DETECTING REGULATORY VULNERABILITY IN FUNCTIONAL REQUIREMENTS SPECIFICATIONS

Motoshi Saeki
Tokyo Institute of Technology, Ookayama 2-12-1-W8-83, Meguro, Tokyo 152-8552, Japan

Haruhiko Kaiya
Shinshu University, Wakasato 4-17-1, Nagano 380-8553, Japan

Satoshi Hattori
Tokyo Institute of Technology, Ookayama 2-12-1-W8-83, Meguro, Tokyo 152-8552, Japan

Keywords: Requirements elicitation, Regulation, Model checking, Use case, Computational tree logic, Regulatory vulnerability.

Abstract: This paper proposes a technique to apply model checking in order to show the regulatory compliance of requirements specifications written in use case models. We define three levels of regulatory vulnerability of a requirements specification by the situation of its non-compliance with regulations. For automatic compliance checking, the behavior of business processes and information systems are specified with use cases and they are translated into finite state transition machines. By using model checker SMV, we formally verify if the regulations that are represented with computational tree logic can be satisfied with the state machines.

1 INTRODUCTION

Information systems to support business processes using Internet have been developed and have come into wide use recently. In this situation, 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 have to re-do its development and its development cost and efforts seriously increase. It is significant to check if a functional requirements specification of the information system to be developed is compliant with regulations as early as possible, in order to reduce its development cost. In an earlier stage of development, we should verify that a functional requirements specification of the information system do not have any *vulnerability* to regulations. Vulnerability means the weak parts that can be attacked so as to cause damages, and threats provide the damages by exploiting the vulnerability. In this paper, we call vulnerability that results from regulations *regulatory vulnerability*. For example, compensation may be claimed and the financial damage of paying its compensation may occur, if your information system is not compliant with regulations. These non-compliant parts of the information system are one of regulatory vulnerability.

We propose the technique to detect formally regulatory vulnerability using a model checker just after specifying functional requirements of an information system. In (Saeki and Kaiya, 2008), in order to detect the possibilities of non-compliance, its authors have developed the technique to represent specification statements and regulatory ones with case frames of Fillmore's case grammar and then to match these case frames. However, it dealt with itemized assertive sentences as specification statements only and it did not consider behavioral aspects such as execution or-

Saeki M., Kaiya H. and Hattori S. (2009).
DETECTING REGULATORY VULNERABILITY IN FUNCTIONAL REQUIREMENTS SPECIFICATIONS.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 105-114
DOI: 10.5220/0002240001050114

der of actions in the system. In this paper, we model the behavior of an information system with use case modeling i.e. a use case diagram and use case descriptions which express the behavior of use cases. A use case model is translated into a finite state transition machine. Regulatory statements are formally represented with temporal logical formulas and a model checker verifies if these logical formulas are true in the state transition machine or not. If the logical formulas are true, we can judge the use case model to be compliant with the regulations. If the formulas are false, since the model checker outputs counterexamples showing the use case behavior unsatisfying the regulations, we can recognize where its regulatory non-compliance exists.

The rest 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) and the typology of regulatory vulnerability based on the structure of logical formulas. In particular, we adopt the new type of regulatory vulnerability to misuse cases. We explain the overview of our detection process for regulatory vulnerability and illustrate its details together with supporting tools in section 3. It includes the translation of use cases into a state transition machine and the terminology matching to retrieve the relevant regulatory statements to be verified. Section 4 shows the regulatory vulnerability to misuse cases using an example. Sections 5 and 6 are for related work and concluding remarks respectively. The essential contribution of this paper is the following three points: 1) the representation of regulations with CTL so that we can use a model checker, 2) the terminology matching technique based on case frames and 3) the usage of misuse cases to detect a specific type of regulatory vulnerability.

2 REGULATION

2.1 Representing Regulations

A typical example of regulations related to IT technology is Japanese Act on the Protection of Personal Information (Cabinet Office, Government of Japan, 2003) that specifies the proper handling of personal information such as names, addresses and telephone numbers of persons 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

> **Article 18, No. 1 of Act on the Protection of Personal Information:**
> 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 (Eckoff and Sundby, 1997), 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 perform.

The first thing that we should address is how to deal with four modalities, obligation, prohibition, permission and exemption using mathematical notation such as formal logic. We use the temporal operators of CTL to represent these modalities. Suppose that we specify the behavior of an information system with a finite state transition machine. Since state transitions occur non-deterministically in it, there exist several execution paths in 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.

The value of a proposition is either true or false at a node. Let P and Q be propositions of a situation and an act respectively. Q is true if the act is being executed. By using the above four operators, we can represent a regulatory statement with the modalities as follows.

Obligation : $P \rightarrow \mathbf{AF}\ Q$
Prohibition : $P \rightarrow \mathbf{AG}\ \neg Q$
Permission : $P \rightarrow \mathbf{EF}\ Q$
Exemption : $P \rightarrow \mathbf{EG}\ \neg Q$

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. $\neg$ 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. Note that CTL has binary temporal operators based on **until** and we can represent with these operators time intervals such as the deadline when an obligation keeps on holding. For simplicity, we will not refer to them throughout this paper but we can deal with them in the same way.

In the cases of permission and exemption, although the regulatory statement is not true on an information system, we cannot say that it violates the regulation. For example, if "P $\rightarrow$ **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 always need to execute Q and non-execution of Q is not a regulatory violation. Thus we can have some categories of the cases where logical formulas of regulations are not true. In the above example, we can have two categories: regulatory violation and regulatory non-violation. The details of these categories will be discussed as a typology of regulatory vulnerability in the next sub section.

We continue to discuss how to represent a regulatory statement with a CTL formula, using as an example the Article 18, No. 1 of Act on the Protection of Personal Information, mentioned in the beginning of this sub section. This article claims the obligation of the acts "announce" or "notify". The situation part and the act one in a regulatory statement can be described with logical combinations of case frames as shown in (Saeki and Kaiya, 2008). 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;

get(x, Personal_information, y)
$\wedge \neg$ announce(x, Purpose_of_use)
$\wedge$ 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 the identifiers of lower case characters such as "x" and "y" stand for variables, and we can fill them with any words. In this sense, the formula can be considered as a template.

2.2 Regulatory Vulnerability

We can classify regulatory vulnerability into the following categories using the four modalities of regulations.

Type 1: The entity (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.

Type 2: The entity cannot execute the acts that are permitted by regulations, or the entity is obligated to execute the acts that are exempted by regulations

Type 3: The misuse cases (wrong or malicious usages) of the entity are permitted by or made obligations by regulations.

The regulatory vulnerability of type 1 is a regulatory violation and this type of vulnerability is a serious problem. On the other hand, the vulnerability of type 2 is not a regulatory violation, as mentioned in section 2.1. However, if the information system will not have the function to execute the act permitted by a regulation, it may have a disadvantage to competitors' products having this function in the market. Moreover, there is a possibility that its users may accuse it of inconvenience because they cannot use the function. Type 3 is not also a regulatory violation. This type may be considered as the weakness of the regulations and therefore they should be improved. Since, however, it takes much longer time to improve them and enforce their improved version officially, rather we select a way of redesigning the underlying business process from technical and/or management aspects so as to mitigate these misuse cases. The risk of

type 1 is the highest in the regulatory vulnerability of three types.

3 DETECTION TECHNIQUE

3.1 Overview

Figure 1 shows the process of detecting regulatory vulnerability in an information system. The behavior of 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 2.1, and verify if the CTLs are true on the state transition machine by using a model checker. If the model checker finds that the CTLs are false, it produces the examples where the CTLs are false, i.e. the counterexamples. We can explore the counterexamples and identify which parts of the use case descriptions may cause the detected vulnerability.

The reason why we adopt use case modeling is that it includes natural language descriptions in addition to its popularity. However, the words, terms and phrases (terms, hereafter) appearing in regulatory statements are different from the terms in the use case descriptions, but they may have the same meaning. We need a task for identify the terms having the same meaning and unify them into a single expression. "Terminology matching" is for matching the terms appearing in the CTLs to those in the state transition machines by using synonym dictionaries such as WordNet. The supporting technique for this task will be illustrated in section 3.4.

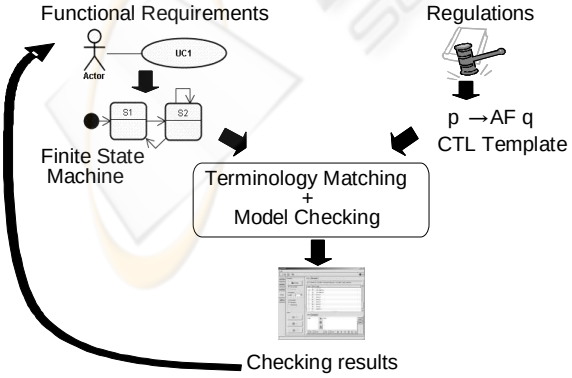

Figure 1: Overview of a Detection Flow.

3.2 Example

In this subsection, we explain an example that will be used throughout this section and the next one. The example is an on-line shop for documents like IEEE Digital Library, and Figure 2 depicts its use case diagram and the use case description of "Retrieve documents". A user retrieves documents in the database of the shop. If she can find a document necessary for her, she buys it from the shop. The user first creates her account (Create an account) and then logs in her account (Login). She inputs retrieval information to the system (Retrieve documents) to get the necessary documents. If the documents are found, the system shows her their abstracts (3rd action of Normal Flow in Retrieve documents). She reads the shown abstracts, and buys the documents by inputting her credit card information if the abstracts are what she wants (Buy documents).

Note that the use case "Distribute documents" is a misuse one that threatens the achievement of the aim of "Buy documents", and its details will be explained in section 4. Ignore it in this section.

3.3 Translating to FSMs

We use the model checker SMV (NuSMV, 2007), because it can deal with CTLs. In SMV, a system to be checked is represented as a set of concurrent sequential processes and each process is defined as a nondeterministic finite state transition machine (FSM). In a FSM, state transitions are defined as changes of values of explicitly declared state variables. SMV has two types of process models: one is called synchronous model where state values are synchronously changed over all of the processes and another is called asynchronous model where state values are changed one by one at each process. Since actions of use cases are asynchronously performed, we adopt an asynchronous model in this paper and a use case model is transformed into asynchronous FSMs.

Each use case is translated into a FSM. 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 in the whole system, 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

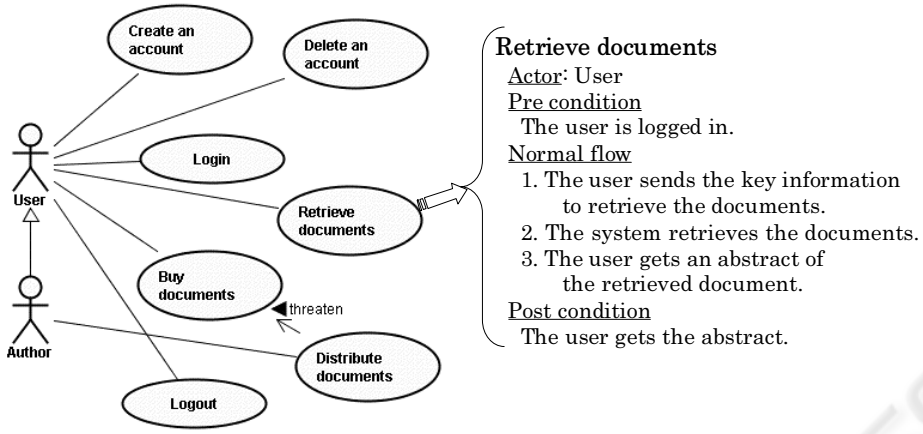

Figure 2: On-line Document Shop.

sequentially executed in this order) and a post condition. Its translation is a FSM whose state transitions occur as the sequence of pre condition ⇒ A1 ⇒ A2, ..., ⇒ An ⇒ post condition, as shown in Figure 3.

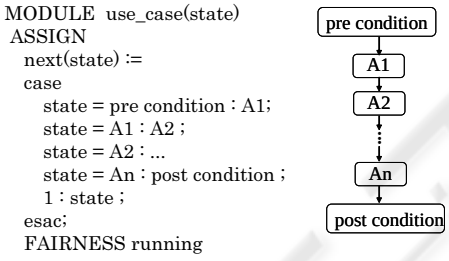

Figure 3: Translating a Use Case Description to a FSM.

The left part of the figure shows the description of a FSM for SMV, and it has only one global variable named "state". The expression "next(state)" denotes the value of "state" at the next state. A case block (case ··· esac) sets the value of the next state according to the current value. The values before and after the colon (:) expresses the current value and the next one respectively. For example, if the current value is "pre condition", the next value is set to "A1" after a transition. The final action "An" is executed and then the "post condition" comes to be true. The bottom line of the case block includes "1" before the colon and it means "otherwise". If the current state is neither "pre condition", "A1", "A2", ..., "An" nor "post condition", the value of the state is not changed. Finally, we can get the translation of the use case model by combining the FSMs of use cases as asynchronous concurrent processes.

Note that we don't intend to propose a new technique to translate use cases into a state transition machine. In fact, more elaborated translation techniques can be found in (Whittle and Jayaraman, 2006; Nebut et al., 2006) and they may be used. The purpose of this sub section is just to show that a use case model can be transformed into a FSM of SMV.

3.4 Terminology Matching

The goal of terminology matching task is 1) retrieving the regulatory statements relevant to a use case description by unifying terms of the regulatory statements to the terms of the use case description, and 2) generating the CTL formulas in SMV-acceptable form from the unified formulas of the regulatory statements.

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 after parsing the sentence. 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)". After this replacement, we fill the variables "x" and "y" with "System" and "User" respectively so as to match with the example sentence the resulting case frame (the situation part of the Article 18, No.1). From the context of the use case descriptions, since it is obvious that the system ("x") handles with personal information and that the user ("y") has personal information, we can omit the predicates "handle" and "aggregate".

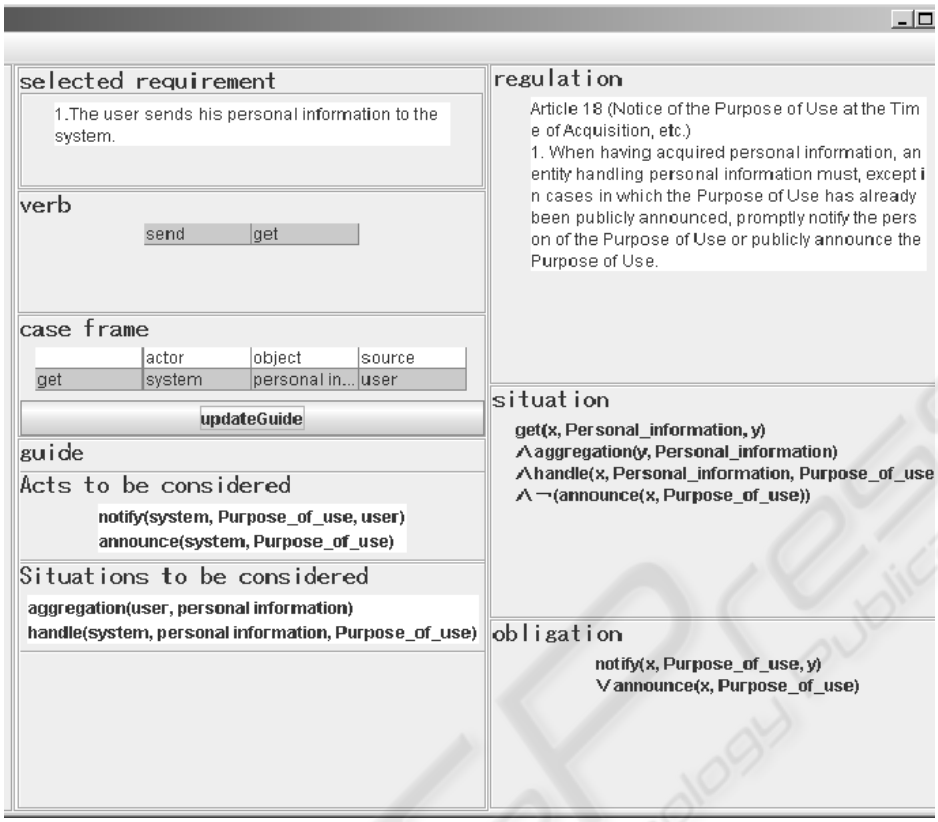

Figure 4: Supporting a Terminology Matching Task.

Finally we have the following CTL as the result of this Terminology Matching task.

state = "send(User, Personal_information, System)"
→ **AF** (state = "notify(System, Purpose_of_use, User)"
∨ state = "announce(System, Purpose_of_use)")

The above is just the CTL formula to be checked if the use case has regulatory vulnerability or not, and an input to a model checker.

Since Terminology Matching 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 (Saeki and Kaiya, 2008). 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) retrieving the regulatory statements which can be matched to case frames of use case descriptions, and 4) generating the CTL formula of regulatory statements by unifying the terms and simplifying it. Figure 4 shows a screen shot of our tool. The tool suggests a pair of use case descriptions and the related regulatory statements after matching their case frames. This matching process uses Japanese-English translation dictionary and synonym ones. We have extracted case frames from the Articles 15 - 36 of Act on the Protection of Personal Information and stored them in a dictionary beforehand. The tool checks if the verb phrases appearing in the use case descriptions can be matched to these case frames. As a result, the left and right areas of the window of Figure 4 show the sentence "The user sends his personal information to the system" of the use case "Create an account" and the Article 18 No.1 respectively, because they can be matched. Some information helpful to produce the input CTL formula is displayed in the other area of the window, e.g. situations and acts to be considered during producing the formula.

3.5 Model Checking

The model checker of SMV is called NuSMV. Since NuSMV checks if a formula is true at the initial state of a FSM, we attach the operator AG in the head of the formula that was obtained in the Term Matching task.

The NuSMV shows that the CTL of Article 18 No.1 is false, and we can recognize that our example has a regulatory violation. Since the verified CTL

is an obligation, this violation is the regulatory vulnerability of type 1. The counterexample that the NuSMV outputs suggests the instance of execution paths where the CTL comes to be false. According to it, as shown in Figure 5, we can recognize that the following scenario caused the regulatory violation. After executing the use case Create an account, Login is executed and then its login is successful. After that, the Logout use case is immediately executed. Login and logout are iterated as a loop. This path does not satisfy the CTL of Article 18 No.1 specifying the obligation of the action "notify" or "announce".

Figure 6 illustrates the verification example mentioned above. Note that we used abbreviated literal forms instead of string data types to represent the values of "state" variable for brevity and the ability of NuSMV. For example, we used the literals send_personal_info and notify_purpose in the NuSMV tool instead of the strings "send(User, Personal_information, System)" and "notify(System, Purpose_of_use, User)" respectively. The figure includes two windows; one is the verification result and another is a counterexample. The rear window shows that the CTL of Article 18 No.1 is false (the CTL numbered with 0 in the Property tab), and we can recognize that our example has a regulatory violation. The counterexample is shown in the front window of the figure. According to it, after executing the use case Create an account (step 21: issue_ID, which denotes "3. The system issues login ID and password to the user" in Figure 5), Login is executed (steps 24-30: input_ID, denoting "1. The user inputs login ID and password to the system" in Figure 5) and then its login is successful (step 31: logged_in, the post condition of the Login use case). After that, the Logout use case is immediately executed (step 32: exit). Login and logout are iterated as a loop (step 23 → 33 → 23).

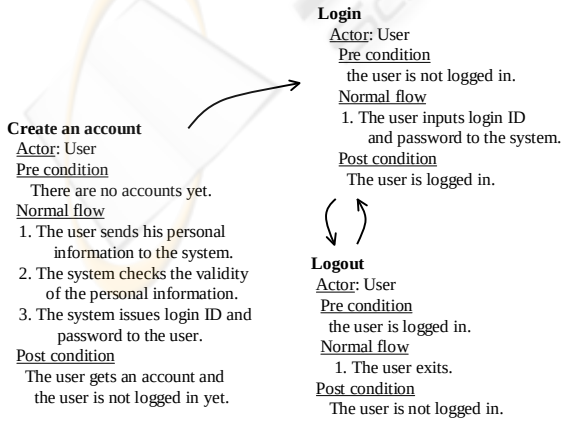

Figure 5: A Counterexample.

By investigating the counterexamples, we can identify where regulatory vulnerability is in the use case descriptions and what parts we should correct. In this example, we can find that there is regulatory vulnerability in either Create an account, Login or Logout and that we can resolve it by inserting the obligation act "notify(System, Purpose_of_use, User)" to either of them. We add this act to the use case Create an account.

4 USING MISUSE CASES

Let's move to regulatory vulnerability of type 3. We add one or more misuse cases to a normal use case model compliant with a regulation and check if the regulation has ability to legally prevent the execution of the added misuse cases. If it has no ability, the use case model includes regulatory vulnerability of type 3 for the regulation. Figure 7 shows the cases of the results of checking. Suppose that a normal use case model U is compliant with the regulation p as shown in the case (a) of the figure. Under this case, we add a misuse case MU and get the case (b), i.e. a false value as a result of checking. The case (b) asserts that the regulation p can legally prevent the execution of MU, because adding MU causes the regulatory violation. On the other hand, the case (c) may be problematic. Its detail including case (d) will be explained below using the example.

Our example shown in Figure 2 contains a misuse case "Distribute documents". When a user of the on-line document shop wants the full texts of the retrieved documents, she should buy them. There is another alternative to get the full text without any fee by asking its author to send it to her. The use case "Distribute documents" specifies this behavior and it threatens the achievement of "Buy documents". Its description is shown as follows.

```
Distribute documents
 Actor: User, Author
 Pre condition
  The user gets the abstract.
 Normal flow
  1.The user asks for the author
          to send the document.
  2.The author sends the document to the user.
  3.The user gets the full text of the document.
 Post condition
  The user gets the document.
```

After executing "Retrieve documents" and getting the abstract of the document that she wants, she executes "Distribute documents". She first asks its author to send its full text to her, and the author sends the full text to her. As a result, she can get the full text of the

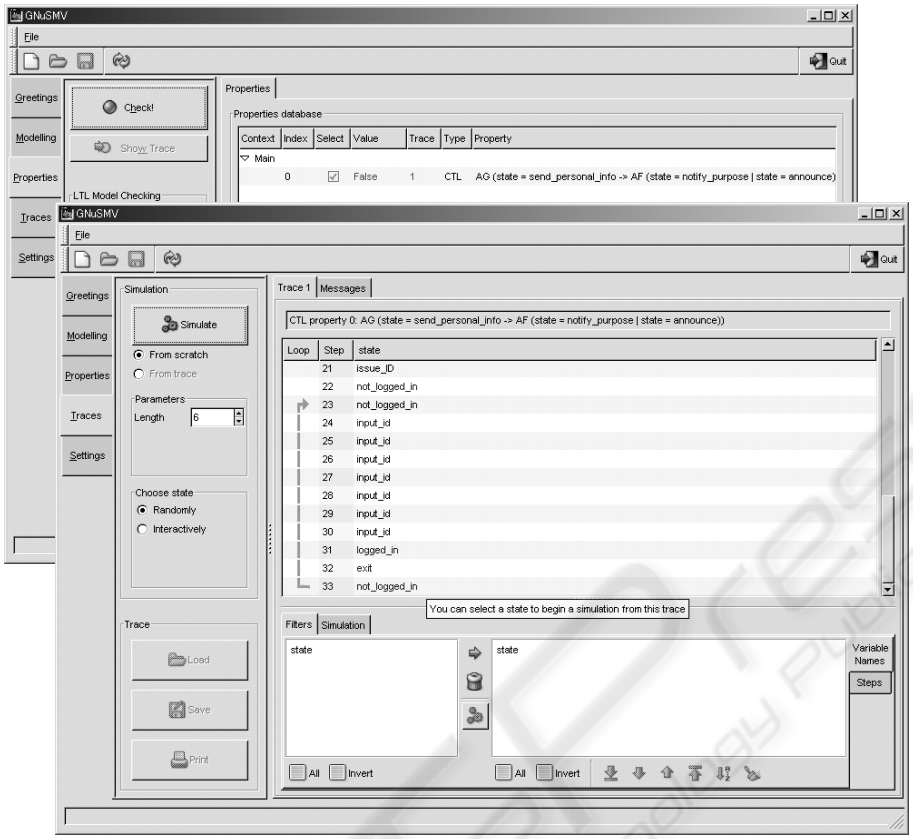

Figure 6: Verification Example with NuSMV.

document free of charge and this use case obviously prevents the aim of the on-line document shop, i.e. earning money by “Buy documents”.

Suppose that a regulation related to copyright prescribes the following;

> By request, an entity can send the contributions produced by the entity itself to a third party acting in good faith.

This statement, denoting a permission, can be transformed to a logical combination of case frames as follows.

$$\text{request}(x, y, z) \wedge \text{produce}(z,y) \rightarrow \textbf{EF}\ \text{send}(z,y,x)$$

where “x”, “y” and “z” are a person who wants the document, the document and the author of the document respectively. Through terminology matching process, we replace “request(x,y,z)” with “ask(x,y,z)” using the synonym dictionary because the first sentence of the use case description includes the verb “ask”, and get the following formula to be checked.

$$\textbf{AG}\ (\text{state}=\text{“ask(User, Document, Author)”} \rightarrow \textbf{EF}\ \text{state}=\text{“send(Author,Document,User)”})$$

NuSMV returns the value true and it means that our on-line document shop together with the misuse case is compliant with the regulation. That is to say, the regulation cannot legally prevent the execution of the misuse case “Distribute documents” and this is the case (c) shown in Figure 7. Although this case includes the possibility of type 3 vulnerability, there may be the possibility that the regulation p can be thoroughly irrelevant to MU, e.g. the acts of the regulation p are not included in MU at all. To investigate the relevance of p to MU, we should check if the execution paths compliant with p include the situations and the acts of p. We negate p and obtain counterexamples from the model checker. If a counterexample includes the situations and the acts of p, p is relevant to MU and permits the execution of MU, i.e. a regulatory vulnerability of type 3. However, NuSMV outputs many useless counterexamples where the situation is false only. In fact, it outputs many counterexamples that do not include state=“ask(User, Document, Author)”. To filter out these useless counterexamples, we negate the act part of the CTL and get the counterexamples where the situation holds. The counterexamples of AG ($s \rightarrow \neg a$) make EF ($s \wedge a$) [1]

[1] $\neg$ AG ($s \rightarrow \neg a$) = EF ($s \wedge a$)

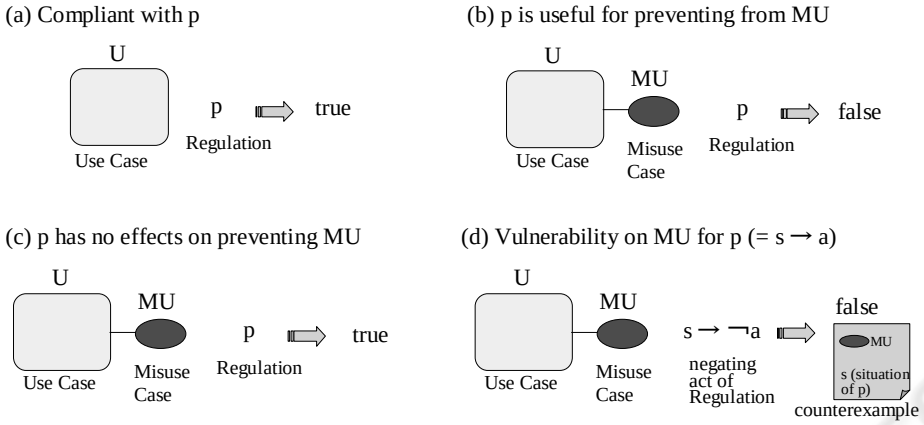

Figure 7: Computation Tree and Modality of Regulations.

true, and we can get the execution paths where both the situation s is true and the act a is executed. This is shown in the case (d) of Figure 7.

In our example, we negate the act part of the CTL of the regulation and input it to NuSMV in order to obtain counterexamples. The input formula is as follows:

AG (state="ask(User, Document, Author)"
→ ¬**EF** state="send(Author,Document,User)")

If the obtained counterexamples have the execution paths where the misuse case is included, the regulation permits the execution of the misuse case and there is a regulatory vulnerability of type 3.

NuSMV has produced a counterexample where the action "send(Author,Document,User)" in the misuse case is executed after the execution of "Retrieve documents". That is to say, the regulation permits the execution of this misuse case. It is not a regulatory violation, but an undesirable case because the misuse case can be performed and as a result causes financial damages to the shop. To avoid it, we can consider two alternatives: one is to modify the related regulations and another is to change this underlying business process. As the example of the second alternative, i.e. changing the business process, we can take the way of getting a fee from an author when she registers her contributions to this shop. It could mitigate financial damage to the on-line shop. Note that it is just an example and that we can take the other ways to mitigate the financial damage of the shop. Detection of type 3 vulnerability is significant to evolve a business process and an information system incrementally into more robust one.

5 RELATED WORK

The research topics related to regulatory compliance in requirements engineering area being actively focused on. The state of the art of this area and some achievements can be found in (Otto and Anton, 2007). We can find many approaches to represent regulations with formal expressions (REMO2V, 2006; REMOD, 2008; RELAW, 2008), and many of them used classical logic as formal representations of regulations. For example, Hassan et al. used logical formula to represent regulatory statements and enterprise requirements, and Alloy analyzer to check the consistency between them (Hassan and Logrippo, 2008). Although they benefited from powerful and well-established theorem provers and model checkers, an issue on the treatment of the modalities of regulatory statements still remains. Furthermore, they did not consider the classification of regulatory vulnerability, in particular, the detection of type 3 vulnerability. Analyzing regulatory compliance of misuse cases leads to the elicitation of a class of non-functional requirements such as security, safety, reliability etc. Deontic logic is one of the alternatives to represent the modalities of obligation and prohibition more intuitively and comprehensively (Jones and Sergot, 2004). However, its theorem prover or model checker has been less established yet rather than CTL (Dinesh et al., 2008; Castero and Maibaum, 2008). There are some studies on applying goal-oriented analysis to regulations in order to identify the rationales of and the dependencies among regulatory statements. (Darimont and Lemoine, 2006). Although their aim does not have the same direction as ours, we can consider deeper analysis by checking regulatory compliance from the viewpoints of rationales of regulations and requirements specifications.

6 CONCLUSIONS AND FUTURE WORK

This paper presents the technique to detect regulatory non-compliance of an information system by using a model checking. In addition, we classified regulatory non-compliance as regulatory vulnerability into three categories and emphasized type 3 vulnerability. 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 to retrieve and manage counterexamples. The functions on manipulating counterexamples are significant to resolve regulatory vulnerability and the methodology how to find from the counterexamples the solutions to mitigate regulatory vulnerability should be developed.
3. Developing a supporting technique for the processes to translate correctly regulatory statements into CTL formulas. In addition, since regulatory documents include meta level descriptions such as application priority of the statements, the technique to model them should be more elaborated.
4. Considering how to deal with scalability problems on the techniques of model checking.
5. Dealing with non-functional requirements such as security.
6. Combining tightly our approach to requirements elicitation methods such as goal-oriented analysis and scenario analysis,
7. Developing the technique to manage and improve the requirements that have the potentials of regulatory non-compliance,
8. Developing metrics of measuring compliance, in fact three types of regulatory vulnerability can be considered as strength degrees of non-compliance.

REFERENCES

Cabinet Office, Government of Japan (2003). Act on the protection of personal information. http://www5.cao.go.jp/seikatsu/kojin/foreign/act.pdf.

Castero, P. and Maibaum, T. (2008). A Tableaux System for Deontic Action Logic. In *Lecture Notes in Computer Science (DEON2008)*, volume 5076, pages 34–48.

Darimont, R. and Lemoine, M. (2006). Goal Oriented Analysis of Regulations. In *REMO2V, CAiSE2006 Workshop*, pages 838–844.

Dinesh, N., Joshi, A., Lee, I., and Sokolsky, O. (2008). Reasoning about Conditions and Exceptions to Laws in Regulatory Conformance Checking. In *Lecture Notes in Computer Science (DEON2008)*, volume 5076, pages 110–124.

Eckoff, T. and Sundby, N. (1997). *RECHTSSYSTEME*.

Hassan, W. and Logrippo, L. (2008). Requirements and Compliance in Legal Systems: a Logic Approach. In *Requirements Engineering and Law (RELAW 2008)*, pages 40–44.

Jones, A. and Sergot, M. (2004). Deontic Logic in the Representation of Law: Towards a Methodology. *Aritificial Intelligence and Law*, 1(1):45–64.

Nebut, C., Fleurey, F., Traon, Y., and Jezequel, J.-M. (2006). Automatic Test Generation: A Use Case Driven Approach. *IEEE Transaction on Software Engineering*, 32(3):140–155.

NuSMV (2007). Nusmv: A new symbolic model checker. http://nusmv.fbk.eu/.

Otto, P. and Anton, A. (2007). Addressing Legal Requirements in Requirements Engineering. In *Proc. of 15th IEEE International Requirements Engineering Conference*, pages 5–14.

RELAW (2008). 1st international workshop on requirements engineering and law. http://www.csc2.ncsu.edu/workshops/relaw/.

REMO2V (2006). International Workshop on Regulations Modelling and Their Validation and Verification (REMO2V), CAiSE2006 Workshop. http://lacl.univ-paris12.fr//REMO2V/.

REMOD (2008). Interdisciplinary workshop: Regulations modelling and deployment. http://lacl.univ-paris12.fr/REMOD08/.

Saeki, M. and Kaiya, H. (2008). Supporting the elicitation of requirements compliant with regulations. In *Lecture Notes in Computer Science (CAiSE'2008)*, volume 5074, pages 228–242.

Whittle, J. and Jayaraman, P. (2006). Generating Hierarchical State Machines from Use Case Charts. In *Proc. of 14th IEEE Requirements Engineering Conference (RE2006)*, pages 19–28.