A Model-Based Framework for Legal Policy Simulation and Legal Compliance Checking

Ghanem Soltana
(Supervisors: Mehrdad Sabetzadeh and Lionel Briand)
SnT Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
Email: ghanem.soltana@uni.lu

***Abstract*—Analyzing legal policies for many laws, such as taxes and social benefits, is a common way for governments to identify risks, e.g., risk of legal policies not achieving expected revenue. A typical analysis includes validation of policies and the verification of the systems implementing them. One efficient way to validate policies is simulation, e.g., by simulating whether a proposed law reform would realize target objectives. Once validated, policies are implemented into public administration procedures and eGovernment applications. Systems implementing legal policies also need to be analyzed and verified, e.g., through testing, to ensure that they are compliant with the underlying policies.**

Currently, legal policy analysis is conducted using a combination of spreadsheets and software code. Such strategy suffers mainly from being hard to use by legal experts due to the lack of adequate background. This is partly rooted in the fact that available techniques to formalize legal policies are based on complex logical expressions and code. The main goal of this research project, that this paper describes, is to narrow the aforementioned expertise gap by proposing convenient, systematic and automated techniques to support analysis of legal polices from their design to their implementation.

***Index Terms*—Legal Policies, Model-Based Simulation, Compliance Verification, Model-Driven Code and Data Generation**

I. Problem Statement and research methodology

A. Problem Statement

[Context] Laws are extended, reviewed and revised on a regular basis to ensure that they meet fiscal, monetary, and social expectations. Thoroughly assessing risks related to designing and implementing legal policies for many laws, e.g., taxation and social benefits, is a major concern for governments. Computer-assisted analyzers have been increasingly used in recent years as tools for: (1) validating legal policies [1]–[3], e.g., by checking that policies are aligned with the provisional budget of the government, and (2) verifying systems compliance to the underlying laws [4], e.g., through testing.

[Scope] In a nuanced field such as law, providing adequate support for automated analysis depends mainly on having a focused scope. This project targets *prescriptive* laws, i.e., highly regulated laws, such as taxation, that are defined by a set of legal policies– a legal policy being the textual definition of step-by-step guidance in the form of procedures to apply for compliance [5]. Nevertheless, we believe that this works can be tailored to other laws, e.g., declarative laws that are founded on permissions, obligations, and prohibitions. In this paper, we take legal policies to mean "procedural" legal policies.

[Motivation] Legal policies need to be validated to check that they actually lead to their expected results and nothing else. In practice, validation is done by conducting several simulation scenarios over the policies. An example of a simulation scenario is to quantify the effect of modifying the policies on variables of interest such as the revenue, while assessing that no undesirable side effects can occur, e.g., provoking a high increase in tax dues for low-earning families. Quantifying the outcome of a set of policies in a precise manner is not a trivial task due to: (1) the high number of policies to consider, (2) the complexity of the procedures specified by the policies, and (3) the inter-relationships between policies, e.g., some policies being mutually exclusive or complementary. Once validated, policies are implemented into administrative procedures and software systems. At this stage, system verification is needed to ensure that systems being built are indeed compliant to the law. To be effective, compliance verification has to consider the subtleties of the targeted legal jurisdiction, e.g., the procedural aspect of prescriptive laws.

[Policy formalization challenge] A key prerequisite for any kind of automated analysis is to formalize legal policies in a syntactically and semantically well-defined form. To date, policy formalization are mainly based on complex logical expressions and code [6], [7]. An important factor to consider here is that these formalizations are simultaneously used by several stakeholders with different and non-overlapping backgrounds, e.g., legal experts and system analysts. An open question here then is how to interpret the underlying law so that the resulting formalizations are, on the one hand, intuitive enough to all types of users, and on the other hand detailed enough to be analyzable through automated means.

In addition, when legal text undergoes modification, it is difficult to trace the impact of changes on the developed artifacts if no support is provided within the policy formalizations. The inability to automatically identify the impact of changes would not only complicate the testing of the system once the changes made, but can also introduce further inconsistencies between the legal texts, legal requirements, and the system implementation.

[Data generation for V&V of legal policies challenge] Outputs of both targeted analysis in this project, i.e., simulation and testing, are obtained by processing some data as input,

e.g., simulation data. However, one might face the situation where no complete and usable input data is available. This is particularly true when new policies are being added to the current legislation (no real historical data). Test case generation has long been studied to generate fault-revealing inputs for testing [8]. On the other hand, few works address simulation data generation. A challenge is to devise a data generator that: (1) is scalable, as many simulation cases are needed, and (2) produce representative samples of the underlying population, to ensure the reliability and precision of the simulation results.

B. Research methodology

The research methodology of this project follows an industry based collaboration paradigm for research innovation [9], [10]. One particularity of such paradigm is that most research activities are realized in close collaboration with industrial partners. Adopting such research methodology promotes tackling tangible problems in their real settings. Furthermore, proposed solutions can be empirically validated thorough industrial case studies. In our context, this project was motivated by an initiative of the government of Luxembourg to enhance its current practices in analyzing legal policies. Our collaborating institutes are: (1) *Centre des technologies de l'information de l'Etat*, Luxembourg's government computing center, and (2) the Inland Revenue Office of Luxembourg.

II. Related Work

In this section, we present related work on the topics of legal policies modeling, legal policy simulation, and legal compliance verification.

Legal policies modeling. Legal policy formalizations have long been studied in the artificial intelligence community [6], [11]–[14]. Notable among these is the work by Rissland and Skal [13] that combines different knowledge representation techniques to capture then reproduce the reasoning of legal experts in tax court cases. Melz and Valente [12] build a domain-specific ontology for the US internal revenue code and discuss applications for tax assistance systems using their ontology. These formalizations were primarily developed for performing expert search and question-answer reasoning without considering the software engineering aspects of systems implementing them. In particular, missing from the above-cited work is an operational view of the legal policies, the verification of software compliance to the law, and enabling automated analysis such as simulation. In contrast, our proposed research has a strong software engineering orientation. The formalization that we propose captures workflows of legal policies while providing executable semantics that will later enable simulation and testing.

Little work exists that addresses the development of large-scale legal applications from a software engineering perspective. For example, Breaux et al. [15] proposed a rule-based framework that enables executives, business managers, and developers to distribute legal obligations. van Engers et al. [7] applied a methodology based on UML for modeling the Dutch tax legislation to facilitate the communication between developers and legal experts. Nevertheless, the aforementioned work also share the same limitations observed for work from the artificial intelligence community. Unlike the modeling approach that we proposed in [16], these strands of work lack operationalization means to support the automated analysis targeted in our context, i.e., simulation and testing. Furthermore, our modeling methodology provides guidance on how to efficiently build policy formalization from scratch. Our modeling notation also manages traceability between the legal text, the detailed legislature that elaborate and interpret the law, and the underlying software system. This enables us, for instance, to locate the system components that are impacted by a change in the law.

Legal policy simulation. Several legal policy simulation tools were proposed in the area of applied economics [1]–[3]. Nevertheless, these tools do not adequately address the expertise gap between legal experts and system analysts. Policies are usually implemented using software code, logical expressions or equations. For example, EUROMOD [2] uses a combination of spreadsheets and C++ to operationalize legal policies. Thus, the implemented interpretation of the law cannot be understood by legal experts. Our framework addresses this gap by providing a more abstracted way to specify legal policies, so that the resulting specifications would be palatable to legal experts with a reasonable amount of training. The second limitation of the above-mentioned tools concerns the simulation data. These tools assume that complete and precise simulation data is given as input. This is a strong requirement that is not always feasible in practice. For instance, when new policies are being introduced, no real historical data may be available. In contrast, our simulation framework comes equipped with a built-in data generator that produces, if needed, hypothetical but realistic simulation data, based on historical aggregate distributions and/or expert estimates.

Legal compliance verification. Several methods were proposed to ensure *business processes* compliance with legal policies in domains such as healthcare [17], [18] and finance [4]. For instance, Ghanavati et al. [17] tracks legal compliance by a framework based on combining goal, use case and privacy goal models. Hassan and Logrippo [4] provide a semi-automatic method for checking compliance of enterprise requirements with respect to legal requirements using formal logic.

Few strands address compliance for *software systems*. Existing work on regulatory compliance for software is geared towards corporate governance, privacy, and security [19]–[21]. In contrast, our research targets specific aspects of compliance that are proper to procedural legal policies that are the scope of this project (see Section I). A major concern in *prescriptive* laws is ensuring that the software systems are performing all calculations, e.g., tax deductions, only for eligible individuals and as specified by the underlying provisions.

III. Proposed solution

This section provides an overview of the approach we propose to enable automated analysis of legal policies while addressing the aforementioned gaps (see Sections I and II).

The proposed solution, shown in Fig. 1, consists on a framework based on model driven engineering techniques [22], where models are the main development artifacts. We opted for such solution because we believe that models are a more structured, precise, and yet intuitive way to formalize policies in comparison with plain text, logical expressions and code. Below, we briefly describe the steps of the framework.

In Step one, *Model legal policies*, models of policies to analyze are built by analysts with help from legal experts. This step has two outputs: (1) a *domain model*, and (2) policy models that capture the procedures envisaged by the law. These models will be further detailed in Section IV.

Step two, *Generate code*, is a transformation that derives executable code from policy models. This code constitutes the core of the analyzer engine of Step five. The appropriate language to use and the analyzer's components that need to be completed are determined according to the type of the analysis. For example, Matlab code is derived for simulation and OCL-based oracle functions are derived for testing.

Both steps, three and four, create an instance model from the domain model. This instance model encapsulates data that will be processed by Step five. Step three, *Use existing data*, creates an instance model based on exiting data, e.g., historical data. If such data is not available, then Step four, Generate data, creates an instance model based on one of the predefined generation strategies. The appropriate strategy is selected according to the type of the analysis to perform. For simulation, a probabilistic generator is used to create data that is statistically aligned to the real population. Whereas for testing, search based techniques are used to meet other objectives, such as maximizing code coverage.

Step five, *Run analysis*, uses an engine to perform automated analysis over legal policies. First, the engine executes the code generated by Step two over the instance model created in the previous step (step three or four). Then, outputs/traces of policy models execution are gathered and analyzed for performing a particular simulation scenario or for compliance verification. For example, one simulation scenario is to quantify the impact of introducing new policies on variables of interest, e.g., taxpayers' net incomes. As shown in Fig. 1, an extra input is used by this step when performing software verification. This input is the outputs/traces of executing the system under test over the same data used for simulation. The system is deemed compliant if and only if outputs/traces from both executions (system under test and policy models) are identical, once mapped to the same level of granularity and precision. Finally, results are plotted and displayed to the user.

IV. Preliminary Work

So far, three main milestones have been reached: (1) a modeling approach for formalizing legal policies, (2) a transformation that enables semantic execution of policy models created using our modeling methodology, and (3) a probabilistic data generator that creates representative simulation samples with respect to a given population.

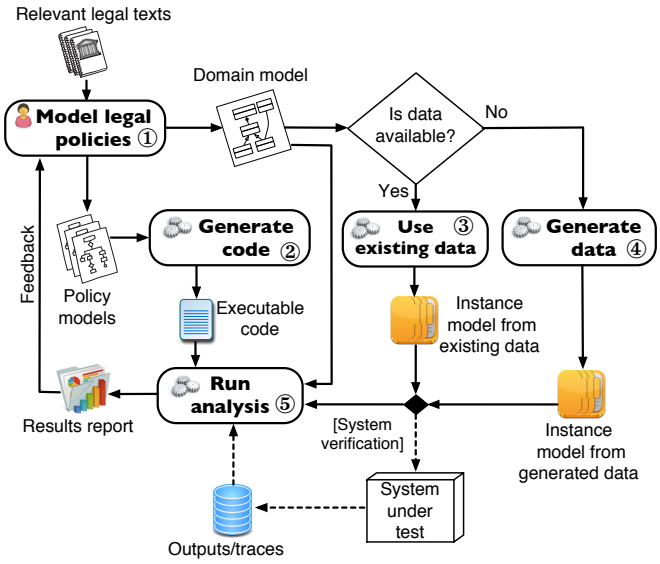

Fig. 1. Model-based Framework for Analyzing Legal Policies

Modeling approach We developed a modeling methodology to formalize legal policies [16]. As introduced in Section III, the application of our modeling methodology yields two types of models: (1) a UML class diagram representing the domain model, and (2) policy models that capture the procedures defined within the law. We use standard practices for domain modeling to build the class diagram. The class diagram in our methodology is an instrument for defining the inputs that policy models use. As for policy models, they are UML activity diagrams extended with additional semantics captured by a UML profile [23]. We opted to use activity diagrams partly because they are well-documented and widely used, but mainly because government agents, including legal experts, are familiar with simple conceptual models and business process models from earlier exposure and training.

An example of a policy model that calculates the tax deduction granted to a taxpayer for disability is depicted in Fig. 2. Gray boxes, at the left side of the model, represent input parameters involved in the calculation of this particular deduction, e.g., *is_disabled*. The core of this policy model encompasses three alternative deduction calculations, denoted by the actions with the *«calculate»* stereotype. Each calculation is defined by a corresponding formula (*«formula»*). Based on the taxpayer's eligibility, assessed through decisions (having *«decision»* stereotype), the appropriate calculation is applied. For instance, if a given taxpayer is not disabled, this policy yields a value of zero; otherwise, another alternative is selected based on disability type, e.g., *Standard deduction*. In addition to the above described elements, the policy model contains an operation, marked by the *«assert»* stereotype, that specifies properties to test during a compliance checking analysis. For instance, the assert operation in Fig. 2 verifies that the outcome produced by system implementing the deduction for disability policy matches the outcome envisaged by the policy model.

Policy models operationalization We developed a rule-based

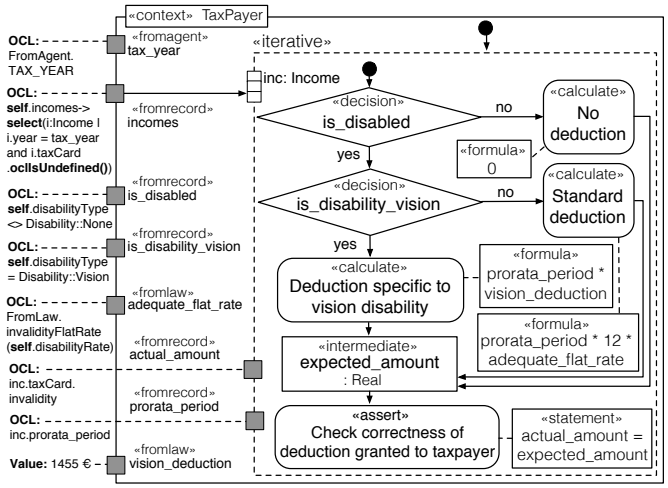

Fig. 2. Policy Model for Calculating Invalidity Tax Deduction (Simplified)

transformation algorithm [16], [24]. This transformation gives policy model execution semantic to enable their automated analysis. This model-to-text transformation was built on top of Papyrus (eclipse.org/papyrus/) and was encoded using Acceleo (eclipse.org/acceleo/). Further tool support details and examples of generated code can be found in [24]. At this stage, the transformer supports three target languages, i.e., Java, OCL, and Matlab. As discussed in Section III, the appropriate language is used based on the type of desired analysis, e.g., OCL invariants for testing.

Probabilistic simulation data generation To be able to faithfully capture the properties of the population that is subject to simulation, we proposed in [25] a UML profile (not to be confused with the profile for policy models). This profile extends class diagrams with a variety of probabilistic notions, including probabilistic attributes, multiplicities and specializations, as well as conditional probabilities. Using this profile one could indicate, for instance, that values of a given attribute should be drawn from a normal distribution. Such statistical information is available from census data or can be elicited from domain experts. We have also developed an automated data generator that uses the above described profile to generate realistic simulation data samples. Further information, such as the generation strategy we developed to satisfy the multiplicity constraints, can be found in [25].

V. Expected contributions

A modeling methodology for expressing legal policies: We propose a modeling methodology that: (1) raises the level of abstraction for specifying legal policies, (2) provides a clear interpretation of the law, and (3) maintains traceability between models, legal texts and software systems. We believe that a higher level of abstraction mitigates the expertise gap that legal experts are facing (discussed in Sections I and II).

A model-based probabilistic data generator: We introduce a UML-class diagram instantiator that is based on a new UML profile for capturing the probabilistic characteristics of a given population. As described in Section III, it is important to provide several generation strategies that maximize the fitness of the generated data with regards to the analysis to perform. For instance, one objective to meet when simulating policies is to generate data that is likely to be sampled from the real population. However, when performing testing, an objective should be to maximize the number of detected faults.

A framework for analyzing legal policies: Our ultimate expected contribution is to propose a framework that supports various forms of policy analysis. Our framework should overcome the limitations discussed in Sections I and II while enabling: (1) the estimation of variables of interest, e.g., expected revenue from a given set of policies (2) change impact analysis activities, to determine the impact that a modification in the law would have on development artifacts or on variables of interest, (3) consistency checking, to identify undesirable situations that would violate operating principles, and (4) legal compliance checking, to detect defects in software systems implementing the policies. We believe that the executable semantics of our policy models combined with search-based techniques would provide the adequate infrastructure to support the aforementioned analysis.

VI. Plan for evaluation and validation

To assess the soundness, effectiveness and efficiency of the proposed solutions in addressing the problem of Section I, we rely mainly on industrial case studies.

As mentioned in Section I-B, this project places a lot of emphasis on case studies conducted in collaboration with industry. The validation has started at an early stage of the project as some milestones were already reached. For instance, we conducted a case study over the Luxembourgish personal income tax law to a verify that: (1) the level of effort required to build policy models is reasonable, and (2) policy models built using our methodology are less complex than policies specified using OCL– evaluated through several structural complexity factors from [26]. Results of this case study were promising and suggest that our solution tackles to a large extent the policy formalization challenge described in Section I. We further plan to conduct a usability study to assess how convenient to use our developed tools are, e.g., the probabilistic data generator and the policy models simulator.

VII. Current status

An overview of the planned activities alongside expected completion dates is given in Fig. 3. Outcomes from activities one and two were presented in Section I and Section II, respectively. Status of activities three to five is presented in Section IV. Below, we outline our current and future activities.

We have taken a number of concrete steps towards generating hypothetical but yet realistic simulation data (Activity 5). We developed a probabilistic data generator aimed at producing a large instance model from a given UML class diagram while respecting the probabilistic characteristics of the underlying population [25]. The key enablers of this generator are: (1) a UML profile that captures the probabilities

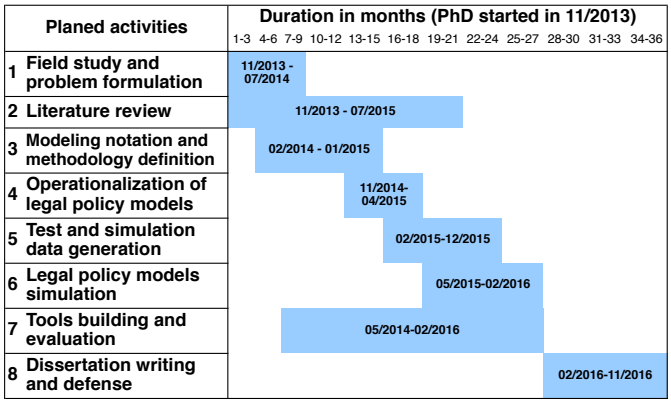

Fig. 3. Planned Activities and their Timeline for Completion

to respect (described in Section IV), and (2) an efficient generation strategy. For example, our generation strategy includes a slicing mechanism that improves scalability by narrowing the generation to only what is relevant for the policy models to analyze. However, our current generation strategy does not consider additional constraints, e.g., OCL constraints attached to the UML class diagram elements, that need to be satisfied for consistency. To overcome this limitation, we plan to investigate how our generator can be enhanced with constraint solving capabilities, e.g., by integrating an OCL solver [27].

Currently, our policy simulator only supports basic simulation scenarios such as *result differencing*, i.e., when an original and a modified set of policies are executed over the same simulation data; then the simulation results are compared to quantify the impact of the proposed changes. In the future, we would like to investigate search-based techniques in order to incorporate more advanced simulation scenarios such as the identification of undesirable situations, e.g., the identification of exploitable loop-holes or operating principles that may be violated by legal means.

Acknowledgment. I thank my PhD supervisors, Dr. Mehrdad Sabetzadeh and Prof. Dr. Lionel Briand for their guidance and support. I am grateful to members of Luxembourg Inland Revenue Office, in particular Thierry Prommenschenkel, for sharing their legal knowledge with me. I acknowledge financial support from *Centre des technologies de l'information de l'Etat* and *Fonds National de la Recherche* under grants FNR/P10/03 and FNR9242479.

REFERENCES

[1] L. Canova, L. Piccoli, and A. Spadaro, "SYSIFF 2006: A microsimulation model for the french tax system," MicroSimula - Paris School of Economics, Technical report, 2009.

[2] F. Figari, A. Paulus, and H. Sutherland, "Microsimulation and policy analysis," in *Handbook of Income Distribution*. Elsevier, 2015, vol. 2, pp. 2141–2221.

[3] S. Hohls, "How to support (political) decisions?" in *Electronic Government*. Springer, 2013, pp. 111–122.

[4] W. Hassan and L. Logrippo, "Requirements and compliance in legal systems: a logic approach," in *Proceedings of 1st International Workshop on RE and Law (RELAW'08)*, 2008, pp. 40–44.

[5] K. Petersen, "The regulation of assisted reproductive technology: a comparative study of permissive and prescriptive laws and policies," *Journal of law and medicine*, vol. 9, no. 4, pp. 483–497, 2002.

[6] T. Bench-Capon et al., "A history of AI and Law in 50 papers: 25 years of the International Conference on AI and Law," *Artificial Intelligence and Law*, vol. 20, no. 3, pp. 215–319, 2012.

[7] T. van Engers, R. Gerrits, M. Boekenoogen, E. Glassée, and P. Kordelaar, "POWER: using UML/OCL for modeling legislation - an application report," in *Proceedings of 8th International Conference on Artificial Intelligence and Law (ICAIL'08)*, 2001, pp. 157–167.

[8] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, 2010.

[9] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson, "A model for technology transfer in practice," *IEEE Software*, vol. 23, no. 6, pp. 88–95, 2006.

[10] L. C. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue, "Research-based innovation: A tale of three projects in model-driven engineering," in *Proceedings of the 15th International Conference on Model-Driven Engineering Languages and Systems (MODELS'12)*, 2012, pp. 793–809.

[11] J. Stelmach and B. Brożek, *Methods of legal reasoning*. Springer, 2006.

[12] E. Melz and A. Valente, "Modeling the tax code," in *Proceedings of 2nd International Workshop on Regulatory Ontologies (WORM'04)*, 2004, pp. 652–661.

[13] E. Rissland and D. Skalak, "CABARET: Rule interpretation in a hybrid architecture," *International Journal of Man-Machine Studies*, vol. 34, no. 6, pp. 839–887, 1991.

[14] L. Bibel, "AI and the conquest of complexity in law," *Artificial Intelligence and Law*, vol. 12, no. 3, pp. 159–180, 2004.

[15] T. Breaux, "Exercising due diligence in legal requirements acquisition: A tool-supported, frame-based approach," in *Proceedings of 17th IEEE International Requirements Engineering Conference (RE'09)*, 2009, pp. 225–230.

[16] G. Soltana, E. Fourneret, M. Adedjouma, M. Sabetzadeh, and L. C. Briand, "Using UML for modeling procedural legal rules: Approach and a study of luxembourg's tax law," in *Proceedings of the 17th International Conference on Model-Driven Engineering Languages and Systems (MODELS'14)*, 2014, pp. 450–466.

[17] S. Ghanavati, D. Amyot, and L. Peyton, "Towards a framework for tracking legal compliance in healthcare," in *Proceedings of 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, 2007, pp. 218–232.

[18] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions," in *Proceedings of 7th Workshop on Business Process Management (BPM'06)*, 2006, pp. 5–14.

[19] T. Breaux and A. Anton, "Analyzing regulatory rules for privacy and security requirements," *IEEE Transactions on Software Engineering*, vol. 34, no. 1, pp. 5–20, 2008.

[20] S. Islam, H. Mouratidis, and J. Jürjens, "A framework to support alignment of secure software engineering with legal regulations," *Software and System Modeling*, vol. 10, no. 3, pp. 369–394, 2011.

[21] S. Ingolfo, A. Siena, J. Mylopoulos, A. Susi, and A. Perini, "Arguing regulatory compliance of software requirements," *Data & Knowledge Engineering*, vol. 87, pp. 279–296, 2013.

[22] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering (FOSE'07)*, 2007, pp. 37–54.

[23] Object Management Group, "UML 2.2 superstructure specification," 2009.

[24] G. Soltana, E. Fourneret, M. Adedjouma, M. Sabetzadeh, and L. Briand, "Using UML for modeling legal rules: Approach and a study of luxembourg's tax law," Interdisciplinary Centre for Security, Reliability and Trust (SnT), Technical report no. TR-SnT-2014-3, 2014, http://people.svv.lu/soltana/Models14.pdf.

[25] G. Soltana, N. Sannier, M. Sabetzadeh, and L. C. Briand, "A model-based framework for probabilistic simulation of legal policies," in *Proceedings of the 18th International Conference on Model-Driven Engineering Languages and Systems (MODELS'15)*, 2015, (to appear).

[26] L. Reynoso, M. Genero, and M. Piattini, "Towards a metric suite for OCL expressions expressed within UML/OCL models," *Journal of Computer Science and Technology*, vol. 4, no. 1, pp. 38–44, 2004.

[27] S. Ali, M. Zohaib Iqbal, A. Arcuri, and L. Briand, "Generating test data from OCL constraints with search techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1376–1402, 2013.