Towards a Process for Legally Compliant Software

Waël Hassan
Privacy in Design
Toronto, ON, Canada
whassan@privacyindesign.org

Luigi Logrippo
Université du Québec en Outaouais and University of Ottawa
Ottawa-Gatineau, Canada
luigi@uqo.ca

***Abstract*— We propose a method and a process for legal software requirements extraction and compliance checking. We describe a requirements extraction model, a set of rules for specifying the format of the extracted information, a set of UML-based principles for translating the extracted information into a language based on predicate logic, and finally, a tool that analyzes the resulting logic model and displays the results of the analysis. The translation principles are based on a Governance Analysis Model (GAM) which is described in UML; the language is our Governance Analysis Language (GAL) and the tool is our Governance Analysis Tool (GAT). MIT's logic analyzer Alloy is the engine on which GAT runs. GAL is translated into assertions in Alloy's language and the Alloy tool can find counterexamples indicating situations of non-compliance.**

***Index Terms*— Software requirements, formal method, legal compliance, logic analysis, software design process, privacy law**

I. Introduction

Increasingly, organizations use software products, such as web services, to gain efficiency and consistency in governance. Electronic governance software products must comply with legal requirements; in addition they must comply with the requirements of the organization. Organizational requirements must comply with legal requirements as well. Therefore, we have a triangle of entities that should be in relationships of mutual compliance: legal requirements, organizational requirements, and software product. It seems natural then to think of software engineering methods that can be used to develop software where these relationships are taken into consideration during construction. Such methods are part of the idea of Privacy by Design (PbD) [8]. This concept guides the research reported here. Our method consists of processes that are well understood in software engineering and can be adapted for the purpose.

This paper is organized as follows. Section II introduces the general framework for the method. We propose here a process for the development of legally compliant software. Section III introduces our concepts of compliance which, for the purpose of this paper, is essentially consistency. Section IV presents the UML-based GAM model that is the foundation of our method. Section V and VI provide examples to illustrate how the method can be applied, Section VII presents discussion of related work. Section VIII concludes the paper. Appendix I presents Alloy and Appendix II presents our language GAL.

II. Overview of the Process and its Steps

Fig. 1 outlines the main steps of the method and process that we propose, which will be illustrated in the rest of this paper. On the left side of Fig. 1 we are concerned with the legal normative, that is, legal requirements as specified first in the legal texts and then in some software-oriented requirement language. On the right-hand side we see the organizational requirements, their specification in a software-oriented requirements language and then the resulting software that must comply with both legal and organizational requirements.

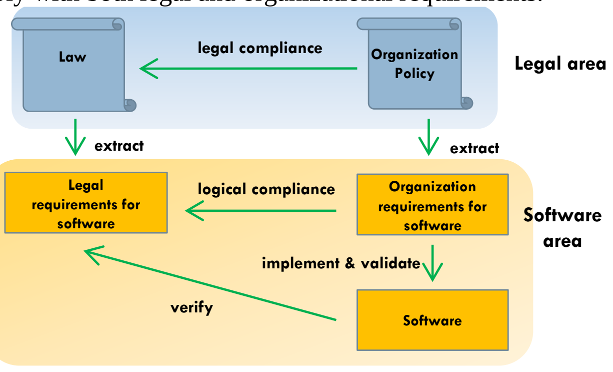

Figure 1: The main steps of a generic process

We shall see that, in order to facilitate the software process, it is convenient to represent the software requirements in a formal language based on a formal model, in our case GAL and GAM. This yields the following refined process, to which we shall refer henceforth:

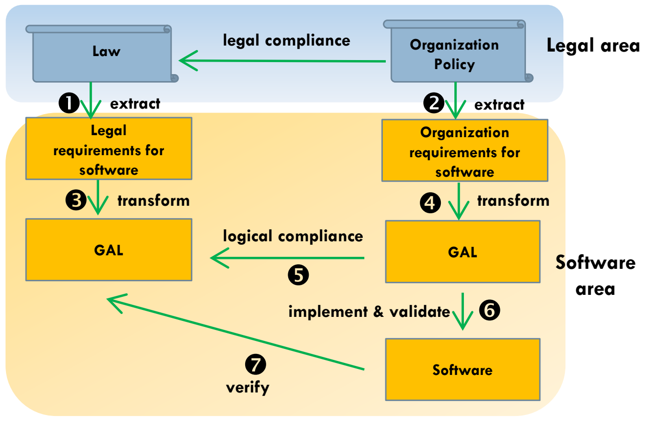

Figure 2: Our refined process

A. *The Legal Area*

Two parallel normative worlds are relevant for the development of software for an organization. One is the legal system of the jurisdiction, and the other is the system of policies and regulations of the organization. The second must be legally compliant with the first. For example, organizational requirements and policies must comply with the law with respect to privacy. Checking and ensuring this compliance is a legal process that must be completed by legal and administrative experts before the software process starts. In other words, we assume that before the software development process starts, organizational policies have been brought into line with the applicable laws.

B. *The Software Area: Overview*

To start the software process, legal and organizational requirements have to be stated in terms understandable to both legal experts and software experts. This *extraction* process, represented in Fig. 2 by steps 1 and 2, is best carried out by a joint team of legal and software experts, or a team of people equally comfortable in both worlds.

In the extraction process of our method, the elements of organizational structures and their relationships are specified using our Governance Analysis Model (GAM) (Section IV). The process is comprised of three passes. In the first pass, elements of organizational structures are extracted and normalized. In the second pass, the relationships among pairs of elements are extracted and normalized. In the third pass, gaps are mitigated and inconsistencies are solved.

For automatic processing, it is necessary to express the resulting models into a formal language. In our Steps 3 and 4, which are collectively called the *transformation* steps, the elements and relationships extracted during the extraction step are expressed in the Governance Analysis Language (GAL), a language capable of expressing many types of legal and organizational requirements. Details about the language are given in Appendix II. The result of Step 4 is a precise, software-oriented specification for the software product. The result of Step 3 is a series of statements representing the legal requirements.

One could assume that the requirements specified by Step 4 are necessarily aligned with the law, because of the work that was done in the legal area. We propose, however, that a second compliance check be done in the software area, not only because the extraction and transformation steps can introduce errors, but also because some situations of non-compliance can be more easily discovered at this point. Software tools are available to double-check that the precise software requirements are consistent with a precise interpretation of the legal requirements (Step 5, Section V).

Our Governance Analysis Tool (GAT), a compiler from GAL to the language of the Alloy logic analyzer, uses Alloy functionalities in order to display the result of this check. In the ideal case, Step 5 will simply confirm that the 'legal compliance' step and the extraction processes have led to the development of legally compliant software requirements. Otherwise, GAT will yield counterexamples showing cases where GAL organizational requirements are not compliant with GAL legal requirements, probably leading to corrections in the former, even as far back as the organization policy. This process may require several iterations.

Once this process is terminated, Step 6 initiates the implementation and validation process. The methods used can follow established software practice. "Validation" here means checking that the implemented software meets the organizational requirements. It could then be assumed that the resulting software will be compliant with legal requirements. We expect, however, that in the last step of the process – Step 7 – the software will be again checked against the legal requirements. Such final checks are common in engineering. For example, a bridge that has been developed according to specifications and engineering principles and which therefore should be safe, receives final tests by having very heavy loads passed over it, up to the requirements initially specified. Similarly, in Step 7, test cases could be extracted from the legal requirements and used to ensure that the software does, in fact, measure up to requirements [29].

III. COMPLIANCE AND CONSISTENCY CHECKING

In normative and legal texts, 'compliance' is a term often used but rarely defined. There is some literature describing how the notion of compliance in law can be understood in terms of formal logic, normative concepts, and software requirements. From the point of view of software requirements, Zowghi and Gervasi [38] make the point that compliance can be understood either as a combination of consistency and completeness, or, more pragmatically, as satisfaction of business goals. This second view agrees with the one of Cannon and Byers [7], who state that "Compliance is simply about ensuring that business processes are executed as expected." According to Governatori and Sadiq [14][33], the term compliance "is often used to denote and demonstrate adherence of one set of rules … against another set of rules". The authors continue to say that, "ensuring compliance of business processes with a normative document means ensuring consistency of norms stated in normative documents and rules covering the execution of business processes". To rephrase this idea, a normative statement A is compliant with a normative statement B if the logical formula expressing A is consistent with the logical formula expressing B: their conjunction is satisfiable, which means that it does not imply False. Neither A nor B needs to imply the other. In practice, our consistency checking tool will be the logical analyzer Alloy, presented in Appendix 1.

The requirements (whether legal or organizational) can be internally inconsistent or contradictory. Based on Alloy, our tool will detect any such inconsistencies at the outset and will make it impossible to proceed further until they are repaired.

An important shortcoming of our definition is that it does not address completeness: a normative statement could be vacuously compliant with another, for example if it says nothing or if it addresses different concerns. This means that, in its current formulation, our method can be used to check whether *some* selected requirements are satisfied, but it cannot provide assurance that *all* significant requirements are satisfied. Several methods for identifying *all* requirements have been proposed in the literature [4][11][37].

It can happen that the legal requirements and the organizational requirements are expressed at such different levels of abstraction or in such different terms that the logical compliance concept that we explore here is not clearly applicable. Other researchers are investigating similar situations [4][5][11].

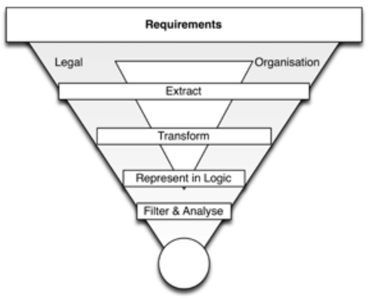

Figure 3: Alternative view of Steps 1 to 5

With these points in mind, Fig. 3 shows another view of the first five steps of Fig. 2, where the two bottom steps of Fig. 3 detail Step 5 of Fig. 2. The requirements, legal and organizational, are extracted and then transformed into an appropriate formal language, such as our GAL. The resulting statements are transformed into an appropriate logical notation, such as the Alloy language. The Alloy tool can then be used to analyse and filter for logical consistency.

To facilitate a direct comparison between legal and organizational requirements, when this is possible, a single model and language should be used for both. This is a common approach among researchers in the area [9][12][33][36].

IV. THE GAM MODEL FOR THE EXTRACTION PROCESS

We enter now in the details of our method. Preliminary versions of parts of what follows were presented in [16] [17] and a more complete version in [18]. However this paper does not follow Reference [18] exactly, since our thinking has evolved in the meantime.

As mentioned, our language GAL is based on an UML model of organizational concepts, called GAM. We will present now some concepts that are at the basis of GAM and GAL. The primary responsibility of the model layer is to define a language that describes the information domain.

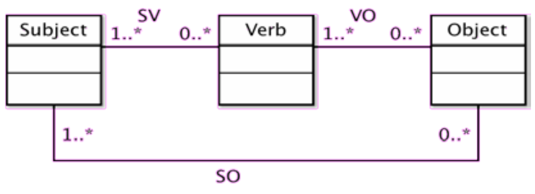

Figure 4: The SVO meta-model

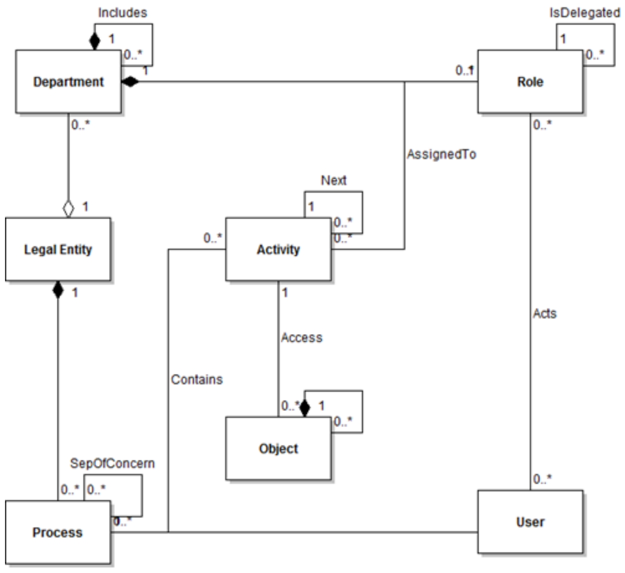

Figure 5: The GAM model

Meta-model: A meta-model is a layer of abstraction that contains a minimal set of concise and precise components. A meta-model can be refined in multiple models, and there can be multiple meta-models associated with a model. A model is an instance of a meta-model.

Base-model: The base-model (a meta-model) proposed in this paper includes three primitive components: Subject, Verb and Object (Fig. 4). A subject represents a rights holder. A verb is an action or a right given to a subject. An object is the object of a right. While of course this base-model is too restricted for general legal thinking, it works well in the area we have selected, as we shall see.

Our extraction process is based on the organizational concepts expressed in the Governance Analysis Model or GAM, of which a significant portion is shown in Fig. 5. The GAM is a refinement of the SVO meta-model of Fig. 4, where User corresponds to Subject, Activity corresponds to Verb and Object corresponds to Object.

This diagram attempts to capture some concepts often found in organizational laws and regulations. Using the diagram, one can recognize in the model the main elements of organizational structures and their relationships. Tables 1 and 2 capture essential information about those entities and about the relationships that exist between pairs of elements.

Note that each organization will have its own diagram, corresponding to its policies. In our diagram, there is a relation *Separation of Concerns* for *Process*. This could be implemented by a constraint preventing any user from being assigned two processes subject to the policy of separation of concerns. However, if Role Based Access Control (RBAC) is implemented in an organization, Separation of Duties (SoD) could be stated as a relation between Roles [3][10].

Table 1: Elements of organizational structure

Elements	Explanation
Activity	A function that a person participates in
Department	A distinct, usually specialized division of a large organization
Legal Entity	An association, corporation, partnership, proprietorship, trust, or individual that has legal standing
Object	The purpose of an activity
Process	A series of actions or steps taken to achieve a result
Role	The function assumed by a person
User	A person who uses or operates an application, equipment, or system

Table 2: Relationship Table

Relationship	The two elements connected	Explanation
Access	Activity & Object	An activity grants access to an object
Acts	User & Role	User acts in a role
AssignedTo	Role & Activity	A role is assigned to an activity
Assumes	User & Process	A user can be associated with a process
Composed Of	Department & Role	A department can have many roles
	Object & Object	An object can be composed of objects
	Department & Legal Entity	A legal entity contains many departments
	Legal Entity &Process	A legal entity contains many processes
Contains	Activity & Process	An activity is contained in a process
Includes	Department & Department	A department can consist of many sub-departments
IsDelegated	Role & Role	A role can be delegated to another role
Next	Activity & Activity	An activity starts after another activity is finished
SepOfConcern	Process & Process	Two different processes cannot be associated with the same user

Following the structure of entities and relationships presented in this diagram, a number of requirements have been specified and checked, as will be seen in the following examples.

Many related formalisms and models can be found in the literature [28], but these are not targeted to the extraction tasks required by our method.

V. Specifying and Checking the Relationships

The 'Contains' relationship of Fig. 5 is used to describe how activities are contained within processes. In a very small organization, the relationship could be as follows:

Contains (Loans, PublishApplication)
Contains (Loans, ReceiveFilledApp)
Contains (Loans, Wapplication)
Contains (Loans, JReceiveFilledApp)
Contains (Loans, ConsentClient)
Contains (Loans, LegalReasonException)
Contains (Loans,ThankClient)
Contains (Loans, DisposeData)
Contains (OrderMgt, ReadApplication)
Contains (OrderMgt, ValidateInfo)
Contains (OrderMgt,SaveInfo)

Already we can check some very simple properties, for example: Does the Loan activity contain a process DisposeData? Formally, this amounts to checking whether the set of relationships specified above is consistent with a property

Contains (Loans, DisposeData)

which is trivially true because the set of relationships immediately includes this property, but one can think of organizations where processes and activities are deeply nested and an automatic checking tool can find the answer.

The following specifies the "Next" relationship of Fig. 5 between activities:

Next (ValidateInfo, SaveInfo)
Next (ReadApplication, ValidateInfo)
Next (Wapplication, JreceivedApp)
Next (JReceivedApp, ConsentClient)
Next (JReceivedApp, LegalReasonException)
Next(ThankClient, DisposeData)
Next (PublishApplication, ReceiveFilledApp)
Next (ReceiveFilledApp,Wapplication)
Next (ValidateInfo, WApplication)
Next (WApplication, ReadApplication)

On the basis of these relationships, we can check the following legal requirement:

PIPEDA[1] Requirement:

The fifth principle of PIPEDA expressed in provision 4.5.3 requires that personal information shall be retained only as long as the purpose of collection is not reached.

In other words:

Validate that no information is retained once its purpose is achieved. This requirement can be validated by using the following GAL statement:

Activity-Trace-All(ReceiveFilledApp,DisposeData)

This statement checks whether in all traces, the activity *ReceiveFilledApp* is followed by the activity *DisposeData*.

The diagram in Fig. 6 is an edited version of the diagram that was obtained by running *Activity-Trace-All*. The organization may be in violation of Provision 4.5.3. It can be seen that the step *WApplication* leaks information to the process *OrderMgt*. The scenario check has found that *ReceivedFilledApp* is not followed by *DisposeData* in this second process. The related assertion has failed and this has caused the production of this diagram, which shows that the main process on the left correctly disposes of the data, while the process on the right saves it.

Alloy has a filtering facility that is useful to select the information that will be displayed.

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

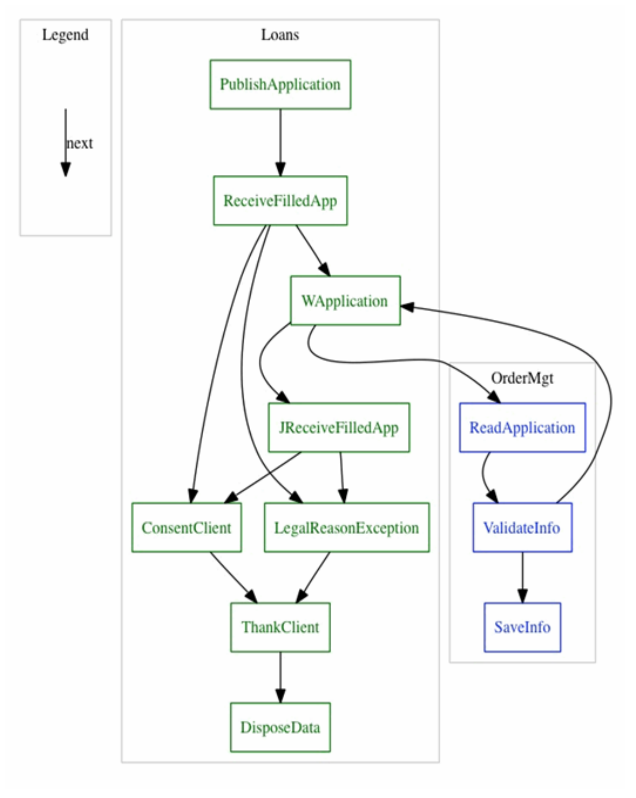

Figure 6: Result of "Next" relationship check

We briefly describe here other examples that we have developed in our work.

1. PIPEDA requires that organizations name individuals as privacy officers. This is a structural completeness requirement. One way to satisfy this requirement is to include in the definition of the organization an *Acts* statement that assigns a named user to the role of Privacy Officer. The existence of this user can be checked by using *CheckActs(Label, PrivacyOfficer, ANY)*.
2. The same law also requires the existence of a privacy process in the organization. This is also a structural completeness requirement. One way to satisfy this criterion is to include an appropriate *ComposedOf* statement in the structure of the organization. The presence of this statement can be checked by using *Process-Includes*.
3. Separation of concerns motivates a whole family of other examples such as the following: In an organization, it could be stated that individuals who have access to ProcessA cannot have access to ProcessB. In GAL, this can be expressed by a *Separate* statement. As mentioned in [10] the duties 'Check Preparation' and 'Check Issuing' are often separated, i.e. they cannot be assigned to the same user. In an organization, this constraint could be represented by two processes in the relation of Separation of Concerns. If a user's role allows the user to be assigned an activity that can contain both processes, then the constraint is violated. This will be shown by our tool as a counterexample to a relational constraint expressed by the relational operator *Separate*; see Appendix II Section 3. A similar result is obtained if Separation of Concerns can be violated by way of delegation.

Again, although these checks can be performed easily by hand if there are only a few rules in an organization, computer assistance will be very helpful for larger sets of rules.

It could be asked, what do diagrams such as the one of Fig. 6 really indicate, given the fact that they omit many details. As in similar application areas, including some types of error messages provided by compilers, our tool's diagnostics do not necessarily imply the presence of compliance issues. They should rather be interpreted as warnings for which further checking should be carried out.

VI. PERSONAL HEALTH INFORMATION PRIVACY

The applicability of our method for legal compliance audits is further shown by the following real-life example. Suppose a person has been tested HIV-positive, and his lab report has been stored in the Ontario Laboratory Information System (OLIS) [30]. As a result of his condition, he has been prescribed several medications, including a drug that stalls reproduction of HIV. His medication history is stored in the Ontario Drug Benefit (ODB) [31] database, which contains prescription claims data for eligible recipients. This person is worried about his employment prospects given that he is HIV-positive, so he places a consent directive (a policy to deny access) to block access to his test results, and the OLIS does so accordingly. However it may be possible to learn about the patient's HIV positive status by accessing the patient's medication history through the ODB database.

This example has characteristics similar to the example of Section V. An auditor can discover this possible leak by modeling the law, the healthcare institution, and patient consent policies. In this example, the User is the patient, the Activity is "block-access", and the Object is all HIV-related information. Following this discovery, the auditor can recommend that the organization create an object structure that binds HIV-lab results with HIV-Medication. Therefore, a block to disclosure of HIV-related information would result in blocking both OLIS and ODB records.

VII. RELATED WORK

The problem we are addressing is quite generic, and many approaches are possible. Several of the methods that we list below are mutually orthogonal or complementary and can be expected to merge as the field matures.

Antòn, Bertino et al. [1] propose that the specification and analysis of legal requirements be done through the development of a formal language, supporting information flow analysis based on policies. This paper follows another paper by a related team [19] where a semi-automated method was presented to identify inconsistencies between requirements specifications, database design, and organizational security and privacy policies. Their ReCAPS method is based on a number of heuristics quite different in nature from ours. Related work was presented in References [23][4].

To support specification of requirements, Hruby [20] provides a meta-model based on resources, events, and agents (REA); REA was originally proposed by McCarthy [27] as a generalized accounting model. Similarly, Brodie et al. [6]

discuss regulatory compliance. They propose that formal compliance validation requires a rigorous method based on formally defined syntax and semantics in addition to a formal validation method. This method was applied to industrial case studies recognizing the ability of computing systems in the compliance validation process [24].

Many authors have proposed first order logic abstractions of legal concepts. Sartor [34] contains an extensive study of various types of legal statements with first-order logic interpretation. Many authors have also demonstrated the use of deontic logic in this context. Deontic logic is more appropriate than first-order logic for expressing deontic concepts, however many legal concepts can be efficiently expressed in simple first-order logic. An interesting tool based on deontic logic has been presented by Governatori et al. in several papers [12] [13][14].

Formal methods for capturing the concepts of laws have been surveyed by Otto et al. [32]. This paper lists several techniques including the use of symbolic logic, knowledge representation using PROLOG, deontic logic using LEGOL, defeasible logic, first-order temporal logic, direct access control, markup-based representations, and goal modeling. Other related work is presented under the banner of legal programming [35]. An important work is Barth et al. [2]. Their work presents a temporal logic implementation of what they define as contextual privacy. Their approach uses linear temporal logic to define privacy model as agents, attributes, and messages. In addition, they model contextual integrity using roles, context, and traces.

Elgammal and Turetken [9][36] introduce the concept of compliance constraints, based on the notion of compliance patterns, leading to root-case analysis. They also demonstrate the use of Linear Temporal Logic with the model-checker SPIN to check compliance requirements. Undoubtedly, SPIN is a more powerful tool than Alloy to find behavioral counter-examples such as the one of Fig. 6, and in the future we are planning to explore the use of SPIN. However SPIN will be less powerful than Alloy in the discovery of purely logical inconsistencies.

A related approach, using Linear Temporal Logic with automata, was presented in [25].

Ghanavati, Amyot et al. [11] have dealt with the issue of compliance with requirements from the point of view of goal satisfaction, which is different and complementary with respect to the aims of this paper. Their approach can be used to identify completeness requirements, see Section III.

From a strategy perspective, we fulfill the goals proposed by Antòn, Bertino et al. [1]. We are also in line with the approach suggested by Cannon et al. [7], by which the first research priority is consolidating policy management and the second is automating compliance. From a modeling perspective, our high-level model compares to the REA model, however ours is more refined since it specifies more precisely resource control.

Our approach shares some principles with Hambrick et al. [15], which lays out the importance of the relationship between enterprise components, such as formal structure and behavioral process on one hand, and legal and normative compliance on the other hand.

VIII. Conclusions

We have outlined issues related to the development of legally compliant software in organizations, and we have proposed a development process consisting of a number of steps, or sub-processes, which are based on well-known software engineering concepts. We have adopted a definition of compliance based on the concept of logical consistency. We have then focused on the process of establishing compliance between organizational requirements and legal requirements. Compliance checking is done by translating both groups of requirements into GAL, our Governance Analyst Language. The translation from organizational and legal requirements into GAL is done by using GAM, a UML model of organizational concepts. GAL is then translated into the language Alloy and the Alloy tool is used to check for logical consistency, thus compliance, between the two groups of requirements. Although logic model checking is considered to be a computationally inefficient process, so far we haven't found a property that could not be checked in a matter of seconds (at most), and we refer to [18] for a systematic study of performance. See also [26] for discussion of progress in the efficiency of logical satisfiability algorithms.

With respect to other related methods existing in the literature, our method is characterized by the use of UML modeling, Alloy, and the specialized language GAL, which contains many useful constructs for compliance checking, and which can be directly mapped into the Alloy language. We have shown how our tool can find situations of non-compliance, and we have closed with considerations on how our process could be used for compliance auditing in the area of Personal Health Information Privacy.

The main strengths of our method are

- Systematic: the extraction method is based on the UML organizational model, an ontology of legal concepts which can be extended according to need.
- Expressive: since GAL can represent many types of legal requirements, and is extensible.
- Repeatable: the method is repeatable and has been tested for federal and provincial laws in conjunction with hospital and electronic health organizations.
- Adaptable: the model embedded in the method is generic and can be updated to reflect changes in the laws and regulations.

This will be further documented in forthcoming publications and work is underway to quantify the above metrics.

However our method is not always directly applicable. The necessary model may be quite different from our GAM, in which case a different UML model and different logic relationships will have to be developed. Or it may be difficult to express the requirements in a way that they can be logically checked (Section V). Further, our method needs to be complemented by a method addressing completeness (Section III).

ACKNOWLEDGMENT

This work was funded in part by grants of the Natural Sciences and Engineering Research Council of Canada. We thank the anonymous referees for useful suggestions. We also thank the open software community for their contributions to Alloy, Graphviz, KodKod, Python, and ArgoUML.

APPENDIX I - ALLOY FORMAL METHOD

The Alloy formal method [21] consists of a language and a logic analysis tool. The Alloy language is a structural modeling language based on first-order logic. It has four basic constructs: Signatures, Facts, Predicates, and Assertions. A signature introduces a basic type and a collection of relationships; this is suitable for representing hierarchical specifications. Facts are explicit constraints that must be satisfied in order for the Alloy tool to be able to generate a model instance. Predicates are expressions that are used to express constraints on generated objects; these are not pervasive as facts, and Alloy will generate an instance with examples even if a predicate is violated. Finally, assertions specify whether constraints are validated.

The Alloy-4[2] Analyzer is a logic analyzer and model finder, which accepts as input specifications written in the Alloy language. The tool can generate instances of invariants, simulate the execution of operations, and check user-specified properties of a model. Alloy-4 is modular and extensible. It has a core relational logic engine that incorporates up-to-date optimization techniques. The logic engine can be accessed in two ways: a compiler allows the model to be expressed in textual form, and a set of Java™ API methods allow the model to be constructed, queried, and analyzed dynamically. Additional interfaces can be easily written to integrate it into another analysis framework.

The Alloy tool produces not only the visual result that we have seen, but also a DOT language representation and an XML representation. These can be useful for other types of displays and validation.

The Alloy Analyzer 4 uses Kodkod, an efficient SAT-based analysis engine, for first order logic with relations, transitive closure, and partial instances. Optimizations are performed first at the Alloy level, and the reduced problem is then given to Kodkod.

Formal constraints can start from a minimal representation of the rules and be incrementally strengthened, adding constraints until the rules are completely specified; this method is called incremental validation. However, since we are validating legal compliance, the model is restricted to the enterprise structure, whereas the legal rules are asserted for validation. If the enterprise model fails, then the auditor should look for conflicting policies. Once the enterprise model is conflict free, assertion validation can be done incrementally. An auditor can aggregate multiple statements in joint predicates. The use of the analyzer in an interactive fashion assists the users in making the incremental changes and checking their validity.

Given that an enterprise structure is discrete at the time of validation, Alloy's bounded-exhaustive analysis implies that its results are valid with respect to the given instance only, that is, if the analyzer fails to validate an assertion of an Alloy predicate, this result is limited to the current enterprise instance. Alloy has the ability to create randomized instances; however, such a feature is not necessary in the analysis given that an enterprise instance is constant during the period of analysis.

One of the tenets at the basis of Alloy's method is what the author calls the "Small Scope Hypothesis" [21], which essentially states that most software design errors have small counterexamples, which can be generated automatically. We dare to extend this concept to the legal domain by hypothesizing that many violations of law can be detected by exploring small legal scenarios.

2 http://alloy.mit.edu/alloy4/

APPENDIX II – GAL – GOVERNANCE ANALYSIS LANGUAGE

The following list shows the currently implemented GAL statements according to their type. This set is open-ended and other statements can be added as needed.

In what follows, literal entities such as Process, Activity, Role, etc. are variables that must be instantiated with the names of actual processes, activities, and roles.

Note that the statements have been simplified for clarity. Among others, for traceability, each statement can include a literal comment indicating the normative statement from which it is obtained. This literal is output when a situation of non-compliance is detected.

1. Construction

ComposedOf (Object, Object)
Asserts that an object includes another object
ComposedOf (Process, Process)
Asserts that a process includes another process
Contains (Process, Activity)
Asserts that a process contains an activity
Includes (Department, Department)
Asserts that a department includes another department

2. Equivalence

EquActivity (Activity, Activity)
Asserts that two activities are equivalent (one can replace the other)
EquProcess (Process, Process)
Asserts that two processes are equivalent
EquRole (Role, Role)
Asserts that two roles are equivalent
EquDepartment (Department, Department)
Asserts that two departments are equivalent

3. Relational

Access (Activity, Object)
Asserts that an activity leads to accessing a particular object
Acts (Role, User)
Asserts that a user acts in a particular role
AssignedTo (Role, Activity)
Asserts that a user has been assigned a particular role
Assumes (User, Process)
Asserts that a user assumes a process
Delegate (User|Role, User|Role, Role|Activity|Process)
Asserts that a role or user delegatesan activity or process to another role or user
Next (Activity, Activity)

Asserts a sequence between activities

Separate ((Activity|Process, Activity|Process)[(Activity|Process, Activity|Process)]…)

Asserts that users or roles who have access to certain activities or processes cannot have access to certain other activities or processes

4. Assignments

CanAct (Allow|Deny, User, Role)

Allows or denies a user the possibility of acting in a role

CanAssignTo(Allow|Deny, Activity, Role)

Allows or denies a role the possibility of being assigned an activity

CanAssume (Allow|Deny, Process, User)

Allows or denies a user the possibility of being assigned a process

CanDelegate(Allow|Deny, User|Role, User|Role, Role|Activity|Process)

Allows or denies a user or role the possibility of delegating an activity or a process to another user or role

5. Checks

Activity- Predecessor (Activity, Activity)

Checks if the immediate predecessor of an activity is another activity

Activity-Process-Pred (Process, Activity, Activity)

The predecessor check is limited to a particular process

Activity-Process-Trace-Exist (Process, Activity, Activity)

Checks if within a process there is a trace starting with an activity and containing another activity later

Activity-Trace-Exist (Activity, Activity)

Checks if there is a trace starting with an activity and ending with another activity

Activity-Process-Trace-All (Process, Activity, Activity)

Checks if within a process all traces starting with an activity contain another activity later on

Activity-Trace-All (Activity, Activity)

Checks if all traces starting with an activity contain another activity later on

checkActs (Role, User)

Checks if a user acts in a particular role

checkAssignedTo (Role, Activity)

Checks if an activity is assigned to a role

checkAssumes (User, Process)

Checks if a user has access to a process

checkDelegate (User|Role, User|Role, Role|Activity|Process)

Checks if it is possible to delegate a role or activity or process from a user or role to another user or role

checkInstance (User | Process | Department|Role| Activity)

Checks that a specific class instance exists

Dept-Includes (Department|Role, Department|Role)

Checks if a Department or Role includes another Department or Role (directly or indirectly)

Process-Activity (Process, Activity)

Checks if a process includes the specified activity

Process-Includes (Process, Process)

Check if a process includes another (directly or indirectly)

References

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

[2] Barth A., Datta A., Mitchell J., Nissenbaum H. Privacy and Contextual Integrity: Framework and Applications. IEEE Symposium on Security and Privacy 2006: 184-198.

[3] Botha, R.A., Eloff, J.H.P. Separation of duties for access control enforcement in workflow environments. IBM Systems Journal, 40 (3), 2001, 666-682.

[4] Breaux, T. D., Vail, M. W., and Antòn, A. I. Towards Regulatory Compliance: Extracting Rights and Obligations to Align Requirements with Regulations. Proc. 14th IEEE international Requirements Engineering Conference (RE'06) IEEE Computer Society, 46-55.

[5] Breaux, T.D., Antón, A.I., Boucher, K., Dorfman, M. Legal requirements, compliance and practice: an industry case study in accessibility. IEEE 16th Int. Requirements Engr. Conf., 2008, 43-52.

[6] Brodie, C. A., Karat, C., 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.

[7] Cannon, J. C., Byers, M. Compliance deconstructed. Queue 4(7) 2006, 30-37.

[8] Cavoukian, A., Chanliau, M. Privacy and Security by Design. 2013. Retrieved May 2013 from http://www.privacybydesign.ca

[9] Elgammal, A., Turetken, O., van den Heuvel, W.J. Using Patterns for the Analysis and Resolution of Compliance Violations. Intern. J. of Cooperative Information Systems. 21 (1), 2012, 31–54

[10] Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R. Role-Based Access Control. Artech House, 2007.

[11] Ghanavati, S., Amyot, D., Peyton, L., Siena, A., Perini, A., Susi, A. Integrating business strategies with requirement models of legal compliance. Intern. Journ. Electronic Business 8(3), 2010, 260-280.

[12] Governatori, G., Shek., S. Regorous: A business process compliance checker. International Conference on Artificial Intelligence and Law (ICAIL) 2013.

[13] Governatori, G. Law, Logic and business processes. Third International Workshop on Requirements Engineering and Law, RELAW 2010, 1-10.

[14] Governatori, G., Sadiq, S. The Journey to Business Process Compliance. In Jorge Cardoso and Wil van der Aalst, editors, Handbook of Research on BPM, IGI Global, 2009.

[15] Hambrick, D.C., Werder, A.v., Zajac, E.J. New Directions in Corporate Governance Research. Organization Science, 19(3) 2008, 381-385,

[16] Hassan, W., Logrippo, L. Requirements and Compliance in Legal Systems: a Logic Approach. In Proc. IEEE 16th International Requirements Engineering Conference (RE'08): RELAW Workshop. Barcelona, Spain. Sep. 2008.

[17] Hassan, W., Logrippo, L. A Governance Requirements Extraction Model for Legal Compliance Validation. Intern. Workshop on Requirements Engineering and Law, RELAW 2009, 7-12.

[18] Hassan,W. Validating Legal Compliance - Governance Analysis Method. University of Ottawa, PhD thesis in Computer Science, 2009.

[19] He, Q., Otto, P., Antòn, A. I., Jones, L. Ensuring Compliance between Policies, Requirements and Software Design: A Case Study. Proc. 4th IEEE international Workshop on information Assurance, IWIA2006. IEEE Computer Society, 79-92.

[20] Hruby, P. Model-Driven Design Using Business Patterns. Springer 2006.

[21] Jackson D. Software Abstractions: Logic, Language, and Analysis. MIT Press. Cambridge, MA. 2006.

[22] Kajan, E., Stoimenov, L. Toward an ontology-driven architectural framework for B2B. Comm. ACM, 48 (12), 2005, 60-66.

[23] Kiyavitskaya, N., Zeni, N., Breaux, T. D., Antón, A. I., Cordy, J. R., Mich, L., Mylopoulos, J. Extracting rights and obligations from regulations: toward a tool-supported process. Proc. 22nd IEEE/ACM international Conference on Automated Software Engineering, ASE 2007, ACM, 429-432.

[24] Kudo, M., Araki, Y., Nomiyama, H., Saito, S., and Sohda, Y. Best practices and tools for personal information compliance management. IBM Syst. J. 46(2), 2007, 235-253.

[25] Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P. Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. Business Process Management. LNCS 6896. Springer 2011, 132-147

[26] Malik, S., Zhang, L. Boolean Satisfiability, from Theoretical Hardness to Practical Success. Comm ACM 52(8), 2009, 76-82.

[27] McCarthy, W.E. The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review, 57 (3), 1982, 554-578.

[28] Mili, H. Tremblay, G. Jaoude, G.B. Lefebvre, E., Elabed, L., El-Boussaidi, G. Business process modeling languages: Sorting through the alphabet soup. ACM Comput. Surv. 43(1): Art. 4 (2010).

[29] Morrison, P., Williams, L., Holmgreen, C., Massey, A. Proposing Regulatory-Driven Automated Test Suites for Electronic Health Record Systems. 5th Intern. Worksh. on Softw. Engineering in Health Care (2013).

[30] Ontario Laboratory Information System (2013). Retrieved May 1, 2013, from http://www.ehealthontario.on.ca/en/initiatives/view/olis

[31] Ontario Ministry of Health and Long-Term Care. (2013). *The Ontario Drug Benefit (ODB) Program.* Retrieved May 1, 2013, from http://www.health.gov.on.ca/en/public/programs/drugs/programs/odb/odb.aspx

[32] Otto P. N. and Antón A. I. Addressing Legal Requirements in Requirements Engineering, 15th IEEE International Requirements Engineering Conference, 2007, 5-14.

[33] Sadiq, S., Governatori, G. A Methodical Framework for Aligning Business Processes and Regulatory Compliance. In: Brocke, J., Rosemann, M. (eds.) Handbook of Business Process Management. Springer (2009)

[34] Sartor, G. Legal Reasoning, a cognitive Approach to the Law. Vol 5 in: Pattaro, E. A Treatise of Legal Philosophy and General Jurisprudence. Springer, 2005.

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

[36] Turetken, O., Elgammal, A., van den Heuvel, W.J., Papazoglou M.P. Capturing Compliance Requirements: A Pattern-Based Approach. IEEE Software 29(3), 2012, 28-36.

[37] Young Schmidt, J., Anton, A., Earp, J.B. Assessing identification of compliance requirements for privacy policies. Proc. of RELAW 2012, 52-61.

[38] Zowghi, D., Gervasi, V. On the Interplay betweem Consistency, Completeness, and Correctness in Requirement Evolution. Inform. and Softw. Technol. 46 (2004) 763–779.