Applying a Model Checker to Check Regulatory Compliance of Use Case Models

Motoshi Saeki[1] and Haruhiko Kaiya[2] and Satoshi Hattori[1]

[1] Dept. of Computer Science, Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo 152-8552, Japan
[2] Dept. of Computer Science, Shinshu University
Wakasato 4-17-1, Nagano 380-8553, Japan
{saeki,satoshi}@se.cs.titech.ac.jp,kaiya@cs.shinshu-u.ac.jp

Abstract. This paper proposes the technique to apply model checking in order to show the regulatory compliance of requirements specifications written in use case models. For automatic compliance checking, the behavior of business processes and information systems are specified with use case models and they are translated into finite state transition machines, while we represent regulations with branching time temporal logic (CTL: computational tree logic). By using model checker SMV, we formally verify if the regulations can be satisfied with the state machines.

1 Introduction

Recently, more laws and regulations related to information technology (simply, regulations) are being made and maintained in order to avoid the dishonest usage of information systems by malicious users, and we should develop information systems that are compliant with these regulations. If we developed an information system that was not compliant with the regulations, we could be punished and its compensation could be claimed to us, as a result we could take much financial and social damage. Furthermore, if we would find that the information system that is being developed was not compliant with its related regulations, we would have to re-do its development and its development cost and efforts seriously would increase. It is significant to check as early as possible if a business process and/or a requirements specification of the information system to be developed are compliant with related regulations, in order to reduce its development cost and efforts. In fact, the research topics related to regulatory compliance in requirements engineering area being actively focused on, and the state of the art of this area and some achievements can be found in [11]. However, many works on the automated techniques to check regulatory compliance of requirements specifications are just being studying and developing using various techniques [10, 4, 3].

We propose the (semi-)automated technique to check regulatory compliance of a business process and/or requirements specification of an information system. A business process and the behavior of an information system are modeled with use case models, i.e. a use diagram and use case descriptions which express the behavior of use cases. A use case model is translated into a set of finite state transition machines, which

are concurrently operated. Regulatory statements are formally represented with temporal logical formulas and a model checker verifies if these logical formulas are true in the state transition machines or not. If the logical formulas are true, we can judge the use case model to be compliant with the regulations. The outline of the paper is organized as follows. Section 2 presents how to represent regulatory statements with branching time temporal logic (another name, CTL: computational tree logic, and we use the abbreviation CTL below). We explain the overview of our checking process for regulatory compliance in section 3.

2 Representing Regulations

A typical example of regulations related to IT technology is Japanese Act on the Protection of Personal Information [2] that specifies the proper handling of personal information such as names, addresses and telephone numbers in order to prevent from making misuse of this information. For example, the Article 18, No. 1 of Act on the Protection of Personal Information provides that

> When having acquired personal information, an entity handling personal information must, except in cases in which the Purpose of Use has already been publicly announced, promptly notify the person of the Purpose of Use or publicly announce the Purpose of Use.

According to [7], a regulatory statement consists of 1) the descriptions of a situation where the statement should be applied and 2) the descriptions of obligation, prohibition, permission and exemption of an entity's acts under the specified situation. In the above example, we can consider that "when having acquired personal information, except in cases in which the Purpose of Use has already been publicly announced" is a situation where this act should be applied, while "notify" and "announce" represent the acts of "the entity". These acts are obligations that the entity should do.

There are several works to represent regulatory statements formally using mathematical notations like predicate logic [9] and deontic logic [8]. One of the issues is how to deal with four modalities, obligation, prohibition, permission and exemption using formal logic like predicate logic. Although deontic logic has an expressive power to represent the modalities of obligation and prohibition, its automated reasoning techniques has been less established yet [6, 5]. We use the temporal operators of CTL to represent these modalities. Suppose that we specify the behavior of a business process or an information system with a finite state transition machine. Since state transitions occur non-deterministically in it, there exist several execution paths in the business process or the information system. When we define the states as nodes and the transitions as edges, we can get a tree called *computational tree* that specifies these execution paths. The properties that hold on the tree can be defined with CTL formulas. Suppose that R is a logical formula. We use four types of temporal operators **AF**, **AG**, **EF** and **EG** and their intuitive meanings are as follows. **AF** R is true *iff* R is eventually true for every path, **AG** R is true *iff* R is always true for every path, **EF** R is true *iff* there is a path where R is eventually true, and **EG** R is true *iff* there is a path where R is always true.

Let P and Q be propositions of a situation and an act. The act Q is true *iff* Q is being executed. By using the above four operators, we can represent a regulatory statement with the modalities as follows.

Obligation : P → **AF** Q	Prohibition : P → **AG** ¬Q
Permission : P → **EF** Q	Exemption : P → **EG** ¬Q

In the case of obligation, we should perform Q if the situation P is true, whatever execution path we take. Therefore, Q should be eventually true for every path outgoing from the node P. On the other hand, a regulatory statement of prohibition says that we are not allowed to execute Q on any path. ¬ Q should continuously be true on any node of every path outgoing from P, i.e. Q is always false for every path. If there exists a path where Q is eventually true, Q is permitted to be executed. If there exists a path where Q is always false, we are exempted from executing Q.

In the cases of permission and exemption, although the regulatory statement is not true on a business process or an information system, we cannot say that it violates the regulation. For example, if "P → **EF** Q" (permission of Q) is not true, there are no paths where Q can be executed. Even though the act Q is *permitted*, we don't necessarily need to execute Q and non-execution of Q is not a regulatory violence. However, although the act Q has been permitted by the regulation, if the information system will not have the function to execute Q, it may have a disadvantage to competitors' products having this function in the market. In summary, we can have two categories when a logical formula is not true; regulatory violence and regulatory non-violence. The former category is on obligation and prohibition, and the entity (a business process or an information system) may not execute the acts that are made obligations by regulations, or the entity can execute the acts that are prohibited by regulations. If it occurs, we get a serious problem.

The situation part and the act one in a regulatory statement can be described with logical combinations of case frames as shown in [12]. The technique of case frames was originated from Fillmore's Case Grammar to represent the semantics of natural language sentences. A case frame consists of a verb and semantic roles of the words that frequently co-occur with the verb. These semantic roles are specific to a verb and are called *case*. For example, the case frame of the verb "get", having the cases "actor", "object" and "source", can be described as "get(actor, object, source)", where "get" denotes the acquisition of the thing specified by the object case. The actor case represents the entity that performs the action of "get" and that will own the thing as the result of the "get" action. The source case denotes the entity from which the actor acquires the object. By filling these case slots with the words actually appearing in a sentence, we can obtain its semantic representation. In the example of the sentence "an entity handling personal information acquires from a member her personal information", we can use the case frame of "get" and have "get(entity handling personal information, personal information, member)" as its intermediate semantic representation.

Finally, we can represent the example statement of Article 18, No.1 using case frames and CTL as follows;

$$\text{get(x, Personal\_information, y)} \wedge \neg \text{ announce(x, Purpose of use)} \\ \wedge \text{ aggregation(y, Personal\_information)}$$

$\wedge$ handle(x, Personal_information, Purpose_of_use)
$\rightarrow$ **AF** (notify(x, Purpose_of_use, y) $\vee$ announce(x, Purpose_of_use))

Note that lower case characters such as “x” and “y” in the slots of the above formula stand for variables and we can fill them with any words. In this sense, the formula can be considered as a template.

3 Overview of Checking Process

Figure 1 shows the process of checking regulatory compliance of a business process or an information system. The behavior of the business process or the information system is specified with use case modeling. The description of a use case consists of pre condition, normal flow, post condition and alternate flow, and they are written with simple natural language sentences. These descriptions are translated into a finite state transition machine. In our approach, we also translate regulatory statements into CTLs as shown in section 2, and verify if the CTLs are true on the state transition machine by using a model checker. The words appearing in regulatory statements are different from the words in a use case description, but they may have the same meaning. We need a task for unifying the words that are used in regulations and use cases. “Terminology matching” is for matching the words appearing in the CTLs to those in the state transition machines by using synonym dictionaries such as WordNet.

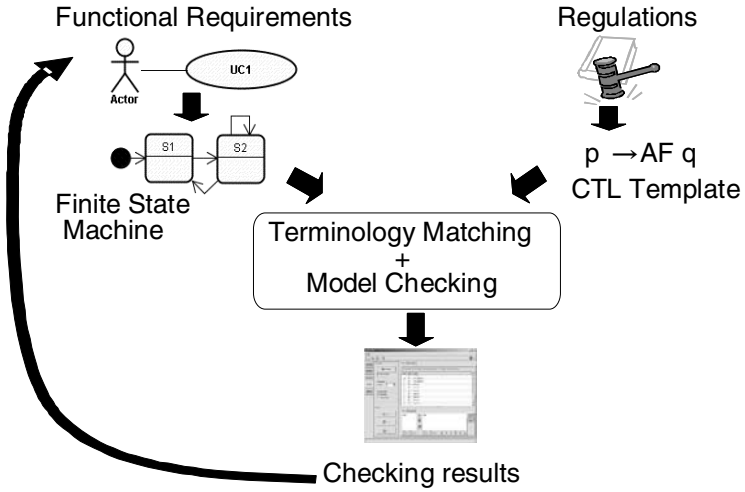

Fig. 1. Overview of a Checking Process

3.1 Translation to State Transition Machines

We use the model checker SMV [1], because it can deal with CTLs. In SMV, a whole system to be checked is represented as a set of concurrent sequential processes and each process is defined as a non-deterministic finite state transition machine (FSM). We consider that use cases in a use case model are concurrently executed and each of them is done sequentially. Thus a use case is translated into a FSM. In a FSM in SMV, state transitions are defined as changes of values of explicitly declared state variables. In our translation technique, we have only one global state variable that store the current state. We consider an action currently executed in a use case as a current state, and the global variable holds the name of the currently executed action. If this action finishes and the next action starts being executed, the name of the next action is assigned to the variable. As for pre and post conditions in a use case, by assigning the name of the condition to the variable, we represent the state where it comes to be true. Suppose that a use case consists of a pre condition, a normal flow A1, A2, ..., An (where A1, ..., An are actions and they are sequentially executed in this order) and a post condition. Its translation is a FSM whose state transitions occur as the sequence of pre condition $\Rightarrow$ A1 $\Rightarrow$ A2, ..., $\Rightarrow$ An $\Rightarrow$ post condition. We have a global state variable *state*, and specify the state transitions in the FSM like "if *state* = precondition then *next(state)* = A1 else if *state* = A1 then *next(state)* = A2 $\cdots$" where *next(state)* denotes the value of *state* after a state transition, i.e. at the next state.

3.2 Terminology Matching

The goal of terminology matching task is 1) converting predicate logical formulas into propositional ones and 2) unifying words in regulatory statements to the words of use case descriptions. Since this task deals with the semantics of sentences, we cannot fully automate it. However, we have a computerized tool to support this task, based on the technique in [12]. The tool has the following functions; 1) analyzing use case descriptions and extracting their case structures, 2) having a dictionary of case frames of regulatory statements, 3) matching case frames of use case descriptions to those of regulatory statements to suggest which statements we should focus on and 4) replacing the words of regulatory statements to unify the used words.

Suppose that a use case has the sentence "The user sends his personal information to the system". We can get a case frame "send(User, Personal_information, System)" as its semantic representation. In this example, the verb "get" in the case frame of Article 18, No.1 is semantically the same as "send" but the flow of the object (personal information) of this act is reverse to "send". We have a dictionary of case frames and it includes information on synonym verbs and their case slots. It also has the rules of replacing a verb and its case slot values, keeping the same meaning. For example, a rule says that the frame "get(actor:x, object:y, source:z)" can be replaced with "send(actor:z, object:y, target:z)". We can match this sentence of the use case description to the situation part of Article 18, No.1, and get the following CTL using the unified case frame by omitting the self-obvious frames "aggregation", "handle" and "announce".

state = "send(User, Personal_information, System)"
$\rightarrow$ **AF** (*state* = "notify(System, Purpose_of_use, User)"

$\vee$ *state* = "announce(System, Purpose_of_use)")

The above is just the CTL formula to be checked if the use case is compliant with Article 18, No.1 and an input to a model checker.

4 Research Agenda

This paper proposes the technique to check regulatory compliance of a business process and an information system by using a model checking. The future work can be listed up as follows.

1. Elaborating the automated technique to translate use case models including alternate action flows into SMV FSMs. In addition, we also consider the other types of descriptions such as UML Activity Diagram and are developing its translation tool.
2. Elaborating the supporting tool and its assessment by case studies, in particular NuSMV is not so powerful from the view of performance and thus we should consider how to deal with scalability problems.
3. Combining tightly our approach to requirements elicitation methods such as goal-oriented analysis and scenario analysis,
4. Managing and improving the requirements that have the potentials of regulatory non-compliance,
5. Developing metrics of measuring regulatory compliance.

References

1. Nusmv home page. `http://nusmv.fbk.eu/`.
2. Act on the protection of personal information. `http://www5.cao.go.jp/seikatsu/kojin/foreign/act.pdf`, 2003.
3. 1st international workshop on requirements engineering and law. `http://www.csc2.ncsu.edu/workshops/relaw/`, 2008.
4. Interdisciplinary workshop: Regulations modelling and deployment. `http://lacl.univ-paris12.fr//REMOD08/`, 2008.
5. P. Castero and T. Maibaum. A Tableaux System for Deontic Action Logic. In *Lecture Notes in Computer Science (DEON2008)*, volume 5076, pages 34–48, 2008.
6. N. Dinesh, A. Joshi, I. Lee, and O. Sokolsky. Reasoning about Conditions and Exceptions to Laws in Regulatory Conformance Checking. In *Lecture Notes in Computer Science (DEON2008)*, volume 5076, pages 110–124, 2008.
7. T. Eckoff and N. Sundby. *RECHTSSYSTEME*. 1997.
8. A. Jones and M. Sergot. Deontic Logic in the Representation of Law: Towards a Methodology. *Aritificial Intelligence and Law*, 1(1):45–64, 2004.
9. S. Kerrigan and K.H. Lawa. Logic-based Regulation Compliance-Assistance. In *Proc. of 9th International Conference on AI and Law*, pages 126–135, 2003.
10. R. Laleau and M. Lemoine, editors. *International Workshop on Regulations Modelling and Their Validation and Verification (REMO2V), CAiSE2006 Workshop*. 2006.
11. P. Otto and A. Anton. Addressing Legal Requirements in Requirements Engineering. In *Proc. of 15th IEEE International Requirements Engineering Conference*, pages 5–14, 2007.
12. M. Saeki and H. Kaiya. Supporting the elicitation of requirements compliant with regulations. In *Lecture Notes in Computer Science (CAiSE'2008)*, volume 5074, pages 228–242, 2008.