrowtail 60 * second\ (z) \}$$

$$hour = \{ z: \mathbb{Z} \bullet z \rightarrowtail 60 * minute\ (z) \}$$

$$day = \{ z: \mathbb{Z} \bullet z \rightarrowtail 24 * hour\ (z) \}$$

The performance of an action is called an *event*. All *event* objects are in the sub-class $Event \subseteq Object$ with the following required properties:

$$activity = \{e{:}Event, a{:}Activity \bullet e \mapsto a \}$$

$$time = \{e{:}Event, t{:}\ \mathbb{Z} \bullet e \mapsto t \}$$

$$duration = \{e{:}Event, d{:}\ \mathbb{Z} \bullet e \mapsto d \}$$

The *activity* property specifies which activity the event pertains to; the *time* property specifies the start time of an event; and the *duration* property specifies the length of time during which the event has occurred. The stop time for an event is equal to the *time* plus *duration*.

Consider the statement S_3, followed by the class declarations C_3 and temporal constraints in Z_3:

S_3: The provision is before 60 days after the request**.**

C_3: $Provision, Request \subseteq Activity$

Z_3: $(\exists e_1,e_2{:}Event; v_1{:}Provision; v_2{:}Request; t_1,t_2{:}Integer \mid activity\ (e_1, v_1) \wedge activity\ (e_2, v_2) \wedge time\ (e_1, t_1) \wedge time\ (e_2, t_2) \bullet t_2 < t_1 \wedge t_1 < t_2 + day(60))$

The events e_1, e_2 for the activities v_1 (a provision) and v_2 (a request), respective, and the temporal constraint states that the time of e_1 is after e_2 but before 60 days after e_2. By formalizing these constraints, we can derive function requirements that ensure these events occur within the intended deadline.

In the Privacy Rule, we identified nine phrases with 92 occurrences that are numerical constraints and 41 occurrences that are arithmetic operations [7]. These results include temporal constraints expressed between activities using prepositions (e.g., before, after, during) but exclude temporal constraints acquired from verb tenses.

4.3.4. Rules, Rights and Obligations

Rules are statements intended to control behavior within the environment. Deontological rules are statements that include obligations, which express "what ought to be," and rights, which express "what is permissible" [14]. On the other hand, causal rules are implications with condition keywords (if, unless, except if) that place pre- or post-conditions on events. We now discuss the two types of rules in more detail.

Rights and obligations are identified using common phrases [7, 8]. For example, "must" and "is required to," are phrases that indicate obligations and "may," "is permitted to," or "is a right of" indicate rights. The engineer must assign a single modality to occurrences of these phrases in the regulation text, observing that some phrases are ambiguous (e.g., "may" can mean "is permitted to" or "is capable of.") To assign activities to actors as their rights or obligations, we use the following two optional properties:

$$right = \{a{:}Actor; v{:}Activity \bullet a \mapsto v\}$$

$$obligation = \{a{:}Actor; v{:}Activity \bullet a \mapsto v\}$$

In §164.502–§164.532, we codified 256 rights to "use or disclose PHI" and 50 obligations "to not use or disclose PHI." In §164.520–§164.528, among the individual activities we identified 51 rights and 88 obligations. In prior work, we show how the rights of one stakeholder can be used to uncover the implied obligations of another stakeholder using formal models [8]. In Section 5, we further show how to operationalize stakeholder rights and obligations into discretionary and mandatory requirements.

Causal rules have an antecedent that must be satisfied before the consequent is satisfied. For an antecedent a and consequent c, we express a causal rule using the implication $a \Rightarrow c$. In regulatory statements, the events in causal rules use different tense verbs. Consider the following causal rule from §164.528, paragraph (a)(2)(i), describing the covered entity (CE) and health oversight agency (HOA):

S_4: The CE must suspend an individual's right to receive an accounting of disclosures to a HOA, if the HOA provides the CE with a written statement that such an accounting would be reasonably likely to impede the agency's activities.

$S_{4.1}$: The CE suspends an individual's right to receive an accounting of disclosures to a HOA.

$S_{4.2}$: The HOA provides the CE with a written statement.

The activities from statement S_4 are separated into $S_{4.1}$ and $S_{4.2}$. Using the patterns for basic activities and transactions, we derive $v_{4.1}$ and $v_{4.2}$ from $S_{4.1}$ and $S_{4.2}$, respectively, and the pattern for events to derive $e_{4.1}$ and $e_{4.2}$. We express the rule in S_4 as a combined causal/ deontological rule, as follows:

$\mathbf{Z_{4.5}}$: $\exists a$:*Actor* • ($e_{4.2} \Rightarrow$ *target* ($v_{4.2}$, a) $\wedge$
obligation (a, $v_{4.1}$))

The statement S_4 states that, in the event of $e_{4.2}$, the actor $a \in CE$ is obligated to perform the activity $v_{4.2}$. We can thus infer, if the actor fulfils their obligation $v_{4.1}$, that the event $e_{4.2}$ would occur sometime in the future. In regulations, the distinction between causal and deontological rules is intended to help engineers distinguish between the existence of rights and obligations from the conditions governing their assignment to stakeholders.

4.3.5. Descriptions as Aggregations`

Descriptions are trivial activities, e.g., "a description describes something." To simplify matters, descriptions are treated as aggregations of the things that they describe. In the Privacy Rule, there are several objects that are modeled as descriptions, including: written and oral statements, authorizations, requests, testimonials and accountings, each of which contains important information to complete a related transaction. The contents of these descriptions include deadlines for performing actions, records of activities such as disclosures to third parties and beliefs about past and future events. When these contents appear in a stakeholder transaction that is operationalized, the descriptions must be formalized as an aggregation of data objects. We account for these aggregations using the sub-class *Description* $\subseteq$ *Object*; recognizing that sub-classes in *Description* have properties motivated directly by statements in the regulation text.

For example, consider the description for an "accounting of disclosures" from §164.528, paragraph (b)(2), and paraphrased as follows:

S_5: The accounting must include for each disclosure:
- (i) The date of the disclosure;
- (ii) The name of the entity who received the PHI and, if known, the entity's address.
- (iii) A brief description of the PHI.
- (iv) A brief statement of the purpose of the disclosure.

Notice that the accounting of disclosures does not contain the disclosure objects, per se, but records that contain properties of disclosures including the *date*, *object*, *target*, and *purpose*. We define the sub-classes *Record*, *Accounting* $\subseteq$ *Description* and the property:

record = {a:*Accounting*, r:*Record* • $a \mapsto r$}

We first define any new properties on the objects used to derive the description, such as the name and address of the entity. For example, based on the statement (ii) in S_5, we define the *name* property:

name = {a:*Actor*, s:*String* • $a \mapsto s$}

The properties of a record are defined as a subset of properties of other objects; thus, the name in a record is the name of the actor who receives PHI in a disclosure. The *name′* property in a record achieves this degree of abstraction, as follows:

name′ = {r:*Record*, s:*String* |
($\exists a$:*Actor*; n:*String* | *name*(a, n) • n) • $r \mapsto n$}

Descriptions, if operationalized, are class structures that map directly to data schema and that entail corresponding data maintenance requirements, as discussed in Section 5.

4.3.6. The Act of Production

Productions are activities that yield other objects or properties. For example, in regulatory statements the activities "to account" and "to document" yield the objects *Accounting* and *Documentation*, respectively. In some cases, the same noun refers to the production (the act) as well as the product. For example, the noun "accounting" means two different things — the act "to account" and the product of the act, an "accounting." We define the sub-class *Production* $\subseteq$ *Activity* that consists of these production objects with the required property:

product = {p:*Production*, o:*Object* • $p \mapsto o$}

For example, consider the statement S_5 from §164.528, paragraph (d)(2), paraphrased below:

S_5: The CE must document the written accounting provided to the individual.

In statement S_5, the activity "to document" yields an object that describes the written accounting, in this case, the accounting specified in statement S_5. The documentation of an accounting is only a description of the accounting and not the actual accounting object. Regulations that are complete and consistent will describe these products so that, if these activities are stakeholder obligations, they can be realized as data objects within systems.

5. Deriving Software Artifacts

The semantic primitives in Section 4 were applied to §164.528 in the Privacy Rule to codify 65 classes and 25 properties. The classes and properties were then

used to specify software artifacts, including interfaces between business and software processes, discretionary vs. mandatory requirements, data schemas and data requirements and event-based test cases. We now present the techniques to acquire these artifacts.

5.1. Identifying the Software Interfaces

The first step to derive software artifacts from regulated activities is to select the "in-scope" stakeholder activities to be supported by the system. In §164.528, we identified 9 actors whose role in an activity could be defined by a software interface. For example, Figure 2 shows the in-scope activities in §164.528 where the covered entity (CE) plays a role; here, the third parties are external to the system scope but transactions that involve the CE are all in-scope. Arrows indicate transactions directed from the subject to the target of the transaction (e.g., the CE discloses PHI to the Business Associate). The lines without arrows are actions performed by the CE that are not transactions with other stakeholders.

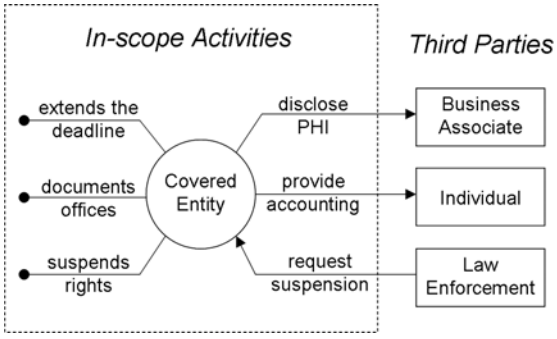

Figure 2: In-Scope Stakeholder Activities

The transactions that cross the in-scope boundary to third parties define the software interfaces, in which the objects of the transaction are encoded by a user interface and later processed by the system. For example, in Figure 2, the suspension requested by law enforcement will become an object with an electronic "description" in terms of classes and class properties sufficient to complete the transaction. The time of the suspension and disclosures to which the suspension pertains, encoded as properties of the suspension object, are passed through the interface defined by this transaction. In §164.528, we extracted 30 transactions, each of which could be made an "in-scope" activity.

The other "in-scope" activities that are not transactions are also refined into requirements, although their interfaces will be internal to the system. We next discuss this refinement process in terms of discretionary vs. mandatory requirements.

5.2. Discretionary vs. Mandatory Requirements

In general, stakeholder rights within the system scope are refined into discretionary requirements whereas "in-scope" obligations are refined into mandatory software requirements. Exceptions to the general rule are discussed in this section. Consider the following regulation statement from §164.528, paragraph (c)(1), in which the right follows from the obligation:

A_7: [Obligation] The CE must provide the individual with the requested accounting within 60 days of the date of the request. **§164.528(c)(1)**, **(c)(1)(i)**

A_8: [Right] The CE may extend the time to provide the accounting by no more than 30 days. **§164.528(c)(1)(ii)**

Because the provision of accountings to individuals is within the system scope, the obligation A_7 is refined as the following mandatory requirement, in which the stakeholder is replaced by the system or component designated to implement this requirement:

Req_1: *The system shall provide the individual with the requested accounting within 60 days of the date of the request.*

The extended deadline described by right A_8, however, reflects a discretionary requirement; meaning, the CE can develop a system that implements the mandatory requirement and never worry about the extended deadline — assuming the verified system never fails.

However, the CE may find other factors beyond the system's control that would cause the system to violate the mandatory requirement; such as including the disclosures made by business associates beyond the system scope. These factors make absorbing the costs to implement discretionary requirements, necessary. By identifying regulated activities as rights or obligations, the stakeholders and system designers can weigh the value of discretionary requirements based on their unique circumstances to avoid introducing unnecessary costs and complexity into the system.

For obligations that follow from (or are implied by) a stakeholder exercising their right, the obligation must be refined into a mandatory requirement only if that right is also refined into a system requirement. When implementing discretionary requirements from rights, the engineer must enumerate any causal rules that were extracted from the regulations (see Section 4.3.4), in which a right implies an obligation, to identify new mandatory requirements.

In §164.528, we identified five discretionary requirements, 10 mandatory requirements and five causal rules that affect rights or obligations. There are two cases in which obligations are pre-conditions to

other rights; that is, the circumstances that obligate the stakeholder are necessary to permit the stakeholder to exercise the right. In one case, the right allows the stakeholder to mitigate non-compliance by extending a deadline; in the other case, the right allows the stakeholder to optimize redundancy in a required communication with a third party.

5.3. Data Maintenance Requirements

For each in-scope activity, the object of the activity is a candidate data element. The benefit of using the patterns from Section 4 is that these objects are already described in object-oriented structures, which can be directly mapped to a modeling language like the eXtensible Markup Language[3] (XML) or the UML. For example, consider the following activities:

A_9: The covered entity must document the titles of the offices that (A_9). **§164.528(d)(3)**

A_{10}: The offices receive requests for an accounting from individuals. **§164.528(d)(3)**

In statement A_9, the activity "to document" has the object "titles of offices" which is specified as the *title* property over the classes *Office*, *String* $\subseteq$ *Object*, as follows:

$$title = \{o{:}Office; s{:}String \bullet o \mapsto s\}$$

Because this is an in-scope obligation, the engineer derives the following data maintenance requirement:

Req_2: *The system shall maintain the titles of the offices that receive requests for accountings from individuals.*

Typically, activities within the system scope that are descriptions (see Section 4.3.5) or productions (see Section 4.3.6) incur data requirements because the objects described or produced must be stored in the system for later use. Furthermore, the properties of objects that appear in descriptions and numerical constraints also yield data requirements. In §164.528, we identified eight data requirements for objects that are descriptions or products in a production and 25 properties of objects that appear in in-scope activities

5.4. Functional Requirements and Test Cases

Test cases derived from functional requirements can be used to ensure that systems comply with policies and regulations. Events are ideal candidates for developing test cases to evaluate runtime systems, provided these events have codified temporal constraints (see Section 4.3.4). For example, recall activities A_6 and A_7, below:

A_6: The *individual* requests *an accounting of disclosures*. **§164.528(b)(i)**

A_7: The CE must provide the individual with the requested accounting within 60 days of the date of the request. **§164.528(c)(1)**, **(c)(1)(i)**

Z_7: $\exists u,v{:}Integer \bullet (time\ (e_7, u) \wedge time\ (e_6, v) \wedge u < v + day(60))$

In Z_7, we state that the time of the event e_7, which corresponds to the occurrence of activity A_7, must be less than 60 days after the time of the event e_6, which corresponds to the occurrence of A_8. By adding 60 days to the time of event e_6, the temporal constraint creates a functional requirement in which the system can compare the in-scope event times for occurrences of e_6 and e_7. Corresponding test cases would evaluate these constraints based on a simulated or runtime environment, allowing stakeholders to take corrective action if systems deviate from expected behavior.

We identified 11 such test cases from temporal constraints in the study of §164.528. Among these constraints, there were six arithmetic constraints in which a constant quantity of time, such as 60 days, was added to the time of another event.

6. Discussion and Summary

Increasingly, organizations must be able to demonstrate that they have verifiable procedures in-place to implement the restrictions imposed on information collection and use by government regulations and policies. As such, software engineers need support for identifying and classifying ambiguities in policy and regulatory documents, so they may distinguish between necessary and discretionary software requirements that will lead to demonstrably compliant software systems. Using examples from our in-depth analysis of the HIPAA, we have shown how to create the following software artifacts to achieve this objective: specifications for transactions including interfaces between business and software processes; data schemas and data maintenance requirements; and functional requirements with associated test cases for ensuring that systems comply with policies and regulations.

Currently, we are continuing to validate this approach via our analysis of §164.520-§164.526 to further validate the scalability of the techniques and to identify alternative or additional ways to acquire artifacts from formal models of activities. The growing body of structured domain knowledge we have developed from our HIPAA analyses, including the generalized natural language patterns and techniques to perform formal consistency checking [8], are being integrated into tool support to assists others in applying this approach to other regulations and standards.

[3] The XML, W3C, http://www.w3.org/XML

Finally, we are seeking opportunities to work with auditors and compliance officers to assess the challenges they face in aligning the formal abstractions from our methodology with legacy information systems to support their compliance goals.

Acknowlegdements

This work was supported in part by NSF ITR #032-5269, NSF CT #043-0166 and Purdue, CERIAS.

References

[1] R.J. Abbot, "Program design by informal English descriptions." *Comm. ACM*, 26(11):882-894, 1983.

[2] A.I. Antón, "Goal-based requirements analysis," *2nd IEEE Int'l Conf. Requirements Engineering*, pp. 136-144, 1996.

[3] A.I. Antón, J.B. Earp, Q. He, W. Stufflebeam, D. Bolchini, C. Jensen, "Financial privacy policies and the need for standardization," *IEEE Sec. and Privacy*, 2(2):36-45, 2004.

[4] A.I. Antón, J.B. Earp, "A requirements taxonomy for reducing web site privacy vulnerabilities," *Requirement Engineering*, 9(3):169-185, 2004.

[5] T.D. Breaux, A.I. Antón, "Deriving semantic models from privacy policies," *6th IEEE Int'l Workshop on Policies for Dist. Sys. and Net.*, pp. 67-76, 2005.

[6] T.D. Breaux, A.I. Antón, "Analyzing goal semantics for rights, permissions and obligations," *13th IEEE Int'l Conf. Reqs. Engr.*, pp. 177-186, 2005.

[7] T.D. Breaux, A.I. Antón, "Mining rule semantics to understand legislative compliance," *ACM Workshop on Privacy Elec. Soc.*, pp. 51-54, 2005.

[8] T.D. Breaux, M.W. Vail, A.I. Antón, "Towards compliance: extracting rights and obligations to align requirements with regulations," *14th IEEE Int'l Conf. on Reqs. Engr.*, 2006.

[9] P. P-S. Chen, "English sentence structure and entity-relationship diagrams" *Information Sciences*, 29(2-3), pp. 127–149, 1983.

[10] A. Dardenne, A. van Lamsweerde, S. Fickas, "Goal-directed requirements acquisition," *Science of Computer Programming*, 20(1-2):3-50, 1993.

[11] J. Davies, D. Woodcock, *Using Z: Specification, Refinement and Proof*, Prentice Hall, Upper Saddle River, New Jersey, 1996.

[12] B.C. Glaser, A.L. Strauss, *The Discovery of Grounded Theory*, Aldine Publishing Co., 1967.

[13] "Standards for Privacy and Individually Identifiable Health Information," U.S. Department of Health and Human Services, *Federal Register*, 67(157): 53182–53273, Aug. 14, 2002.

[14] J.F. Horty, *Agency and Deontic Logic*, Oxford University Press, New York NY, 2001.

[15] D. Jackson, "Alloy: a lightweight object modeling notation," *ACM Trans. Soft. Engr. Meth.* 11(2): 256-290, 2002.

[16] S. Konrad, B.H.C. Cheng, "Real-time requirements specification patterns." *27th IEEE Int'l Conf. Soft. Engr.* pp. 372-381, 2005.

[17] 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.

[18] S.P. Overmyer, B. Lavoie, O. Rambow, "Conceptual modeling through linguistic analysis using LIDA." *23rd IEEE Int'l Conf. Soft. Engr*., pp .401-410, 2001.

[19] R.L. Smith, G.S. Avrunin, L.A. Clarke, L.J. Osterweil. "PROPEL: an approach supporting property elucidation," *24th IEEE Int'l Conf. Soft. Engr*., pp. 11-21, 2002.

[20] W. Wilkinson, "The office for civil rights and health care privacy," *12th National HIPAA Summit*, Washington, D.C., April 10, 2006.

[21] P. Zave, M. Jackson, "The four dark corners of requirements engineering", ACM Trans. Soft. Engr. Methods., 6(1):1-30, 1997.