VERIFICATION OF SCENARIOS USING THE COMMON CRITERIA

Atsushi Ohnishi and Hiroya Itoga
Department of Computer Science, Ristumeikan University, 1-1-1 Noji-Higashi, Kusatsu, Shiga, Japan

Keywords: Scenario analysis, security requirements elicitation.

Abstract: Software is required to comply with the laws and standards of software security. However, stakeholders with less concern regarding security can neither describe the behaviour of the system with regard to security nor validate the system's behaviour when the security function conflicts with usability. Scenarios or use-case specifications are common in requirements elicitation and are useful to analyze the usability of the system from a behavioural point of view. In this paper, the authors propose both (1) a scenario language based on a simple case grammar and (2) a method to verify a scenario with rules based on security evaluation criteria.

1 INTRODUCTION

Scenarios are important in software development (Cockburn, 2001), particularly in requirements engineering (Alexander and Maiden, 2004), since they provide concrete system description (Sutcliffe et al., 1998), (Weidenhaupt, 1998). Moreover scenarios are useful in defining system behaviors done by system developers and validating the requirements undertaken altogether with customers (Carroll, 2000). In many cases, scenarios become foundation of system development. Incorrect scenarios will lead to negative impact on system development process in overall. However, scenarios are informal and it is difficult to verify the correctness of scenarios. The errors in incorrect scenarios may include:

1. Vague representations,
2. Lack of necessary events,
3. Extra events,
4. Wrong sequence among events.

The authors have developed a scenario language for describing scenarios in which simple action traces are embellished to include typed frames based on a simple case grammar (Fillmore, 1968) of actions and to describe the sequence among events (Zhang and Ohnishi, 2004). Since this language is a controlled language, the vagueness of the scenario written using this language can be reduced (Ohnishi, 1996). Furthermore, the scenario created with this language can be transformed into internal representation. In the transformation, both lack of cases and illegal usage of noun types can be detected, and concrete words will be assigned to pronouns and omitted indispensable cases (Ohnishi, 1996). As a result, the scenario with this language can avoid errors typed 1 previously mentioned.

Furthermore, software security requirements affect the whole behavior of the software system and not only parts of the system. Most stakeholders may not be software security professionals. Almost all users and clients of the system will have no knowledge about software security. However, they still feel that it is important to comply with the laws and standards for information systems and software security.

Therefore, they may suggest requirements to comply with such standards although they may not be able to envision the behavior of the system once these suggestions are incorporated. Consequently, it is necessary for them to leave validation of requirements about their business rules to developers who do not have knowledge about the business rules.

Although developers have knowledge about software security, this is usually limited to general knowledge. They cannot decide who to apply the techniques of software security to specific business rules. If the system satisfies the laws and the standards, the users may find that the behavior differs from that initially envisioned after the completion of development.

Ohnishi A. and Itoga H. (2008).
VERIFICATION OF SCENARIOS USING THE COMMON CRITERIA.
In *Proceedings of the Third International Conference on Software and Data Technologies - SE/GSDCA/MUSE*, pages 5-11
DOI: 10.5220/0001873900050011

Behaviors related to software security often conflict with other requirements, such as usability, cost and performance. Therefore, the developers cannot include all functional and non-functional requirements regarding software security in the software requirements specification.

There are a number of reasons why it is necessary to focus on elicitation of security and usability requirements. The requirements regarding usability will conflict with security requirements. The remaining requirements, such as cost or performance requirements, can be resolved by increasing other resources.

We focus on the verification method of scenarios with rules (Toyama and Ohnishi, 2005) and customize the rules to satisfy the software security common criteria (ISO/IEC 15408, 2005).

2 SCENARIO LANGUAGE

2.1 Outline

Our scenario language has been already introduced in several papers such as in (Zhang and Ohnishi, 2004), (Ohnishi and Potts, 2001). However, in this paper, a brief description of this language will be given for convenience.

A scenario can be regarded as a sequence of events. Events are behaviors employed by users or systems for accomplishing their goals. We assume that each event has just one verb, and that each verb has its own case structure. The scenario language has been developed based on this concept. Verbs and their own case structures depend on problem domains such as elevator control (Ohnishi and Potts, 2001), PC chair's job (Barish, 1997) and train ticket reservation (Railway Information System, 2001), but the roles of cases are independent of problem domains. The roles include agent, object, recipient, instrument, and source, etc (Fillmore, 1968), (Ohnishi and Potts, 2001).

We provide Requirements Frames (Ohnishi, 1996) in which verbs and their own case structures are specified. This frame depends on problem domains. Each action has its case structure, and each event can be transformed into internal representation based on this frame. In the transformation, concrete words will be assigned to pronouns and omitted indispensable cases. With Requirements Frame, we can detect both the lack of cases and the illegal usage of noun types (Ohnishi, 1996).

We assume four kinds of time sequences among events: 1) sequence, 2) selection, 3) iteration, and 4) parallelism. Actually most events are sequential events. Our scenario language defines the semantic of verbs with their case structure. For example, data flow verb has source, goal, agent, and instrument cases. Since such case structure can define the abstraction level, scenario provided using our scenario language becomes the almost same level of the abstraction.

2.2 Scenario Example

We consider a scenario of train ticket reservation in a railway company. Figure 1 shows a scenario of customer's purchasing a ticket of express train at a service center. This scenario is written with our scenario language based on a video that records behaviors of both a user and a staff at one particular service center.

[Scenario title: A customer purchases a train ticket of reservation seat]
[Viewpoints: Staff, customer]
[Pre-condition: the customer has enough money to buy a ticket & has not a ticket & has not reserved a seat]
[Post-condition: the customer will get a ticket & reserved a seat]
1. A staff asks a customer leaving station and destination as customer's request.
2. He sends the customer's request to reservation center with a terminal.
3. He retrieves available trains with the request.
4. He informs the customer of a list of available trains.
5. The customer selects a train that he/she will get.
6. The staff retrieves available seats of the train.
7. He shows a list of available seats of the train.
8. The customer selects a seat of the train.
9. **If** *(there exists a seat selected by the customer)* **then** *the staff reserves the seat with the terminal.*
10. He gets a permission to issue a ticket of the seat.
11. He receives money for the ticket from the customer.
12. He gives the ticket to the customer.

Figure 1: Scenario example.

A title of this scenario is given at the first two lines in Fig.1. Viewpoints of considered scenario are specified at the third line. In this paper, viewpoints mean active objects such as human or system appearing in the scenario. There exist two viewpoints, namely staff and customer. The order of specified viewpoints means the priority. In this example, the first featured object is staff and the second one is customer. In such a case, the former becomes the subject of an event.

In addition, pre-condition specifies a condition that satisfies at the start of the scenario. Post-condition specifies a condition that satisfies at the

end of the scenario.

In this scenario, most events are sequential, except one selective event (the 9th event). Selection can be expressed with if-then syntax like program languages. Actually, event number is for reader's convenience and not necessary.

3 VERIFICATION OF SCENARIOS

When a scenario is described, necessary events may be missing, unnecessary events may be mixed or time sequence among events may be inaccurate. These errors may have a negative impact on system development; therefore, it is necessary to detect these errors. The errors, also employed as correctness verification items, of scenarios include:

1. Lack of necessary events
2. Extra events
3. Wrong time sequence among events

We can check whether an event is lacking, being extra one or being sufficient one by comparing its correct occurrence times with the times that it occurred in the scenario. Similarly, we can check whether the time sequence among events is wrong by comparing the correct time sequence with the time sequence described in the scenario.

We propose a method to verify the correctness of scenarios by using rules to detect the errors in scenarios. We assume that a rule is a description of the correct occurrence times of an event and/or the correct time sequence among events, which the scenario ought to satisfy. One scenario may be verified with several rules.

3.1 Rule

Rule is composed of the description of rule's event and the description of event's occurrence times and/or time sequence among events. In this sense, our rules just specify the occurrence of events and/or the sequence of events. If the abstraction level of events of rules becomes high, the rules can be applied to scenarios of several different domains.

3.1.1 Events in Rule

In a rule, there are one or more events, whose occurrence times and/or time sequences are specified. When a scenario is verified with a rule, the rule's events can correspond to the scenario's events. By finding the corresponding events in the scenario, and by checking the occurrence times and/or the time sequence of these events, one scenario can be verified.

As a result, it is necessary to get the corresponding relation between the rule's events and the scenario's events. For this reason, the rule's events are also described based on Requirements Frames, and can also be transformed into the internal representation. If the rule's event and the scenario's event have the same internal representation, then they are deemed as corresponding events.

In order to improve the verification effect, it is not sufficient that the corresponding relation between the rule's event and the scenario's event is 1 to 1 ratio. It is expected that a rule's event can correspond to several scenario's events. As a result, the occurrence times and/or the time sequence of these scenario's events can be checked with one rule. In such a case, the rule's event has an abstract representation, and the corresponding scenario's events have several concrete representations.

For the above reason, we permit the abstract description of rule's event. An abstract event may be transformed into several concrete events, when finding its corresponding events in the scenario. At this time, the corresponding relation between the rule's event and the scenario's event is 1 to many. There are two kinds of abstract events in the rule.

1. Some indispensable cases are omitted in the event sentence.
2. There include "something," "someone," "same thing," "same one," etc. in the event sentence.

In the first kind of abstract events, the omitted cases fit any noun. This kind of abstract events will be transformed into concrete events by substituting concrete nouns for omitted cases, when finding its corresponding events in observed scenario. For example, a rule's event "system feedbacks to user" can be transformed into an internal representation.

This event can correspond to any scenario's event whose action is "feedback, etc.", agent case is "system", and recipient case is "user". In the second kind of abstract events, "something" / "someone" is similar to the omitted case in the first kind of abstract events, and fit anything / anyone. The reason of dividing abstract events into two kinds should be explained. We assume that there is a rule that describes the time sequence among events. Under this rule there exist two separate events and under these two events there exist a case A and a case B. We want to specify that case A and case B can have any content but they have to be the same content. If we simply omit case A and case B in the

rule description, it cannot be warranted that case A and case B have the same content. By specifying "something" / "someone" for case A, "same thing" / "same one" for case B, case A and case B can have any content, and they are the same content.

In the second kind of abstract events, "same thing" / "same one" fits the same noun with "something" / "someone" that appears in the same rule. This kind of abstract events will be transformed into concrete events by substituting concrete nouns for "something" / "someone" / "same thing" / "same one", etc. when finding its corresponding events in the scenario.

3.1.2 Occurrence Times of an Event

The correct occurrence times of an event, which the scenario ought to satisfy, can be specified as a rule. By comparing the correct occurrence times with the times that this event occurred in the scenario, whether this event is lack of or excess of occurrence can be checked. The occurrence times of an event are described based on regular expression as follows.

1. *E*: event E occurs just one time.
2. *E+*: event E occurs one or more times.
3. *E?*: event E occurs one time or does not occur.
4. *E!*: event E never occurs.
5. *E{m}*: event E occurs m times.
6. *E{m,}*: event E occurs m or more times.
7. *E{,n}*: event E occurs n or less times.
8. *E{m,n}*: the occurrence times of event E is from m to n.

We adopted the syntax of regular expression and similar its semantics.

3.1.3 Time Sequence Among Events

The correct time sequence among events that scenario ought to satisfy can be specified as rules. By comparing the correct time sequence with the time sequence described in the scenario, time sequence among events can be checked. According to the time sequence in the scenario described in section 2.1, we assume the following rules.

1. *Before/After E1, E2*: Before/After event E1 occurs, event E2 should occur.
2. *If (condition) (E1, E2)*: Event E1 and event E2 occur selectively. If the condition is true, event E1 occurs. If the condition is false, event E2 occurs.
3. *Do (E1,E2,...) until(condition)*: Until the condition becomes true, event E1, E2, ... occur iteratively.
4. *AND(E1,E2,...)*: All of the events E1, E2, and others parallel occur.
5. *OR(E1,E2,...)*: One of the events E1, E2, ... or more parallel occurs.
6. *XOR(E1,E2,...)*: Just one of the events E1, E2, ... occurs.

As previously described, when a scenario is verified with a rule that includes the abstract event, it is possible that an abstract event corresponds to several scenarios' events. In this case, it is necessary to check the time sequence of every corresponding event in the scenario and the results should be shown one by one.

3.2 Scenario-checking with Rule

Our scenario-checking procedure consists of two phases. The first phase is selection of applicable rules from rule DB. We specify both (1) pre-conditions and post-conditions and (2) viewpoints in each of the rules. When the conditions and viewpoints of a rule are much the same as those of a given scenario, the rule is selected for checking the scenario.

The second phase is analysis of rules and checking the consistency between a rule and the scenario as described in 3.1. The result will be passed to a checking-system user. The scenario checking can be achieved by automatically checking whether the scenario satisfies the rules, through the internal representation of scenario and the internal representations of rules.

```
Initialize a counter (counter=0).
Find the scenario's event that corresponds to the rule's event
while (the corresponding event in the scenario will be found)
  do
    counter=counter+1
  Show the corresponding event and its occurrence condition to user.
  Find the next scenario's event that corresponds to the rule's event.
  od
Compare the occurrence times specified in the rule with the counter, and show the result.
```

Figure 2: Checking procedure of the occurrence of events.

We firstly find events in a scenario each of which corresponds to an event in a rule as described in 3.1.1. When a scenario is checked with a rule, the occurrence times and/or the time sequence of corresponding events in the scenario will be checked.

We provide two checking procedures. One is for checking the occurrence time and the other is for checking the time sequence. Figure 2 shows outline of checking procedure for the occurrence time of an event and Figure 3 shows outline of checking procedure for the time sequence between events E1 and E2.

```
Find the scenario's event that corresponds to E1 from the beginning of the scenario.
if (the corresponding event of E1 not be found)
  then show this error, and the checking ends.
  else do
      Show the corresponding event and its occurrence condition.
      Find an event that corresponds to E2 and satisfies the time sequence
       if (the corresponding event of E2 not be found)
         then show the result that the scenario does not satisfy the rule
         else do
             Show the corresponding event and its occurrence condition.
             Find the next scenario's event that corresponds to E2 and satisfies the time sequence.
            until (the corresponding event of E2 not be found)
       fi
       Find the next scenario's event that corresponds to E1.
     until (the corresponding event of E1 not be found)
fi
```

Figure 3: Checking procedure of the sequence of events.

3.3 Evaluation

We have developed a prototype system based on the method. There exist 35 errors in 15 scenarios. These 35 errors can be classified into three categories, namely (1) wrong sequence of events, (2) lack of events, and (3) extra events. The number of errors grouped into the above categories are 16, 8 and 11, respectively. We could detect part of these errors with our method shown in Table 1.

The detection ratio is 63%. The reason why our method seems to be weak for detecting extra events is that it is very difficult to predict extra events and make rules for them in advance. On contrast, it is not so difficult to predict indispensable events and correct sequence of events and make rules for them in advance. Another reason why our method is strong for detecting wrong sequence of events and lack of events is that rules for the occurrence time of events are effective to detect lack of events and rules for the sequence of events are effective to detect wrong sequence of events. To improve the detection ratio, we have to introduce another type of rules for detecting extra events.

The describers can easily correct the detected errors. Since scenario writers can determine the abstraction level of scenarios, the number of events may differ depending on the scenario writers. Causes of undetected errors related to both lack of events and extra events are misunderstandings of the reservation jobs.

Table 1: Detected errors in scenario.

	The number of errors	**The number of detected errors**
Wrong sequence of events	16	14
Lack of events	8	6
Extra events	11	2
Total	35	22

3.4 Rules Based on Common Criteria

7. Class FIA: Identification and authentication

7.1 Authentication failures (FIA_AFL) require that the system be able to terminate the session establishment process after a specified number of unsuccessful user authentication attempts. It also requires that, after termination of the session establishment process, the system be able to disable the user account or the point of entry from which the attempts were made until an administrator-defined condition occurs.

FIA_AFL.1.1.
The security function shall detect when [selection: *[assignment: positive integer number], an administrator configurable positive integer within [assignment: range of acceptable values]*] unsuccessful authentication attempts occur related to [assignment: *list of authentication events*].

FIA_AFL.1.2.
When the defined number of unsuccessful authentication attempts has been met or surpassed, the security function shall [assignment: *list of actions*].

Figure 4: An excerpt of security evaluation criteria.

[Authentication failures][system, user]
{
The system requests the user for a password.
The system receives the password from the user.
The system authenticates the user *via* the password
if (unsuccessful authentication attempts meet or exceeds **3 times**) then
The system switches to "**Invalidate ID**".
fi
}

Figure 5: An example of a scenario.

The security evaluation criteria suite used in this paper is ISO/IEC 15408 Evaluation Criteria for IT Security (ISO/IEC 15408, 2005). The evaluation criteria suite is useful to verify scenarios because some of them are easily represented as rules. We can detect scenarios which do not satisfy rules based on the security evaluation criteria.

We can verify the scenario shown in Fig. 5 with rules shown in Fig. 4 and confirm the scenario satisfies the rules.

If the authentication event occurs more than three times, the verifier can detect the unsuccessful status and also detect the switching to "invalidate ID." These results will be provided to a user and he will judge the correctness of the scenario.

4 DISCUSSION

ISO/IEC 15408 is a most commonly used security evaluation criteria suite and it has descriptions of functional requirements classified according to purpose and function of IT system to be developed. The suite also has descriptions of simple behaviors to meet the functional requirements.

In the security evaluation criteria suite, there are 68 families of functional requirements into 11 components, such as encryption, authentication, *etc*. In this paper, we focus mainly on usability, and consider 12 functional requirements families and other related families, because these families can be easily represented as rules for the verification.

The remaining requirements families are for quality of function or evaluation of the functions themselves, and so they do not affect the behaviors that can be seen by the users. The remaining families are difficult to transform rules. This point is a major problem of the proposed research. Actually, we can represent 23 rules for security requirements of the security evaluation criteria, while 44 security requirements of the criteria cannot be represented as rules.

In this paper, we consider "7.1 Authentication failures" and show the verification process of the scenario with rules.

The rules must be written by security professionals to detect wrong scenarios which involve incorrect security behavior that causes a critical vulnerability in the system. The professionals can rewrite the scenario to meet the security evaluation criteria into scenario easily.

Some criteria are not suitable to represent as rules and scenarios cannot be verified with these points of view. To solve this problem is left as a future work.

5 RELATED WORKS

Araujo and Whittle *et al.* proposed an analysis method in their scenario description process (Araujo, Whittle and Kim, 2004) (Whittle and Araujo, 2004). This method focuses on generation of state machines by synthesizing scenarios and validating their correctness. This method is very useful for requirements analysis and the design process after requirements elicitation. Instead, our method focuses on validation in the requirements elicitation phase and finding the conflicts of requirements.

Sindre and Opdahl proposed Misuse Cases (Sindre and Opdahl, 2005) and McDermott and Fox proposed Abuse Cases (McDermott and Fox, 1999) for security requirements elicitation. These methods are useful for brainstorming or discussion by clarifying the threats. However, our method uses the security evaluation criteria and focuses on the comprehensive elicitation of security requirements.

SIREN is a security requirements management method that focuses on security evaluation criteria or common criteria (Toval et al., 2002), but this method focuses on reuse of requirements specifications regarding security requirements and they did not mention the behavior of the security requirements functions.

Sutcliffe et al. propose a verification method of scenarios based on validation-frame (Sutcliffe et al., 1998). This frame consists of situation part and requirements part. In situation part, pattern of events and actions are defined. In requirements one, some generic requirements are needed to handle each of these patterns. Using validation-frame, crosschecking between scenario and requirements is possible. Our approach is similar, but we enable to check (1) wrong sequence of events and (2) the number of occurrence of events. On contrast,

validation-frame does not check them.

6 CONCLUSIONS

The authors proposed a scenario checking method with rules based on the security evaluation criteria. We can specify the occurrence times of events and/or the time sequence among events as rules. Both scenario and rules can be transformed into the internal representation so that we can check scenario with rules and evaluate the correctness of one particular observed scenario.

The proposed method was demonstrated by the example and was evaluated. The evaluation results show that errors (the lack of events, extra events, the wrong sequence among events, and wrong behaviors against the security common criteria) in scenario can be effectively detected by checking the scenario with rules. By using this correctness checking method, we can get a scenario that satisfies security common criteria more effectively in system development.

ACKNOWLEDGEMENTS

The authors thank to Mr. Tatsuya Toyama and Mr. Kenta Nishiyuki for their contributions to the research.

REFERENCES

Alexander, I. F., and Maiden, N., 2004. “Scenarios, Stories, Use Cases – Through the Systems Development Life-Cycle”, John Wiley & Sons.

Barish, R, 1997. ACM Conference Committee Job Description, Conference Manual, Section No. 6.1.1, http://www.acm.org/sig_volunteer_info/conference_manual/6-1-1PC.HTM.

Carroll, J.M., 2000. “Making Use: Scenario-based Design of Human Computer Interactions”, MIT Press.

Cockburn, A., 2001. “Writing Effective Use Cases”, Addison Wesley, USA.

Fillmore, C. J., 1968. “The Case for Case”, Universals in Linguistic Theory, ed. Bach & Harms, Holt, Rinehart and Winston Publishing, Chicago.

“IEEE Std. 830-1998, 1998.” IEEE Recommended Practice for Software Requirements Specifications.

“ISO/IEC 15408 common criteria, 2005.”

McDermott, J. and Fox, C., 1999. “Using Abuse Case Models for Security Requirements Analysis”, Proceedings of the 15th IEEE Annual Computer Security Applications Conference (ACSAC’99), pp. 55-65.

Ohnishi, A., 1996. “Software requirements specification database based on requirements frame model”, Proceedings of the Second IEEE International Conference on Requirements Engineering (ICRE’96), pp. 221-228.

Ohnishi, A., Potts, C. 2001. Grounding Scenarios in Frame-Based Action Semantics, Proc. of 7th International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ’01), Interlaken, Switzerland, June 4-5, pp.177-182.

Railway Information System Co., Ltd., 2001. JR System, http://www.jrs.co.jp/keiki/en/index_main.html.

Schneier, B., 2001. Secrets & Lies Digital Security in a Networked World, John Wiley & Sons.

Sindre, G. and Opdahl, A. L., 2005. “Eliciting security requirements with misuse cases,” Requirements Engineering, Vol. 10, pp. 34-44.

Sutcliffe, A. G., Maiden, N. A. M., Minocha S., Manuel D., 1998. Supporting Scenario-Based Requirements Engineering, IEEE Trans. Software Engineering, Vol.24, No.12, pp.1072-1088.

Toval, A., Nicolaus, J. Moros, B. and Gracia, F., 2002. Requirements Reuse for Improving Information Systems Security: A Practitioner’s Approach, Requirements Engineering, Vol. 6, No. 4, pp. 205-219.

Toyama, T., Ohnishi, A., 2005. Rule-based Verification of Scenarios with Pre-conditions and Post-conditions, Proc. Of the 13th IEEE International Conference on Requirements Engineering (RE’05), Paris, France, pp.319-328.

Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P., 1998. Scenarios in System Development: Current Practice, IEEE Software, Vol.15, No.2, pp.34-45.

Zhang, H. and Ohnishi, A., 2004. “Transformation between Scenarios from Different Viewpoints”, IEICE Transactions on Information and Systems, Vol. E87-D, No. 4, pp. 801-810.