Requirements and compliance in legal systems: a logic approach

Waël Hassan[1] Luigi Logrippo[1,2]
[1]*University of Ottawa,* [2]*Université du Québec en Outaouais*
wael@acm.org, luigi@uqo.ca

Abstract

It is shown that the concepts of requirements and implementation exist in normative systems, in particular in law, and are similar to homologous concepts in software engineering. Concepts of compliance and conformance are also similar in the two areas. Further, it is shown how a logic analyzer such as Alloy can be used in order to verify legal compliance by checking consistency between legal and enterprise requirements. Examples are taken from privacy law and financial reporting law.

1. Introduction

Legal systems are traditionally expressed in natural language. However, increasingly, laws include norms that were created with the intention of determining, directly or indirectly, the operation of computing systems. These norms must be implemented in software, which is based on formalized languages, directly or indirectly executable by computer programs. Ideally, a translation mechanism should exist, be objective and repeatable to reflect changes in the law.

However translation between natural language and formalized language presents well-known challenges. Usually, not all the natural language text can be translated, and many assumptions must be made. It is conceivable that some laws including computer code be adopted, but for now this is very rare.

As enterprises must adhere to norms, they strive to correctly translate legal requirements and automate *legal compliance* checks. Privacy and access to information and financial laws are primary areas of concern, they belong to the general area of enterprise legal governance. This area presents several challenges such as (a) determining applicable laws and understanding interdependencies of domain and jurisdiction (b) extracting legal requirements (c) validating extracted requirements for consistency and compliance (d) implementing legal requirements using software architecture. We concentrate on the third challenge.

Since legal systems are normative [1], many of the observations of this paper are valid beyond law, and we will sometimes talk in more general terms than just law. Examples of normative systems that are implemented in software or hardware are information protection systems including firewalls and access control systems, computer networks regulated by policies, e-business and e-governance systems.

2. Contributions and related work

In Section 3, the role of ontologies for the proper representation of legal semantics is discussed. Section 4 presents various methods for the validation of normative requirements. Section 5 discusses normative levels. Section 6 provides a high-level description of our compliance verification method, with some details and examples given in Section 7.

We propose a compliance and consistency validation method that represents and combines legal and enterprise requirements. The method uses ontologies for capturing enterprise definitions and first order logic to represent requirements. The method is able to detect violations that are consequence of lack of compliance and inconsistency.

Deontic logic has been widely used for representing normative rights and obligations. However, we focus on the effectiveness of non-modal, first order predicate logic and ontologies in their ability to represent legal requirements and automate legal compliance and consistency checks in a practical setting. Thus far, we have not compared the two approaches, but we have been able to prove by example the capabilities of our method.

Sartor and others [7][10][12] proposed logic abstractions of legal concepts. In addition, [4][5] have discussed regulatory compliance. Our approach differs from others in this area by the use of ontologies combined with first order logic. Pioneering work in this subject was done in [7].

Related research on extraction of requirements from legal script is presented in [4][9][16]. Methods of implementing laws include legal programming [13]. Our proposed roadmap is similar to ones proposed by [2][16].

3. The role of ontologies

In order to present legal semantics precisely, we need ontologies, usually defined as formal representations of sets of concepts within a domain, together with the relationships between these concepts. These are often implicit in law. A very basic example is family law, which cannot be understood without reference to family ontology. In order to understand laws related to enterprise governance, we need to refer to enterprise ontologies, where enterprises are defined as hierarchical structures including departments or roles, to which it is possible to assign processes or steps [5]. Ontologies can also be explicitly specified by laws, e.g. privacy law may specify that *consent can be received through a signature, a check-off box, or verbal acknowledgment*: this requirement establishes an equivalence relationship between these methods. With human assistance, the concepts of law must be mapped into concepts represented in formal ontologies. For example, a legal ontology can determine applicable laws in e-commerce transactions with multiple parties and cross jurisdictions. Consider *using a credit card to pay for a doctor's note*: such a transaction is subject to the provincial healthcare privacy law as well as the federal privacy law dedicated to commercial activity. A legal-ontology can represent a structure of applicability. This helps in situations where vertical laws are combined in the same jurisdiction and horizontally at the municipal, provincial, and federal levels. For a discussion of legal ontologies, see [15].

4. Validating normative requirements

Once a precise translation of parts of a law or regulation has been done, it must be validated, as it will be seen in the following examples.

Scenario validation is a partial method of testing compliance. An enterprise user, usually an accountable enterprise authority, should have available a tool that provides the results of test case scenarios. For example, a privacy policy of a company specifies that credit card information must be removed from the company's data base after the transaction purpose is achieved. It should be possible to test this case to validate that the information is in fact removed. This is of course useful but testing may not show certain violations. It is quite possible for example that the information will be removed from one data base, but not from others [5], through information sharing, or it is possible that it will be removed in the scenario tested, but not in others. Testing may also ignore user's real intentions. For example, an administrator issues a user specific right, and later on, by mistake, revokes that right through a group policy. A priority scheme of 'deny overrides' would ignore the original intent of the user-specific right, and testing may not show this.

Consistency checks can combine legal and enterprise requirements for validation. They can determine that norms are mutually inconsistent, independent of conflict resolution rules such as 'deny override', 'permit override' or others [11]. In the example above a consistency checker could detect a conflict between user specific right and the group policy. It would detect that the group policy is violated at inception, and the system utilizing the checker could ask whether the user-specific right should be up-held as an exception. An enterprise may wish to check whether an enterprise policy, as a whole, is consistent with applicable laws. The feasibility of these checks is dependent on the characteristics of available analysis tools. Some such packages, often conceived for verification work in software engineering, use highly optimized SAT solvers. They are able to find inconsistencies within bounded scope; an example is Alloy [6], which currently uses the Kodkod [14] SAT solver.

One may also wish to *check completeness*. For example, a city could have a regulation saying that parking on downtown streets is forbidden. Related regulations impose different fines for different named streets, but somehow Murray Street, a downtown street according to an ontology, is not mentioned. Unfortunately in practice this can be a difficult test to implement, because there may be too many cases to consider, especially when continuous domains (time, amounts…) are involved.

5. Normative levels

Laws can express principles at different levels. Two levels are particularly apparent, we call them *rule level* and *requirements level*. By the name of their apparent 'inventors' we could call them the Hammurabi level and the Moses level [10].

- The *Rule level* has cause-effect rules, similar to Event-Condition-Action (ECA) rules in data bases: *if a person does not pay debt, their possessions will be sold*. Rule level specifies the final *implementation* of law, which can be capable by itself of enforcing a legal system. Normative systems can exist at this level only (e.g. the Hammurabi code, XACML policy systems[11], firewalls).
- The *Requirements level* expresses desirable states of affairs, a situation that 'ought to be' [8], such as: *debts must be repaid*. It appears that this level cannot exist alone, and that it depends on the rule level for

enforcement and ultimate effectiveness. The enforcement of the Moses code depends on other rules, notably rules dictating sanctions in the case of violations.

In addition to these two, we have mentioned:

- The *Ontology level* which is orthogonal and expresses the domain structure, usually common to both previous levels: *Financial Controllers should report to CFO* is a requirement that requires reference to an enterprise ontology for its implementation.

These levels, and other intermediate ones that normally exist, may not be explicitly distinguished, and may not be explicitly present, but need to be made explicit for analysis. An example of requirement-level norm in PIPEDA[1] is the "Accountability Principle-1" which states that *An organization is responsible for personal information*. In the same law, one also finds rule level norms such as the "Consent Principle-3": *when an individual expresses a withdrawal of consent, the organization needs to inform the individual of the implications*.

The use of the terms: *requirements* and *implementation*, widely used in software terminology [17], has been intentional, since in our view there is a similarity between legal theory and software theory in this classification. Requirements and implementations are kept separate in software engineering. Perhaps it should be concluded that they should be kept separate in law as well; however many laws include requirements and rules without any clear distinction, and we have just seen an example.

Issues:

1. Deriving rule level norms from requirement level norms. For example, Sarbanes-Oxley[2] (SOX) section 404 asserts that approvals cannot be granted to transactions initiated in other departments (separation of concerns). This can be implemented through an ECA rule *if initiator is in different department then deny access to approval action.* In the presence of a sound ontology, this translation could be partially automated; templates of cause effect rules could be produced, to be completed by human intervention. As a further example, the enterprise accountability principle mentioned previously is a declaration of a fact. The *accountability* fact may be further implemented using other provisions in the law such as, *an organisation shall designate an individual or individuals who are accountable for the organization's compliance*. This can be implemented further in ECA `rule` form. We have found evidence of recurring translation techniques, leading to the concept of *translation patterns*. Patterns can help by identifying common solutions to recurring problems. Our experience has revealed several such patterns: accountability, responsibility, separation of concerns, etc.
2. Determining the logical consistency of coexisting requirements, possibly at different levels and of different origins, including the related ontologies. The relation between compliance and consistency will be discussed in the following section.
3. Determining completeness of rules with respect to requirements. PIPEDA specifies: *all collected data should be used solely for its intended purpose.* This requirement can be refined into a number of requirements such as: *collect data for a purpose; restrict access to data unless purpose is valid; destroy data once purpose is achieved*. These requirements can be further translated into ECA rules. Completeness and consistency checks can validate if the rule level policies satisfy requirement level policies.

6. A method for compliance verification

Our method validates compliance and consistency of normative requirements originating from two sources: enterprise regulations and the law. We define *compliance* (sometimes also called *conformance,* a term used in software engineering with a similar meaning) as the mutual consistency of legal requirements and enterprise requirements. It could be said that there are two aspects to compliance: completeness and consistency. However often completeness reduces to consistency, because if an implementation is incomplete with respect to requirements, then scenarios may exist that are inconsistent with the requirements.

Our semi-automated compliance detection method involves a high-level language and tools [5]. It consists of the following steps, see Fig. 1: a) Representation of legal and enterprise requirements b) Generation of a logic model c) Logic analysis..

In the first step the normative presentation includes legal provisions at several levels. We discovered three types of requirements. The first is structural; these requirements could either be enterprise or process hierarchy requirements. Norms of this type specify how a business process is formed. For example: *Every process should have a secure disposal activity*. We can also have hierarchy requirements, such as: *The company's board of directors should include the chief financial officer and internal financial auditor*. Another type of requirement may suggest specific user assignments, for example: *A chief financial officer*

[1] PIPEDA is Canada's Personal Information Protection and Electronic Documents Act

[2] Sarbanes-Oxley is a financial reporting law in the U.S.

should be assigned to the task of selecting an audit firm. Ontologies may define equivalence relations between activities or enterprise roles.

The second type of requirements is logical. This type consists of ontology requirements expressed in first order logic. Examples are: *there exists a financial officer; if there is a central secure data disposal process there is no need for secure disposal activity in each process; if the company does not have a board of directors or if the board has been dissolved then the financial submission activity is halted.*

The third type of requirements is at a higher level. Under this category, high-level requirements can be decomposed into ontological or logic requirements. e.g.: *an enterprise is accountable for private information; an enterprise must report its financial information quarterly to the Securities and Exchange Commission.*

In summary our language is able to directly represent statements of the first two types, whereas statements of the third type need to be re-written manually in the first two types.

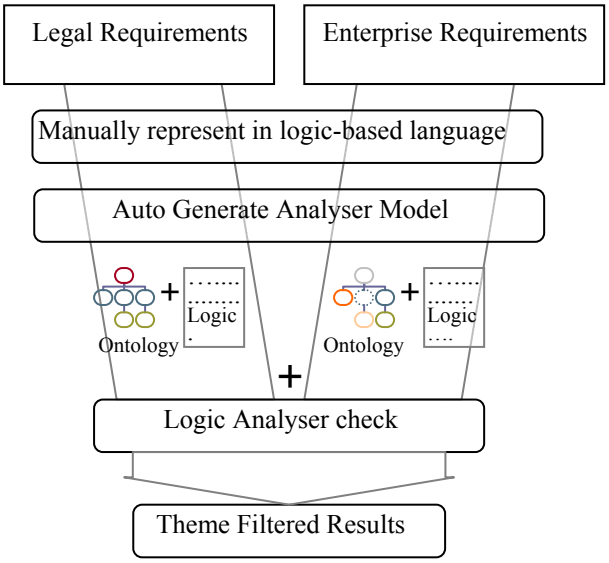

Figure 1. Compliance Checking Method

Figure 1 shows the principle of our method, establishing a repeatable compliance process. One sees on the left hand side that the compliance validation process starts from enterprise law translated into our own logic-based legal requirements specification language. Similarly, on the right hand side, we start from an enterprise requirements specification.

The tool we have implemented takes as input our specification language and generates a logic analysis model representing requirements. The logic analysis model can be understood as a ontology representing structural and logic requirements.

The tool then passes the constructed model to the Alloy logic analyser, mentioned in Section 4. In addition, the tool creates *theme filters*, these are meta-files that are able to filter output based on entity type and relations. During analysis the tool visualizes complex enterprise entities including processes, departments and roles, user assignments and their relations. Theme filters are able to project this complex state of affairs into various *views* for detailed analysis.

In other words, having received the model and associated theme filters, the Alloy analyzer detects compliance or produces a counterexample. In the second case, the analyzer output displays a complete violating model showing the instance objects and associated relations. The output is stored in various formats and it is possible to create visual displays focused on one view on the basis of one of the theme filters. The intention is to give the user the ability to do several analyses based on various views.

7. Implementation details and examples

The analysis model is governed by the capabilities and limitations of the tool of choice Alloy: essentially, first-order relational logic operating over ontology specifications. Our experience shows that the system is suited for the analysis of enterprise normative requirements. The tool is able to join and disjoin logical assertions. Such a capability enables an enterprise officer to validate concurrent assertions. It can also simulate an instance model particular to a specific scenario or refute a certain argument. We can ask the tool to *generate a possible instance of a banking model where requirements of lending are in conflict with those of borrowing*, if both sets of rules are well defined. Another possibility is to validate an assertion to ensure that: *all forwarded data has been preceded by a signed agreement*. However, the tool may not be able to assert that data is securely disposed of internally and externally in all cases once the purpose has been achieved, since Alloy can only produce verdicts within bounded scope.

We were able to validate compliance of legal norms to enterprise specifications, validate consistency, and detect interactions. In the privacy domain we have studied the effects of delegation of authority and accountability on enterprise ontology and process, as specified by law. Our examples showed violations related to collection, retention, and distribution of private data. In the financial domain we have validated examples related to separation of concerns, delegation

of authority, and basic access control conflicts. We have also been able to capture financial requirements such the ones given in SOX section 404. We were able to validate consistency of enterprise and legal requirements with respect to a combination of financial and privacy laws.

Our experience, documented in [5], shows that privacy and financial reporting audits can be assisted by the use of a tool such as the one we have briefly described.

Our tool is able to uncover compliance problems and assist in localizing violations. A visual interface serving multiple theme analysis is an added benefit.

8. Conclusion and Future Work

We have argued that the concepts of requirements and implementation (rules) exist in normative systems and law as they exist in software engineering. They are not always clearly distinguished in law, however this distinction should be done to improve analysis. On this basis, we have categorized various types of challenges facing normative systems: deriving rule level laws from requirements level laws, as well as validating consistency, completeness and compliance of systems of norms at various levels. The key role of ontologies was also mentioned.

We follow existing approaches, however we offer a specific method proposing that legal and enterprise requirements can be validated for compliance using logic analyzers and logic models including ontologies. We have implemented consistency and compliance checks using a tool that takes as input our legal and enterprise requirements language and produces a logic analysis model for validation. Our normative examples are taken from privacy and financial disclosure laws such as PIPEDA (Canada) and SOX (U.S.A.). In related work, we have considered enterprise requirements, e.g. from the Royal Bank of Canada. In various ways, we have demonstrated our ability to detect inconsistencies and non-compliance between laws and enterprise regulations.

We have been able to detect compliance violations in different practical scenarios: e.g. violations due to information leakage and process dependency. Inconsistencies were detected in examples of conflicts involving separation of concerns and process structural requirements. Other examples include delegation of authority conflicts with privacy requirements.

Immediate future work will focus on patterns of translation of requirements to rules.

9. Acknowledgment

This work has been supported in part by grants of the Natural Sciences and Engineering Research Council of Canada. We are grateful to the anonymous referees for several suggestions.

10. References

[1] Alchourròn, C.E., Bulygin, E.: *Normative Systems*. Springer, 1971.

[2] Antón, A. I., Bertino, E., Li, N., and Yu, T. : A roadmap for comprehensive online privacy policy management. Comm. ACM 50, 7 (Jul. 2007), 109-116.

[3] Breaux, T. and Antón, A. : Analyzing Regulatory Rules for Privacy and Security Requirements. IEEE Trans. SE 34, 1 (Jan. 2008), 5-20.

[4] Brodie, C. A., Karat, C., and Karat, J. : An empirical study of natural language parsing of privacy policy rules using the SPARCLE policy workbench. In SOUPS '06, ACM Internat. Conf. Proc. Series Vol. 149, 8-19.

[5] Hassan, W., Logrippo, L.: Validating Compliance with Privacy Legislation. Submitted for publication.

[6] Jackson D. : *Software Abstractions: Logic, Language, and Analysis.* MIT Press. March 2006.

[7] Jones, A.J.I, Sergot, M.: Deontic logic in the implementation of law: Towards a methodology. AI and Law, 1(1), 1992, 45-64.

[8] Kelsen, H.: *General Theory of Law and State*. Harvard University Press, 1945.

[9] Kiyavitskaya, N., Zeni, N., Breaux, T. D., Antón, A. I., Cordy, J. R., Mich, L., and Mylopoulos, J. : Extracting rights and obligations from regulations: toward a tool-supported process. In Proc. ASE '07, 429-432.

[10] Logrippo, L.: Normative Systems: the Meeting Point between Jurisprudence and Information Technology. SoMet 2007, Rome, 343-354 .

[11] Mankai M., Logrippo L. : Access control policies: Modeling and validation. In Proc. of the 5th NOTERE Conference, 2005, 85-91,

[12] Sartor. G. *Legal Reasoning: A Cognitive Approach to the Law.* Berlin: Springer. 2005.

[13] Subirana, B. and Bain, M. 2006. Legal programming. Comm. ACM 49, 9 (Sep. 2006), 57-62.

[14] Emina Torlak, Felix Sheng-Ho Chang, Daniel Jackson: Finding Minimal Unsatisfiable Cores of Declarative Specifications. FM 2008: 326-341

[15] Valente, A. : *Legal Knowledge Engineering.* IOS Press, 1999

[16] Vàzquez-Salceda, J., Aldewereld, H., Grossi, D., Dignum, F. : From human regulations to regulated software agents' behavior - Connecting the abstract declarative norms with the concrete operational implementation. A position paper. Artif Intell Law (2008) 16:73–87.

[17] Zave, P. and Jackson, M.: Four dark corners of requirements engineering. ACM Trans. Softw. Eng. Methodol. 6, 1 (Jan. 1997), 1-30.