MODELING LEGISLATION BY USING UML STATE MACHINE DIAGRAMS

Vjeran Strahonja, PhD
Faculty of Organization and Informatics
University of Zagreb, Croatia
email: vjeran.strahonja@foi.hr

Abstract

The basic idea of modeling law, as presented in this paper, is capturing domain knowledge of legislation and specifying it in a generic way by using commonly agreed and understandable modeling concepts of the Unified Modeling Language (UML). State machine diagrams provide a graphical notation for describing the dynamic (time-dependent) behavior of a system. Business oriented behavioral models of legislation enable to understand the system better, support the detection of anomalies and help to improve the quality of legislation by validation and verification. Different types of anomalies in legislation are classified and described in the article. More specific, this paper presents a static analysis approach to the checking of correctness and consistency of the UML state machine diagrams specifications of legislation. The presented framework includes semantic and syntactic anomalies.

Other motivation for modeling legislation is a desire to build court case management systems. The prerequisite of building such models is the transformation of legislation and regulations into system models that focus on different aspects of the computer system, such as programs that automates the business process and business rules, database, user interface, system procedures etc.

Based on empirical research, assessment of proposed method is made.

***Keywords**: Modeling legislation; behavioral model; state machine diagram; static and dynamic analysis; court case management system*

1. Introduction – Modeling of Legislation

Traditionally, modeling is an essential part of business analysis and reengineering, as well as software development. Specific modeling methods and techniques are enabling specification, visualization, and documentation of business and system models. Some of advantages of modeling may be used in domain of legislation. The term "legislation" in this paper refers to the set of laws, statutes and other legal acts that cover a particular subject of law or practice.

The aim of this paper is to discuss modeling of procedural aspects of legislation. The separation of substantive and procedural aspects of legislation is well known. The procedural regulation defines the "court procedure" in terms of the process that the case will go through. From the point of view of parties and judge, procedural regulation sets the rules for proceedings and enforcement of substantive law. Application of the procedural regulation is not focused on the quality of substantial decisions, but on the quality of the process (workflow, duration, delays, number of hearings etc.). In contrast to procedural, the substantive regulation (i.e. law or its part) deals with the "substance" of the matter. It defines crimes and punishments, how the facts in some type of the case or legal procedure will be handled, how the crime will be charged, or the dispute will be resolved.

The basic idea of modeling law, as presented in this paper, is capturing domain knowledge of procedural legislation and specifying it in a generic way by using commonly accepted and understandable modeling concepts of the Unified Modeling Language (UML) [1]. Currently, UML is de facto standard for expressing object-oriented analysis, design modeling and documenting object-oriented and component-based system architectures. Although the strengths of UML are at software development, it is commonly used for representing business domain.

In the context of modeling, a business domain has two distinct aspects: the structural or static aspect (functionality, business data etc.), and the behavioral or dynamic part (states, transitions, activities, sequences etc.). From the point of view of this paper, emphasis is on the behavioral features of the system, e.g. the ways a system behaves in response to certain events or actions. Dynamic models of legislation, in a form of UML state machine diagrams, provide a graphical notation for describing the dynamic (time-dependent) behavior of a legal system and improve understanding of a legal domain. They also provide a framework for validation and verification of legal regulation and its model. Sometimes the motivation for modeling legislation is a desire to build court case management systems.

2. Behavioral Modeling and Analysis of Legislation

Although the static aspects of legislation are also very important, this paper focuses on behavioral modeling of legislation. More specific, this paper presents an analysis approach based on the UML state machine diagrams of legislation. Ideally, we would like to have such model of legislation, even formal specifications, to check correctness and consistency of legal regulation and its model.

State machines and other UML models offer over all:

- describing system behavior in an intuitive way by using visual modeling
- readability and understandability by other human readers, and lower level of required expertise, as compared with formal specifications
- lower ambiguity, as compared with natural languages.

State machine diagrams show the possible states of the object and the transitions that cause a change in state. In object oriented approach, state modeling and state machines are usually related to classes and class diagrams, and describes the allowable states a class or element may be in and the transitions that allow the element to move there. State modeling means discretization of continuous phenomena.

Roots of the UML 1.x statecharts formalism [1] are traditional state-transition diagrams, which are developed as a part of the structured system analysis. These are extended with some new concepts taken from David Harel, such as the concurrent states and nesting of states. UML 2.0 introduces state machines with some new constructs [2]. Basic concepts of state machine diagrams are: state (simple, composite, submachine), transition, event, pseudo-state (initial, final, shallow history, deep history) and guard condition. Some of them made some confusion and created discussions, such as entry and exit operations, or distinction between activities and actions.

Based on current research, we suggest the iterative process of modeling legislation that consists of four steps (Figure 1.):

1) Basic analysis of selected legislation (classification, conceptualization and refactoring)
2) Transformation to UML constructs and representation in a form of diagrams
3) Validation and verification (detection of anomalies based on static and dynamic analysis)
4) Improvement of model and legal sources.

Classification of statements must consider some classification patterns (procedural / substantial, terms and definitions, case management, court activity, decision making, conducting the procedure, document management, human communication …).

Conceptualization comprises the identification of structural and behavioral constructs of selected legal act.

Refactoring is the process of rewriting of legal source to improve its readability and structure from the point of view of further modeling technique, with the explicit purpose of keeping the meaning and behavior of the source. As presented on the Figure 1., the applied refactoring form was simple table, with columns: who/actor, facts and rules (time limit, initial state, event, action, final state), with reference on the legal act.

As UML and its modeling techniques move from academic

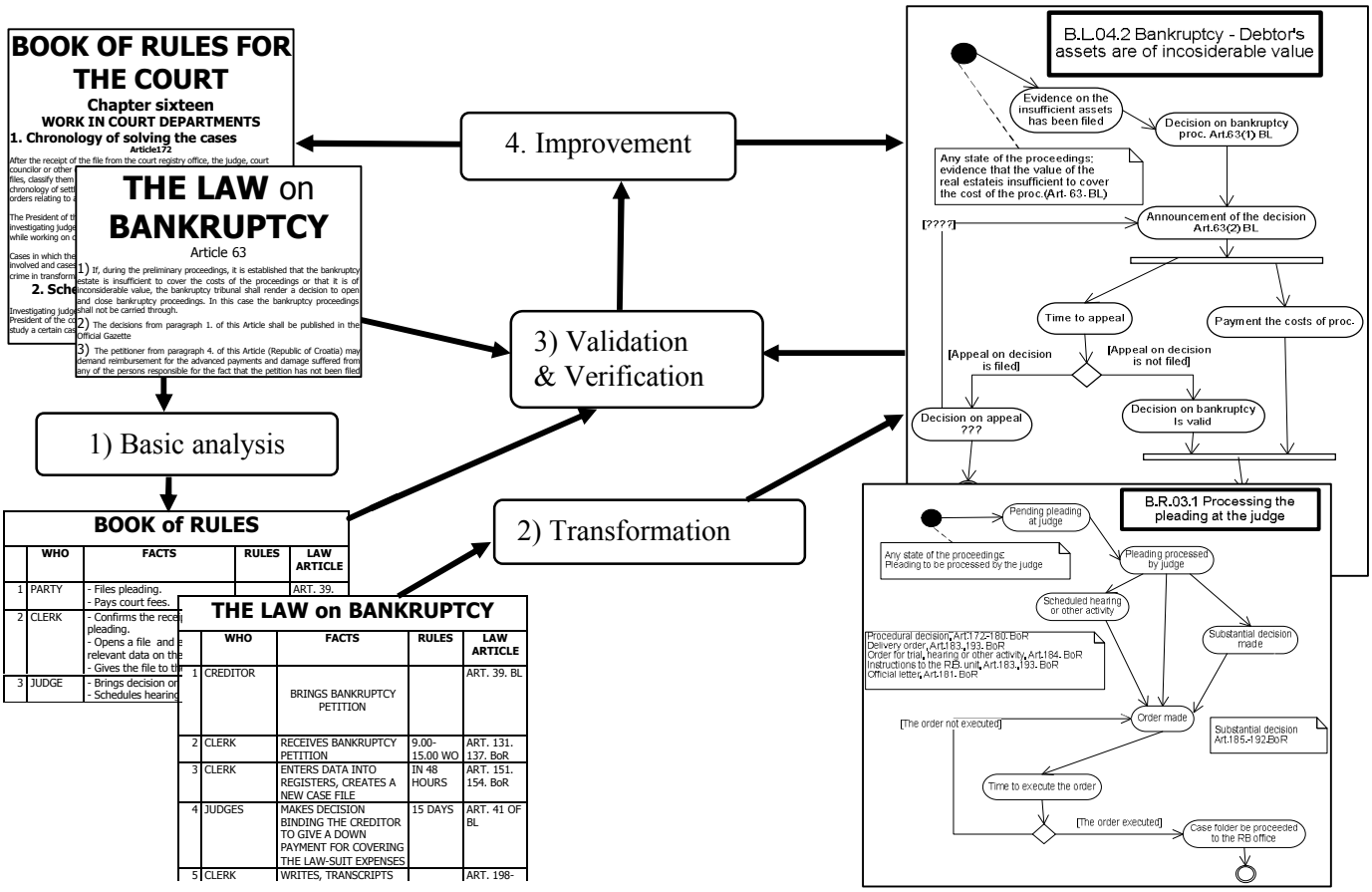

Figure 1. Iterative process of modeling legislation

institutions into commercial software development and domain modeling, they have to fulfill stronger requirements concerning correctness and consistency. Therefore, verification and validation are inherent activities of modeling.

There are two basic complementary analysis techniques for modeling legislation: static and dynamic (Table 1.).

The static analysis lies on the concept of class and gives a behavioral model that is valid for all possible case proceedings. In our example, the Procedural Manual (the Book of Rules) defines a general court procedure, valid for all courts and all types of case. The Bankruptcy Law, the Law on Civil Proceedings, the Execution Law etc. define specific court procedure valid for all courts and for some specific type of case. Static analysis as a process comprises modeling and evaluation of a model or system, based on its form, structure, content, or documentation. The idea is to understand how the system works and establish certain correctness criteria. This is a conservative technique, where we analyze the implementation to prove which states and transitions are illegal. The static analysis checks those criteria that are not related with the global state space (an upper bound).

The dynamic analysis uses a specialization, i.e. an implementation sub model of the static analysis model that is valid for one particular case proceeding. Conceptually, it lies on the object as an implementation of the class. In the dynamic analysis, we observe instances of states and transitions that are a subset (a lower bound) of the ideal, complete model. We use these specializations as proof of existence. For example, reachability analysis is detection of unreachable states, undesired global states or illegal sequence of actions. Unfortunately, the examination of a global state space often results in a state space explosion [6].

Table 1. Comparison of static and dynamic analysis.

Static Analysis	Dynamic Analysis
Represents all possible states and transitions in all possible case proceedings	Represents one particular case proceeding
Superset of ideal model, generalization, upper bound	Subset of ideal model, specialization, lower bound
Conservative analysis detects illegal states and transitions	Proof of existence and reachability analysis detects legal states and transitions
Assumes that an exception will be produced on an illegal input	Global state of space results in state space explosion

3. Anomalies in Legislation

The most desirable "technical" properties of legislation, but not all, are completeness, consistency and logical/semantic contradiction. Absence of this and other desirable properties are anomalies in legislation. It's generally accepted that anomalies in legislation impact on law implementation and enforcement. Many of the theories, methods and formal approaches in a field of business modeling and modeling of legislation originated in linguistics and in knowledge based systems. Some general aspects are similar in both linguistics and modeling theory, like syntax, semantics and pragmatics.

As an analogue of linguistics, we can also define syntactic, semantic and pragmatic anomalies of models. In the case of models, modeling syntax comprises the set of allowed modeling concepts, reserved words and their parameters and the correct way in which modeling concepts are used. Syntactic anomalies are caused by a violation of the structural (grammatical) rules for a modeling technique.

In a modeling theory semantics deals with the meaning systems of modeling language and concepts and their mapping to the real world. Semantic anomalies deal with violation of meaning and sense, for example conflicting truth conditions, name conflicts, dangling references etc. In a modeling theory, semantic anomalies are mostly result of inconsistent or inadequate use of modeling concepts.

Pragmatics is the study of information structure and the use of language in communicative context. Pragmatics is concerned with bridging the gap between a theory and its implementation in some context.

Some other theories that we need for validation and verification of legislation and its models are traditionally addressed in knowledge based systems. Although research of anomalies in legislation is still an attractive field for research, we use some common issues like consistency, completeness and logical/semantic contradiction, that are considered more then ten years ago [5].

Incompleteness means that there is at least one improvable schema (sentence, statement) that could be added as an axiom schema without creating simple inconsistency. Incompleteness issues are: dead-end rules, missing rules, unreachable rules, dangling references, unreferenced attribute values and other unintentional non-determinism.

Inconsistency of the specification implies that there are conflicting statements. Discrepancy is simply the difference between conflicting statements, definitions or rules of the same fact or situation. Some of the appearances of inconsistency are redundancy, unnecessary IF conditions logical contradiction, subsumed rules, circular rule, name conflicts (synonyms, homonyms), inconsistent generalization/specialization and other logical/semantic contradictions.

Anomalies in legislation may be caused by different semantic, syntactic or pragmatic reasons. For example, homonyms are pure semantic anomalies (if they are not desired), but an unintentional non-determinism can occur in the model as a semantic anomalism of the legal pattern, or a syntactic failure.

Anomalism can occur on two different levels, on the level of a model, or on the level of legislation itself. For example, some missing rule can disappear during a modeling process, but can also be omitted in regulation during the legislative procedure.

4. The Approach to the Validation and Verification of Anomalies

Since manual checking of anomalies in legislation is error-prone and time-consuming, currently the development of computer supported and automated methods for validation and verification of legislation attract researchers and practitioners from all around the world. All these validation and verification methods lie on decreasing complexity of legislation, by using some methods of modeling laws.

In general, the detection of anomalies in legislation comprises validation and verification. Although the objective is the same, the approaches to validation and verification differ in their orientation. In the field of software engineering, validation answers the question: "Are we building the right product?", and verification: "Are we building the product right?" [3]. Validation means testing a model or specification against the users' requirements and expectations, and verification means testing against the design specification, methodology, use of modeling concepts, rules of design etc.

Validation is the process of checking if statements of some legal act are true and meet common regulatory requirements and statutory compliance. Validation is based on human expert opinion. The idea of validation, as applied in this paper, is to transform legal structure and procedure into a visualization model, to enable experts to validate their scenarios. Visualization of legal structure and procedures can help the expert to make decision whether some potential anomaly (redundancy, synonym, circular definition etc.) really is an anomaly in legislation or not.

Validation should not be confused with verification. Verification is the act of proving or disproving the correctness of a legal act with respect to a certain specification or property. Verification is the process of reviewing, auditing, inspecting, testing, checking, or otherwise establishing and documenting whether some specification or model conforms to predetermined requirement. In contrast of validation, that is a human-directed proof, verification can be automated to some extend, as described in relevant papers [4] [6] [7]. First step of all automated methods of verification is the translation of legislation into formal models that are input for some tool supporting automated-reasoning (model checking or theorem proving).

The current limitations of automated verification of legislation are:

- These techniques need human assistance. Otherwise model checkers and theorem provers can quickly mire in millions of uninteresting states to be checked.
- The automated verification is purely syntactical because the legislator may intend to have some anomalies, e.g. redundant rules, synonyms, circularities)
- Construction of formal specification, as well as automated checkers, requires significant efforts
- Readability and understandability of formal specifications is limited by end non-experts.

5. Conclusions and Motivation for Further Research

This work on the modeling legislation by using UML was motivated partially by experiences gathered during the development project of the Croatian Court Case Management System. During the phase of project preparation (2000-2003), business models of current legislation, court proceedings and case management were developed, including business use-case model, domain class model and behavioral statecharts/activity models. Statecharts were developed for general case management procedures, as well as for specific bankruptcy, enforcement, litigation and criminal procedures.

This was the prerequisite of system modeling and development phase (started in 2005), that focuses on different aspects of the computer system, such as programs that automate the business process and business rules, database, user interface, system procedures etc. During this phase statecharts are refined and converted to state machine notation.

Based on empirical research, assessment of used method is made. Some improvements of methodology, like semi-automated syntactical verification are promising but require further research.

Other fields of further research are anomalies in legislation. Different types of anomalies in legislation are classified and worked out.

UML seems to be a cure-all with clearly described concepts, standard notations and suggestions for implementation. But application of UML in particular domains, such as modeling of legislation, needs to be researched and evaluated.

References

[1] J. Rumbaugh, I. Jacobson, G. Booch, *"The Unified Modeling Language Reference Manual,"* Addison-Wesley, 1999.

[2]"Unifed Modeling Language (UML): Superstructure, version 2.0," *Object Management Group (OMG), http://www.omg.org, 2005-11-08,* August, 2005.

[3] B. Boehm, "Verifying and Validating Software Requirements and Design Specifications, " *IEEE Software, vol. 1, pp. 75-88,* January, 1984.

[4] IEEE Expert. Special Issue: Validation & Verification of Knowledge- Based Systems, E. Plaza (ed.), vol. 3, 1993

[5] M. Ayel, J. P. Laurent. (eds.), *"Validation, Verification and Testing of Knowledge-Based Systems,"* John Wiley & Sons Ltd., Chichester, England, 1991.

[6] D. Latella, I. Majzik, M. Massink, "Towards a Formal Operational Semantics of UML Statechart Diagrams," *In P. Ciancarini and R. Gorrieri, editors, IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems,* Kluwer Academic Publishers, 1999.

[7] T.M. Van Engers, R. Gerrits, M. Boekenoogen, E.J.J. Glassée, P.J.M. Kordelaar, "POWER: Using UML/OCL for modeling Legislation - an application report," *Proceedings of the International Conference on Artificial Intelligence and Law, ACM 1-58113-368-5/01/0005*, 2001.