Modeling Regulatory Ambiguities for Requirements Analysis

Aaron K. Massey[1(✉)], Eric Holtgrefe[1], and Sepideh Ghanavati[2]

[1] Department of Information Systems, University of Maryland, Baltimore County, Baltimore, USA
{akmassey,eholtgr1}@umbc.edu
[2] Department of Computer Science, Texas Tech University, Lubbock, USA
sepideh.ghanavati@ttu.edu

Abstract. Lawyers and policy makers regularly and intentionally use ambiguous language in laws, regulations, and other legal texts. Although ambiguity has important policy benefits, such as interpretive resilience in an ever-changing world, it frustrates engineers and businesses seeking to build software systems that are demonstratively compliant with legal obligations. In this vision paper, we propose a method for modeling legal texts alongside models of software requirements or design artifacts. Our approach allows engineers to reason about regulatory ambiguity separately from their system under development and then trace interpretive decisions made about the legal text to affected requirements models. When a regulation is updated or case law demands a new interpretation of a regulation, engineers can evaluate the effect of the changes on the current design and respond appropriately. Inspired by User Requirements Notation, our proposed method can be implemented as an extension to Legal-GRL.

Keywords: Requirements engineering · Ambiguity modeling · Regulatory compliance

1 Introduction

Regulatory Compliance Software Engineering (RCSE) is an emerging field of interdisciplinary research focused on the development of systematic approaches to building, maintaining, and verifying software systems that must comply with laws and regulations. Laws, regulations, and policy documents, and other legal texts are simultaneously useful and challenging as a source of requirements for software engineers [16]. One of the reasons for this challenge is the use of intentional ambiguity as a means of interpretive resilience in rapidly changing technical environments. For example, specifying a particular encryption algorithm is less resilient than using an ambiguous phrase like "reasonable encryption" because if a specified algorithm were broken, then the law would need to be updated. Unfortunately, any ambiguity in a legal text, whether intentional or

H.C. Mayr et al. (Eds.): ER 2017, LNCS 10650, pp. 231–238, 2017.
https://doi.org/10.1007/978-3-319-69904-2_19

not, must be identified, classified, and disambiguated during requirements engineering [13]. That is, at some point "reasonable encryption" will have to be interpreted to identify a particular algorithm prior to implementation.

Current approaches to RCSE focus on performing an interpretation of legal texts, including the resolution of ambiguity, and linking each interpretation back to the subsection of the policy document from which it came. On the surface, this seems to be all that is needed, but interpreting legal texts is more nuanced than this procedure supports. For example, lawyers cannot give a "definitive" interpretation of a law; they can only give an opinion based on how they believe a court or regulator would interpret the text for a particular situation. As a result, many requirements engineering approaches to regulatory requirements fundamentally require legal domain expertise that is not currently supported by goal modeling, requirements modeling, or other standard software engineering modeling activities. New approaches must be developed to support increased participation of non-legal domain experts, provide flexibility in the face of a changing regulatory environment, and incorporate modeling of regulatory requirements in the software development lifecycle.

In this paper, we propose a method for modeling legal texts alongside models of software requirements or design artifacts. Our approach allows engineers to reason about regulatory ambiguity separately from their system under development and then trace interpretive decisions made about the legal text to the affected requirements models. The goal of this approach is to support reanalysis of ambiguities in the event of regulatory change or updated engineering requirements. By identifying, categorizing, and modeling ambiguities, we can document how those ambiguities are resolved by requirements models or other design artifacts. When a regulation is updated or case law demands a new interpretation of a regulation, engineers can evaluate the effect of the changes on the current design and respond appropriately.

The remainder of this paper is structured as follows. Section 2 details related work in regulatory requirements and goal modeling. Section 3 describes our proposed methodology for constructing an ambiguity model and present an example of its use. Section 4 discusses the implications of our method and details possible future work.

2 Related Work

Policy makers write abstract, intentionally ambiguous language to ensure the laws and policies they construct outlast the current generation of technologies. On the other hand, engineers developing software systems must interpret these laws and regulations to address their specific cases and to ensure compliance. Recent research [5,13] demonstrates that ambiguity and vagueness in privacy policies increase privacy risks and decrease the user trust and willingness to share the personal data.

Analyzing and resolving ambiguities has been a research topic in requirements engineering and analysis for decades. Most engineering approaches to analyzing

and resolving ambiguities involve developing tools [8] and techniques based on natural language processing [14,15,18] or machine learning approaches [17,19]. The goal of these approaches is to resolve—once and for all—ambiguities in requirements with a single, definitive interpretation (e.g., identify the correct antecedent to an ambiguous pronoun). Herein, we avoid definitive resolution in favor of modeling options and supporting reuse and re-examination of interpretive decisions.

In our prior work [13], we developed a taxonomy and a classification methodology for legal ambiguities, consisting of seven types of ambiguity: Lexical, Syntactic, Semantic, Vagueness, Incompleteness, Referential, and Other. Understanding an ambiguity's classification supports disambiguation of that ambiguity. This taxonomy was designed to be broadly applicable, but it is not guaranteed to be comprehensive. Our ambiguity taxonomy describes the process of classifying ambiguities according to their types [13]. Our methodology presented herein can be adapted to other methods for classifying ambiguities.

We use User Requirements Notation (URN) [9] for modeling ambiguities derived from our taxonomy. URN combines Goal-oriented Requirements Language (GRL) [2–4] with Use Case Maps (UCM) [6] in one single notation and provides traceability between the two. GRL includes 'lightweight' mechanisms to help extending the language with the help of metadata, rules, concerns, and links. The Use Case Map notation is used to model scenarios and use cases in terms of a set of responsibilities assigned to *components* (▭) , which represent actors, agents, roles, software modules, systems or sub-systems. Paths start with *start points* (●) and traverse through elements along the way until they reach the *end points* (▮) . Paths contain *responsibilities* (✕) which indicate where actions, activities, or transformations are needed. They can be performed in sequence, concurrently (⊣⊢) , or as alternatives (⤙⤚) . UCM also includes static or dynamic stubs (◇) to model parts of a scenario or a process as a plug-in map. URN has an open source tool-support, called jUCMNav [1], which is a plugin for the Eclipse development environment.[1]

Legal-URN [7] is an extension to the URN framework that helps requirements engineers analyze the compliance of business and software requirements with privacy-related regulations. jUCMNav has been extended to capture concepts from Legal-URN. In our approach, we first model ambiguities with Use Case Maps and then provide links from the ambiguity models to Legal-URN models to perform this analysis. The ultimate goal of our approach is to develop a new form of contribution link that connects regulatory ambiguities with traditional modeling elements. Although we do not develop the syntax herein, we discuss how it would support our ambiguity models in Sect. 4.

3 Constructing Ambiguity Models

In this section, we discuss how to develop ambiguity models. We adopt and extend UCM to model ambiguities in legal statements. First, we identify and

[1] http://eclipse.org/.

classify ambiguities in the legal text we wish to model. Herein, we employ the approach introduced in our prior work [13], but we believe similar approaches to ambiguity identification may also suffice. Regardless of the technique chosen, we recommend examining the complete text prior to modeling because this prevents modeling of ambiguities that are resolved or clarified elsewhere in the regulation.

After classification, we follow the process outlined in Algorithm 1. In general, we use static or dynamic stubs (◇) to model legal text. When the legal text includes an ambiguity, we tag the stub with ambiguity marker as («amb»). Specifically, the steps of our approach are as follows. First, model each subsection of a legal statement with a stub. These represent legal statements to be detailed in a plug-in map. Next, model the plug-in map of each related stub. If modeling an ambiguity, use a stub with an ambiguity marker. If modeling another sub-path, use a regular stub. If modeling a non-ambiguous task, use a responsibility (✕) or other appropriate UCM element. Finally, we model the ambiguities tagged with ambiguity markers using new plug-in maps. The plug-in map includes a path, AND- or OR- Fork(s), and Join elements depending on the semantics of the legal text and ambiguity elements (⟨0⟩).

Algorithm 1: Algorithm for Modeling Ambiguities

```
1 FnRecursive(an element of a legal text) begin
2     if the element under examination is atomic and not ambiguous then
3         model any requirements using traditional techniques;
4     else if the element under examination is atomic and ambiguous then
5         create a stub on the path;
6         document the ambiguity inside the stub;
7     else
8         create a stub on the path;
9         recursively apply this algorithm to each sub-element;
```

We illustrate how our approach for modeling ambiguities works using a section from the Health Insurance Portability and Accountability Act (HIPAA)[2]. We selected §164.312, which contains the technical safeguards regulations for HIPAA, because this article has been used extensively in our prior work [10–12] examining those systems. To start the modeling process, we construct an outline of §164.312 with each subsection modeled using a stub because none of them are ambiguous. After completing this step, we follow the recursive step of the algorithm to model the first stub in §164.312 which is (a) Access Control.

(a) Access Control contains two subparts, labeled as (1) Standard and (2) Implementation Specifications. Subpart (a)(1) contains an ambiguity,[3] thus

[2] Pub. L. No. 104–191, 110 Stat. 1936 (1996).

[3] All ambiguity identification is relative to the interpreter. There is no "ground truth" in ambiguity identification. However, for the sake of simplicity, we refer to Subpart (a)(1) as "containing" an ambiguity. In reality, without an interpreter, these same words are neither ambiguous nor unambiguous.

we model it as a stub with ambiguity marker, («amb»). Subpart `(a)(2)` is modeled as a stub and it includes four separate statements as: `(i)` Unique User Identification; `(ii)` Emergency Access Procedure; `(iii)` Automatic Logoff, and `(iv)` Encryption and Decryption. We further expand the stub for `(a)(2)` in another plug-in map. The first three statements of this subpart are ambiguous.[4] Thus, these three are modeled with stubs with ambiguity markers. The fourth statements does not include any ambiguity so it is modeled with a UCM responsibility element.

During this portion of the modeling, we are only interested in three things: (1) reflecting the structure of the actual legal text we are modeling, (2) accurately identifying ambiguity stubs, and (3) accurately modeling unambiguous statements with traditional methods. We are neither resolving nor prioritizing ambiguities because we want resolution, prioritization, and other analyses to be independent of identification. If identification and classification are not separated from other analysis, then the model is not easily reused.

Next, we expand stubs with ambiguity markers into detailed paths with ambiguity elements. As mentioned above, statement `(a)(2)(ii)` was found to be ambiguous. This statement reads as follows:

> `(a)(2)(ii)` Emergency access procedure: Establish (and implement as needed) procedures for obtaining necessary electronic protected health information during an emergency.

Our analysis found two ambiguities, one syntactic and one vagueness. The phrase *and implement as needed* allows the whole statement to have multiple valid meanings. (e.g., A procedure may be 'established' or 'established and implemented'.) In addition, the phrase *during an emergency* is vague in that no definition for an emergency is provided.

At this point, requirements engineers may begin the task of resolving these ambiguities. Resolution may take many forms, but whatever form it takes for a given project, the data necessary to perform an ambiguity resolution should be recorded here as attributes of the ambiguity stubs. In our prior work, we examined intentionality [13]. That is, did the author of the legal text intend for this ambiguity to be written as ambiguous in the way that we identified it. For `(a)(2)(ii)`, we believe the syntactic ambiguity is not intentional (When would you establish and not implement a procedure?) and the vagueness is intentional (Emergencies are difficult to define with clarity). Regardless the resolution approach taken, all data necessary for resolution should be recorded as attributes of the ambiguity stubs. The goal of this step is to support reuse and facilitate changing legal or engineering requirements.

We model these two ambiguities with ambiguity elements (⟨!⟩) which is added to UCM models as an extension. To complete the ambiguity model, we must lay each ambiguity out on the path. Our approach has two options, the ambiguities are independent or one must be resolved before the other. For `(a)(2)(ii)`, we

[4] Again, based on our interpretation.

decided that both of ambiguities determine the actor and their responsibilities. As a result, we model them in parallel and with AND-fork paths.

Figure 1 illustrates modeling the second of these, including the type of ambiguity, which details our analysis of §164.312(b) which reads as follows:

> Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information.

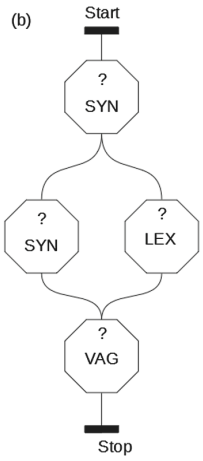

Fig. 1. Ambiguity identified in §164.312(b)

This legal statement includes four ambiguities: two syntactic ambiguities, a lexical ambiguity and a vagueness, summarized in Table 1. We believe all four ambiguities are relatively easy to disambiguate and this text can be implemented by software engineers. They do, however, have a strict ordering, as shown in Fig. 1.

Due to space constraints, we now focus our analysis on the lexical ambiguity resulting from the phrase "...that contain or use electronic protected health information." The "contain and use" part of this phrase is confusing. Does "contain" refer to "having access to some data" or "keeping some data apart from"? No separation of data is explicitly mentioned, so it would be easy to assume the former meaning is correct. However, in this case the word "contain" is superfluous. Any "use" of the data would require access to it. So perhaps the latter meaning of "contain" is correct? This ambiguity is

Table 1. Ambiguities found in 164.312(b)

Type	Phrase	Rationale
Syntactic	"Implement hardware, software, and/or procedural mechanisms that..."	Does the "that" clause apply to the hardware, the software, the procedural mechanisms or some combination of them?
Syntactic	"...that record and examine activity..."	Do the mechanisms need to be implemented for a system that only records activity?
Lexical	"...that contain or use electronic protected health information."	Does contain mean "have access to" or "keep separate from the rest of the system"?
Vagueness	"Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information."	The statement is quite broad. What is actually needed?

also localized and does not affect the meaning of the entire statement. Because both this lexical ambiguity and the second syntactical ambiguity (see Table 1) can be resolved independently and without affecting the resolution of the first syntactical ambiguity, we modeled them in parallel.

4 Discussion and Summary

Although ambiguity has important policy benefits, such as interpretive resilience in an ever-changing world, it frustrates engineers seeking to interpret their meaning and demonstrate due diligence in complying with legal obligations. In this vision paper, we proposed a method for modeling ambiguities in legal texts alongside models of software requirements or design artifacts. We presented an example model that demonstrate how our approach supports engineers as they reason about regulatory ambiguity and come to a disambiguation or resolution strategy. When a regulation is updated or case law demands a new interpretation of a regulation, engineers can evaluate the effect of the changes on the current design and respond appropriately.

Because regulations can change over time, interpretations may also change. A structural model of regulatory ambiguity supports easier change impact analysis because updates to legal texts are denoted with structural changes. For example, consider `§170.314` and `§170.315`. These two sections of HIPAA represent the meaningful use certification criteria for EHR systems for 2014 and 2015, respectively. They are remarkably similar in structure, so if an EHR vendor seeking to transition from 2014 compliance to 2015 compliance used an ambiguity model, many of the implications of the changes could easily be identified.

By modeling legal documents as they are structured to support reuse and reanalysis, we support discussion between the analysts and legal experts as they seek to resolve ambiguities in system design. If ambiguity identification and classification were interleaved with ambiguity resolution, then any change in the regulations may require analysts to either re-identify and re-classify the ambiguities or to undo the resolution process, whatever it may have been (disambiguation, prioritization, etc. . .).

By choosing to model the text in a way that supports multiple interpretations and without a definitive interpretation or resolution in mind, we can meaningfully support analysts seeking to incorporate ambiguity resolution with traditional modeling approaches. Many ambiguity types can only be resolved with domain experts, and for these analysts cannot reach a valid conclusion without a consultation. Unfortunately, by resolving these directly and documenting only the resolution, requirements engineers risk non-compliance resulting from future changes, including changes to regulations, changes to customer requirements, or even simple staff turnover. In essence, the resolved ambiguity has become tacitly hidden, with no indication that the ambiguity existed at all.

References

1. Amyot, D.: JUCMNav. http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/WebHome, October (2016)
2. Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., Yu, E.: Evaluating goal models within the goal-oriented requirement language. Int. J. Intell. Syst. **25**(8), 841–877 (2010)
3. Amyot, D., Horkoff, J., Gross, D., Mussbacher, G.: A lightweight GRL profile for i* modeling. In: Heuser, C.A., Pernul, G. (eds.) ER 2009. LNCS, vol. 5833, pp. 254–264. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04947-7_31
4. Amyot, D., et al.: Towards advanced goal model analysis with jUCMNav. In: Castano, S., Vassiliadis, P., Lakshmanan, L.V., Lee, M.L. (eds.) ER 2012. LNCS, vol. 7518, pp. 201–210. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33999-8_25
5. Bhatia, J., Breaux, T.D., Reidenberg, J.R., Norton, T.B.: A theory of vagueness and privacy risk perception. In: 24th International RE Conference, Beijing, China, September 2016
6. Buhr, R., Casselman, R.: Use Case Maps for Object-Oriented Systems. Prentice-Hall, Upper Saddle River (1995)
7. Ghanavati, S.: Legal-URN Framework for Legal Compliance of Business Processes. PhD thesis, University of Ottawa, Ottawa, Canada (2013)
8. Gordon, D.G., Breaux, T.D.: Reconciling multi-jurisdictional legal requirements: a case study in requirements water marking. In: 20th IEEE International RE Conference, pp. 91–100, September 2012
9. ITU-T. User Requirements Notation (URN) – Language definition. Technical Report ITU-T Z.151, ITU-T, October 2012
10. Massey, A.K., Otto, P.N., Antón, A.I.: Evaluating legal implementation readiness decision-making. IEEE Trans. Softw. Eng. **41**(6), 545–564 (2015)
11. Massey, A.K., Otto, P.N., Hayward, L.J., Antón, A.I.: Evaluating existing security and privacy requirements for legal compliance. Requir. Eng. **15**, 119–137 (2010)
12. Massey, A.K., Rutledge, R.L., Antón, A.I., Hemmings, J.D., Swire, P.P.: A strategy for addressing ambiguity in regulatory requirements. https://smartech.gatech.edu/handle/1853/54573 (2015)
13. Massey, A.K., Rutledge, R.L., Antón, A.I., Swire, P.P.: Identifying and classifying ambiguity for regulatory requirements. In: 22nd International Conference on RE, pp. 83–92, August 2014
14. Nigam, A., Arya, N., Nigam, B., Jain, D.: Tool for automatic discovery of ambiguity in requirements. Int. J. Comput. Sci. Issues **9**(5) (2012)
15. Osborne, M., MacNish, C.K.: Processing natural language software requirement specifications. In: 2nd International Conference on RE, pp. 229–236, April 1996
16. Otto, P.N., Antón, A.I.: Addressing legal requirements in RE. In: 2007 15th IEEE International RE Conference, RE 2007, pp. 5–14 (2007)
17. Popescu, D., Rugaber, S., Medvidovic, N., Berry, D.M.: Reducing ambiguities in requirements specifications via automatically created object-oriented models. In: Paech, B., Martell, C. (eds.) Monterey Workshop 2007. LNCS, vol. 5320, pp. 103–124. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89778-1_10
18. Umber, A., Bajwa, I.S.: Minimizing ambiguity in natural language software requirements specification. In: 2011 Sixth International Conference on Digital Information Management, pp. 102–107, September 2011
19. van Bussel, D.: Detecting ambiguity in requirements specifications. PhD thesis, Tilburg University (2009)