IEICE TRANS. INF. & SYST., VOL.E95–D, NO.4 APRIL 2012

PAPER Special Section on Knowledge-Based Software Engineering

Scenario Generation Using Differential Scenario Information

Masayuki MAKINO[†*], *Nonmember and* **Atsushi OHNISHI**[†a)], *Member*

SUMMARY A method of generating scenarios using differential scenaro information is presented. Behaviors of normal scenarios of similar purpose are quite similar each other, while actors and data in scenarios are different among these scenarios. We derive the differential information between them and apply the differential information to generate new alternative/exceptional scenarios. Our method will be illustrated with examples. This paper describes (1) a language for describing scenarios based on a simple case grammar of actions, (2) introduction of the differential scenario, and (3) method and examples of scenario generation using the differential scenario.

key words: *scenario generation, scenario language, differential scenario, scenario-based requirements elicitation*

1. Introduction

Scenarios are important in software development [5], particularly in requirements engineering by providing concrete system description [16], [18]. Especially, scenarios are useful in defining system behaviors by system developers and validating the requirements by customers. In scenario-based software development, incorrect scenarios will have a negative impact on the overall system development process. However scenarios are usually informal and it is difficult to verify the correctness of them. Errors in incorrect scenarios may include (1) vague representations, (2) lack of necessary events, (3) extra events, and (4) wrong sequence among events.

The authors have developed a scenario language named SCEL for describing scenarios in which simple action traces are embellished to include typed frames based on a simple case grammar of actions and for describing the sequence among events [17], [19]. Since this language is a controlled language, the vagueness of the scenario written with SCEL language can be reduced. Furthermore, a scenario with SCEL can be transformed into internal representation. In the transformation, both the lack of cases and the illegal usage of noun types can be detected, and concrete words will be assigned to pronouns and omitted indispensable cases [14]. As a result, the scenario with SCEL can avoid the errors typed 1 previously mentioned.

Scenarios can be classified into (1) normal scenarios, (2) alternative scenarios, and (3) exceptional scenarios. A normal one represents the normal and typical behavior of the target system, while an alternative one represents normal but alternative behavior of the system and an exceptional one represents abnormal behavior of the system. There are many normal scenarios for a certain system. For example, a normal scenario represents withdrawal, another normal scenario represents money deposit, another one represents wire transfer, and so on. Each normal scenarios has several alternative scenarios and exdceptional scenarios. In order to grasp whole behaviors of the system, not only normal scenarios, but also alternative/ exceptional scenarios should be specified. However it is difficult to hit upon alternative scenarios and exceptional scenarios, whereas it is easy to think of normal scenarios.

This paper focuses on automatic generation of alternative/exceptional scenarios from normal scenarios of two different systems belonging to the same problem domain. We adopt the SCEL language for writing scenarios, because the SCEL is a control language and it is easy to analyze scenarios written with the SCEL.

Suppose a scenario of purchasing a train ticket. One scenario may consist of just one event of buying a train ticket. Another scenario may consists of several events, such as 1) informing date, destination, and the number of passengers, class of cars, 2) retrieving train data base, 3) issuing a ticket, 4) charging ticket fee to a credit card, and so on. If the abstract levels of scenarios are different, it is quite difficult to correctly compare and analyze events of scenarios.

SCEL language for writing scenarios solves this problem, because SCEL provides a limited actions and their case structure as described in Sect. 2 and scenarios with SCEL keep a certain abstract level of actions.

2. Scenario Language

2.1 Outline

The SCEL language has already been introduced [19]. 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 the system for accomplishing their goals. We assume that each event has just one verb, and that each verb has its own case structure [8]. The scenario language has been developed based on this concept. Verbs and their own case structures depend on problem domains, but the roles of cases are independent

Manuscript received June 29, 2011.
Manuscript revised October 18, 2011.

[†]The authors are with the Department of Computer Science, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan.
[*]Presently, with JSOL Corporation
a) E-mail: ohnishi@selab.is.ritsumei.ac.jp

DOI: 10.1587/transinf.E95.D.1044

of problem domains. The roles include agent, object, recipient, instrument, source, etc. [8], [14]. Verbs and their case structures are provided in a dictionary of verbs. If a scenario describer needs to use a new verb, he can use it by adding the verb and its case structure in the dictionary.

We adopt a requirements frame in which verbs and their own case structures are specified. The requirements frame depends on problem domains. Each action has its case structure, and each event can be automatically transformed into internal representation based on the 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 [14].

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.

2.2 Scenario Example

Figure 1 shows a scenario of reservation of a hotel room written with our scenario language. A title of the scenario is given at the first line of the scenario in Fig. 1. Viewpoints of the scenario are specified at the second line. In this paper, viewpoints mean active objects such as human, system appearing in the scenario. There exist two viewpoints, namely "user" and "reservation system." The order of the specified viewpoints means the priority of the viewpoints. In this example, the first prior object is "user," and the second is "reservation system." In such a case, the prior object becomes a subject of an event.

In this scenario, all of the events are sequential. Actually, event number is for reader's convenience and not necessary.

2.3 Analysis of Events

Each event is automatically transformed into internal representation. For example, the 1st event "A user enters his membership number and his name to the reservation system" can be transformed into internal representation shown in Table 1.

In this event, the verb "enter" corresponds to the concept "data flow." The data flow concept has its own case structure with four cases, namely to say, source case, goal case, object case and instrument case. Sender corresponds to the source case and receiver corresponds to the goal case. Data transferred from source case to goal case corresponds to the object case. Device for sending data corresponds to the instrument case. In this event, "membership number and name" correspond to the object case and "user" corresponds to the source case.

The internal representation is independent of surface representation of the event. Suppose other representations of the event, "the reservation system receives user's membership number and his name from a user" and "User's membership number and his name are sent to the reservation system by a user." These events are syntactically different but semantically same as the 1st event. These two events can be automatically transformed into the same internal representations as shown in Table 1.

[Title: Reservation of a hotel room]
[Viewpoints: user, reservation system]
1.A user enters his membership number and his name to the reservation system.
2.The system validates the user with the membership number and the name.
3.The user enters retrieval information to the system.
4.The system retrieves available hotels from the database using the information.
5.The system shows available hotels to the user.
6.The user selects a hotel from the available hotels.
7. The system shows the room rate to the user.
8.The user enters the credit card number to the system.
9.The system asks the status of the card to a credit card company using the card number.
10.The system shows the reservation number to the user.

Fig. 1 Scenario example.

Table 1 Internal representation of the 1st event.

Concept: Data Flow

source	goal	object	instrument
user	reservation system	membership number and name	*NOT specified*

3. Differential Information of Scenarios

Systems that are designed for similar purpose (e.g. reservation, shopping, authentication, etc) often have similar behaviors. Besides, if such systems belong to the same domain, actors and data resemble each other. In other words, normal scenarios of similar purpose belonging to the same domain resemble each other. Since our scenario language provides limited vocabulary and limited grammar, the abstraction level of any scenarios becomes almost the same.

For one system, there exist several normal scenarios. In the case of ticket reservation, reservation can be written as a normal scenario and cancellation can be written as another normal scenario. For a certain normal scenario, there are several exceptional scenarios and alternative scenarios. To make a differential scenario, we select two normal scenarios of two different systems. Each of the two scenarios should represent almost the same purpose, such as reservation of some item.

The differential scenario consists of (1) a list of not corresponding words, (2) a list of not corresponding events, that is, deleted events which appear in one scenario (say, scenario A) and do not appear in the other (say, scenario B) and added events which do not appear in scenario A and appear in scenario B. We also provide (3) a list of corresponding words and (4) a list of corresponding events, and (5) a script

[Title: Reservation of a meeting room]
[Viewpoints: citizen, system]
1.A citizen enters reservation information to the system.
2. The system retrieves available room from the database using the information.
3.The system shows an available room to the citizen.
4. The citizen enters his name and telephone number to the system.
5.The system validates the citizen with the name and the telephone number.
6.The system shows the room rate to the citizen.
7.The citizen pays the rate to the system.
8.The system issues a receipt to the citizen.
9.The system shows the room number to the citizen.

Fig. 2 Normal scenario of reservation of a meeting room.

to apply the above differential information for generating scenarios.

We generally assume that one to one correspondence between two nouns and one to one correspondence between two events. Figure 2 shows a scenario of reservation of meeting room for residents in a city.

We compare the scenario of Fig. 1 with the scenario of Fig. 2 from top to bottom. First we check the actors specified as viewpoints of the two scenarios. In the case of scenarios of Fig. 1 and 2, "user" in Fig. 1 corresponds to "citizen" in Fig. 2 and "reservation system" in Fig. 1 corresponds to "system" in Fig. 2. The correspondence should be confirmed by user.

Second we check the action concepts of events. If there exist events whose action concept appears once in scenario A and B, respectively, we assume that these two events are probably corresponding to each other. For example, the concept of the 2nd event in Fig. 1 and the concept of the 5th event in Fig. 2 are "validate" and there are no more events whose concepts are "validate," so we regard these two events are probably corresponding to each other. Then we provide these two events to a user and the user will confirm that these two events are corresponding to each other by checking whether nouns of the same cases are corresponding or not.

If there exists an event whose action concept appears once in scenario A, but there exists two or more events of the action concept in scenario B, then we regard that one of the events of the concept in scenario B corresponds to the event in scenario A. So, we provide these events to system user and the user will check the corresponding events.

If there are two or more events whose concepts are same in two scenarios respectively, these events are candidates of corresponding events. Then we check that nouns of the same cases are corresponding to. Next we provide candidates to the user and he will select the corresponding event.

The first four events of the scenario in Fig. 1 can be transformed as shown in Table 2. The internal representations of the first five events of the scenario in Fig. 2 are shown in Table 3. In fact, the data flow concept has four cases, that is, source, goal, object, and instrument cases as shown in Table 1, but the instrument cases are omitted in Table 2 and 3 for the space limitation.

Table 2 The internal representation of the first four events of the scenario in Fig. 1.

concept	**agent/ source**	**goal**	**object**
data flow	user	reservation system	membership number and name
validate	*system*	*user*	*membership number and name*
data flow	user	reservation system	retrieval information
retrieve	system	available hotels	database

Table 3 The internal representation of the first five events of the scenario in Fig. 2.

concept	**agent/ source**	**goal**	**object**
data flow	citizen	system	reservation information
retrieve	system	available room	database
data flow	system	citizen	available rooms
data flow	citizen	system	name and telephone number
validate	*system*	*citizen*	*name and telephone number*

Table 4 A list of corresponding words between scenario A and scenario B.

Nouns in scenario A	**Nouns in scenario B**
user	citizen
reservation system	system
membership number and name	name and telephone number
available hotels	available room
retrieval information	reservation information
reservation number	room number
hotel room	meeting room
hotels	room

For the 2nd event in Table 2 and the 5th event in Table 3 as shown with italic font, since the nouns of the cases of the two events are same or corresponding to each other, these two events are corresponding to each other. At this time we get "membership number and name" correspond to "name and telephone number." So, the 1st event in Fig. 1 corresponds to the 4th event in Fig. 2, because concepts are same and all of the nouns of corresponding cases are corresponding to each other.

Similarly we detect corresponding events and corresponding nouns. Table 4 shows a list of corresponding nouns. Figure 3 shows corresponding events of the two scenarios. In Fig. 3, two events connected by an arrow are corresponding to each other. Events without an arrow have no corresponding events. The successive corresponding events are grouped into an event block. The first two events in Fig. 1 are grouped into a block named a1. The block a1 corresponds to a block named b2 consisting of the 4th and the 5th events in Fig. 2.

Finally, we can get the differential scenario between hotel reservation and meeting room reservation shown in Ta-

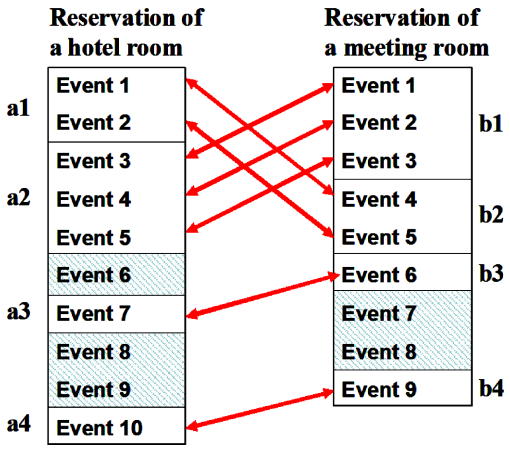

Fig. 3 Corresponding events.

Table 5 Deleted events from perspective scenario A/ Added events from perspective scenario B.

concept	agent/ source	goal	object
select	user	hotel	available hotels
data flow	user	system	credit card number
data flow	system	credit card company	credit card number

Table 6 Added events from perspective scenario A/ deleted events from perspective scenario B.

concept	agent/ source	goal	object
pay	citizen	system	room rate
data flow	system	citizen	receipt

```
1) change positions of block a1 and a2
2) delete events in Table 5
3) insert events in Table 6 followed by a4
4) change the corresponding nouns in Table 4
```

Fig. 4 Script to be applied to alternative/exceptional scenarios of scenario A.

ble 4, 5, and 6 and Fig. 3.

In order to apply the differential information to another scenario of reservation of a hotel room, we also provide a script shown in Fig. 4. Even if there exists a delete command in a script, event blocks will not be deleted when any event blocks in an applied scenario do not match with event blocks in the script. Even if there exists an insertion command in the script, event blocks will not be inserted when the following event block and the followed event block are missing in the applied scenario. Figure 5 shows a script applied to another scenario of reservation of a meeting room.

4. Scenario Generation Using Differential Scenario

Once the differential scenario between system A and B is given, we can apply it to another scenario of system A and get a new scenario of system B by changing corresponding words and by deleting or adding not-corresponding events.

```
1) change positions of block b1 and b2
2) delete events in Table 6
3) insert the first event in Table5 followed by b3
4) insert the last two events in Table 5 followed by b4
5) change the corresponding nouns in Table 4
```

Fig. 5 Script to be applied to alternative/exceptional scenarios of scenario B.

```
[Title: Reservation of a hotel room for aged users]
[Viewpoints: user, system]
1.A user enters his membership number and his name to the system.
2.The system validates the user with the membership number and the name.
3.The user enters retrieval information to the system.
4.The system retrieves available hotels from the database using the information.
5.The system shows available hotels to the user.
6.The user selects a hotel from the available hotels.
7.The system retrieves the date of birth of the user from the database using the membership number and the name.
8.The system checks the age of the user.
9.The system calculates the discount rate of the room for aged users.
10.The system shows the room rate to the user.
11.The user enters the credit card number to the system.
12.The system asks the status of the card to a credit card company using the card number.
13. The system shows the reservation number to the user.
```

Fig. 6 An alternative scenario.

In this section, we apply the differential scenario described in the previous chapter to an alternative scenario of hotel reservation and get an alternative scenario of meeting room reservation.

4.1 Examples of Generation

Figure 6 shows an alternative scenario of hotel reservation. In this scenario, an aged user reserves a hotel room with a discount rate. By applying the differential scenario in Table 4, 5, 6 and Fig. 3 using the application script in Fig. 4, we can get a new alternative scenario of reservation of a meeting room for aged citizen as shown in Fig. 7. In Fig. 7, words of boldface font are changed nouns according to a list of corresponding words in Table 4. Lastly, the generated scenario is investigated by the user. He can modify the generated scenario to eliminate errors.

4.2 Scenario Generator Using Differential Scenarios

Figure 8 shows the outline of the generation of scenarios using differential scenarios. We have been developing a prototype system based on the method. This system has been developed with C# on a Windows XP PC. The line of source code of the system is about 6,000. This system is a 4.5 man-month product.

Figure 9 shows the initial display image of the system. This system mainly provides two functions. One is

[Title: Reservation of a meeting room for aged citizen]
[Viewpoints: citizen, reservation system]
1.The citizen enters reservation information to the system.
2.The system retrieves available room from the database using the information.
3.The system shows available room to the citizen.
4. The citizen enters his name and telephone number to the system.
5.The system validates the citizen with the name and the telephone number.
6.The system retrieves the date of birth of the citizen from the database using the name and the phone number.
7.The system checks the age of the citizen.
8.The system calculates the discount rate of the room for aged citizen.
9.The system shows the room rate to the citizen.
10.The citizen pays the rate to the system.
11.The system issues a receipt to the citizen.
12.The system shows the room number to the citizen.

Fig. 7 A generated new alternative scenario.

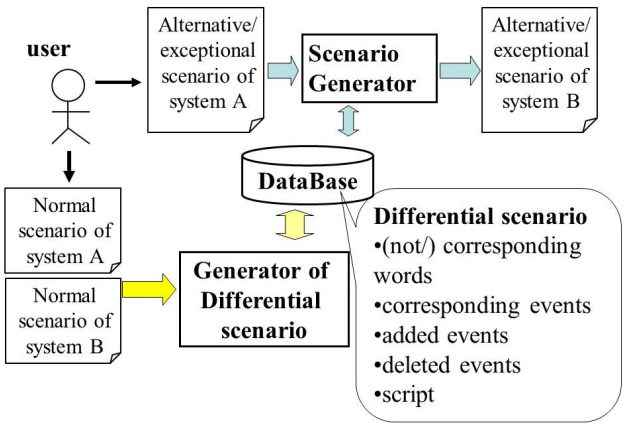

Fig. 8 Outline of scenario generation.

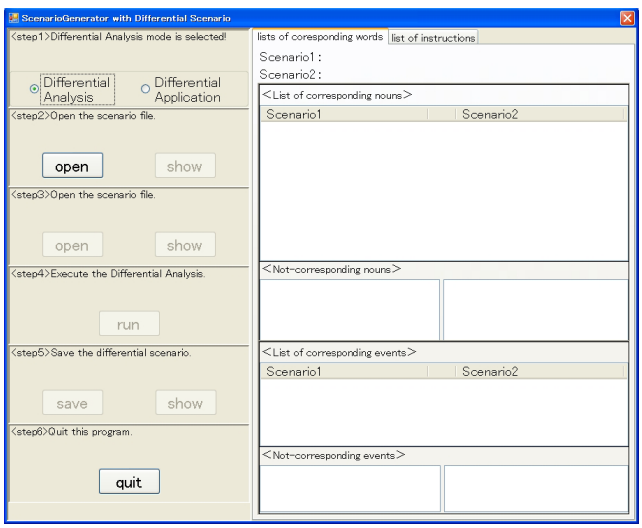

Fig. 9 Initial image of the system.

the derivation of the differential scenario between given two scenarios. The other is the application of the differential scenario to a specified scenario and the generation of a new

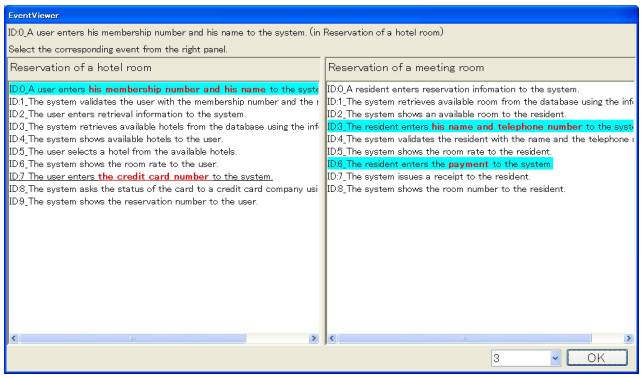

Fig. 10 Candidates of corresponding events.

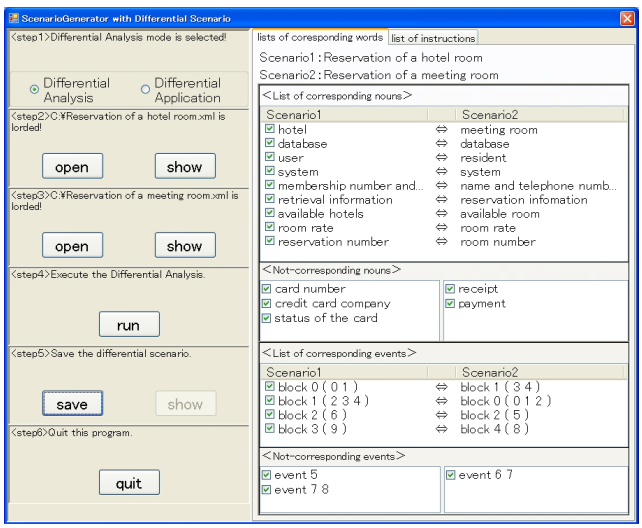

Fig. 11 Derivation of a differential scenario.

scenario. In Fig. 9, a user selects the former function and now he specifies two scenarios. These are scenario of the reservation of a hotel room and a scenario of the reservation of a meeting room. Then differential scenario between them is derived.

In Fig. 10, the user selects the corresponding event for the 1st event of the left-hand scenario. Two events are provided as candidates of corresponding events (the 4th event and the 7th event of the right-hand scenario). Since nouns with boldface font of the events are not registered in the list of corresponding words at that time, the user selects a corresponding event by specifying the id number of the event.

In this case, the user specifies the 4th event of the right-hand scenario as a corresponding event of the 1st event of the left-hand scenario by specifying the id number 3 in the bottom and right-side of the window in Fig. 10. The system automatically registers the correspondence between "membership number and name" of the left-hand scenario and "name and telephone number" of the right-hand scenario in the list of corresponding words. Likewise corresponding words and corresponding events will be determined and registered in the lists, respectively.

In Fig. 11, a list of corresponding words and a list of

corresponding events are displayed in the right-hand side of the window.

In Fig. 12, events of the left-hand scenario in Fig. 10 are blocked. There are 4 blocks numbered 0, 1, 2 and 3 respectively. Three events are not blocked and they do not have any corresponding events.

In Fig. 13, an application script is displayed. By applying this script to an exceptional/alternative scenario of the reservation of a hotel room, an exceptional/alternative scenario of the reservation of a meeting room will be derived as shown in Fig. 14.

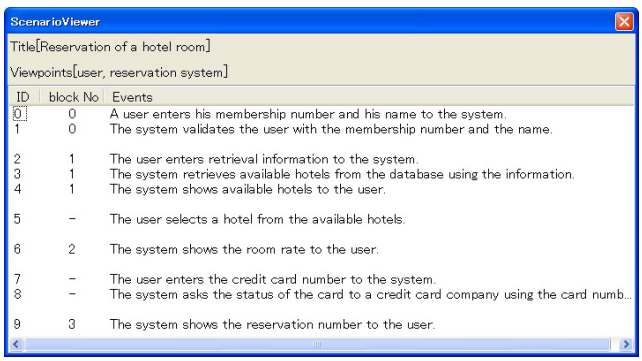

Fig. 12 Blocked events of the left scenario.

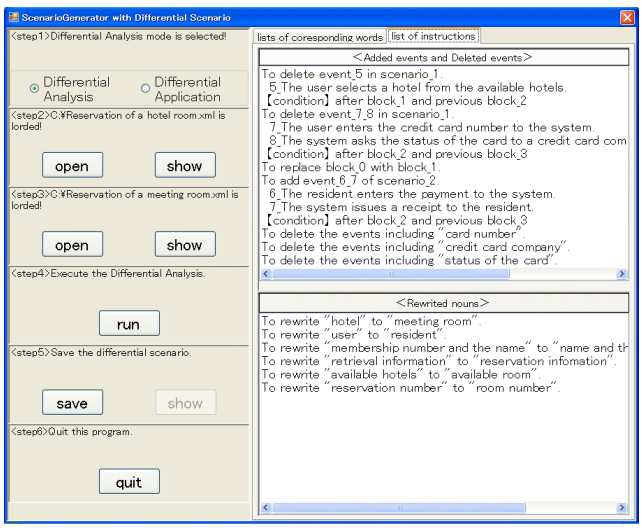

Fig. 13 Generated script.

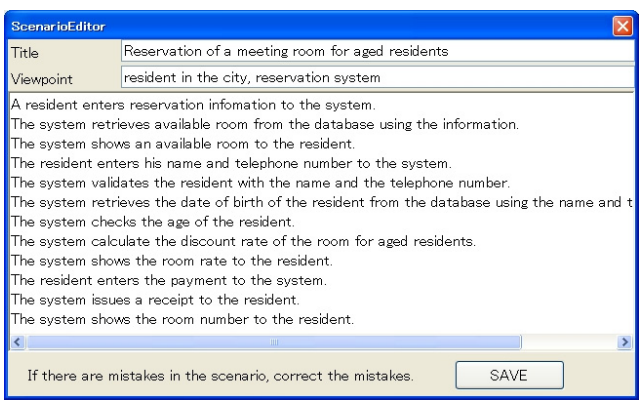

Fig. 14 Generated alternative scenario.

5. Experiment

In order to evaluate our method and system, we performed an experiment. The purposes of the experiment are to confirm the following benefits.

1. to lessen elaboration of writing scenarios
2. to make a scenario of high quality

5.1 Outline of the Experiment

8 students who are graduate students belonging to software engineering laboratory, Ritsumeikan university are divided into two groups of four subjects that named group A and B. Prior to the experiment, we explained scenario language and the way of scenario writing for two hours. We chose a rental system as problem domain. We also gave a job description of a rental system to provide domain knowledge to subjects.

Since the quality of generated scenarios depends on the ability of scenario writing and scenario analysis of subjects, we checked the ability of subjects prior to the experiment. We gave a normal scenario of borrowing a book at a library and asked to subjects to write a normal scenario of borrowing a CD at a CD rental shop. The result is shown in Table 7. A1, A2, A3, and A4 are members of group A, while B1, B2, B3, and B4 are members of group B. It took 17.6 minutes on average to write the scenario. The number of errors in a scenario of Group A is 2 on average, while the number of errors in a scenario of Group B is 0.5 on average. We confirmed that subjects' abilities of scenario writing and scenario analysis are different. The ability of Group A is less than that of Group B. This fact means that the quality of scenarios of Group A is usually less than that of Group B. We gave a correct scenario of borrowing a CD to all the menbers and pointed out the mistakes.

5.2 Generation vs. Description of Scenarios

We provided scenarios of a library system to the members of the two groups. These scenarios consist of 5 normal scenarios, and 2 exceptional scenarios. The member of group A wrote a normal scenario of borrowing a book and gets a

Table 7 Subjects' abilities of scenario analysis.

	Time (min.)	# of errors	# of events
A1	17	3	16
A2	17	0	19
A3	15	3	19
A4	13	2	17
B1	20	1	19
B2	19	0	19
B3	20	0	19
B4	20	0	19

Table 8 Scenarios of CD rental system.

id	title	the number of events
1	CD rental	19
2	CD rental failure by upper limitation	7
3	Return of CD	6
4	Retrun of CD with penalty	9
5	Retrieval of CDs	7
6	Registration of CDs	8
7	Registraion of a new member	16
8	Cancelation of a member	10

Table 9 Result of the experiment.

Scenario id	Group A		Group B	
	Time (min.)	errors	Time (min.)	errors
1	-	-	-	-
2	4	0	10	0
3	1	0	7	0
4	2	0	15	1
5	3	0	10	0
6	8	0	7	0
7	2	0	14	6
8	1	0	5	0
average except for the scenario 1	3.0	0	9.7	1.0

differential scenario between scenario of borrowing a book and a scenario of borrowing a CD. Then they get the scenarios of CD rental system automatically generated using our proposed method and system, while the members of group B wrote one or two scenarios of the CD rental system by themselves using corresponding scenarios of the library system. We checked generated scenarios of group A and written scenarios of group B by comparing correct scenarios with them.

Table 8 shows a list of scenarios of the CD rental system prepared as correct scenarios by the authors. Scenario id number 3, 5, 6, 7 and 8 are normal scenarios of the CD rental system, while a scenario of no.2 and 4 are exceptional scenarios.

Table 9 shows the result of experiment. It took extra 3.0 minutes on average to generate differential scenario for Group A. In using our method and system, scenarios are automatically generated, but the subjects need to check the generated scenarios. It took 3.0 minutes on average to check the scenarios. In checking none of the subjects found any errors in the generated scenarios. This means that our method and system generates exactly correct scenarios. In order to write scenarios by Group B, it took 9.7 minutes on average.

Actually, the ability of writing scenario of Group A is less than that of Group B, but the quality of generated scenarios by Group A is better than the quality of written scenarios by Group B as shown in Table 9.

Through the experiment, we found that our method and system improve the correctness of the scenario and lessen the writing time.

6. Related Works

There is an obvious trend to define scenarios as textual description of the designed system behaviors. The growing number of practitioners demanding for more "informality" in the requirements engineering process seems to confirm this trend. Most of these papers describe how to use scenarios for the elicitation [15] or exploration [9] of requirements. The authors believe that it is also important to support both the generation and the classification of scenarios

Ben Achour proposed guidance for correcting scenarios, based on a set of rules [1]. These rules aim at the clarification, completion and conceptualization of scenarios, and help the scenario author to improve the scenarios until an acceptable level in terms of the scenario models. Ben Achour's rules can only check whether the scenarios are well written according to the scenario models. We propose a method of generating exceptional scenarios and alternative scenarios from a normal scenario.

Neil Maiden et al. proposed classes of exceptions for use cases [11]. These classes are generic exceptions, permutations exceptions, permutation options, and problem exceptions. With these classes, alternative courses are generated. For communication actions, 5 problem exceptions are prepared, that is, human agents, machine agents, human-machine interactions, human-human communication, and machine-machine communication. They proposed a method of generating alternative paths for each normal sequence from exception types for events and generic requirements with abnormal patterns [3], [13], [15], [16]. Our approach for generating scenarios with a differential scenario is independent of problem domains.

Daniel Amyot et al. derive a scenario from use case map [2]. In order to generate several scenarios, they have to prepare several use case maps, while we have to prepare just one normal scenario with our approaches.

Christophe Damas et al. synthesize annotated behavior models from scenarios. They generate a state transition model from several scenarios and this model covers all scenario examples [6], [7]. However they cannot generate scenarios of different systems, while our approach enables to generate scenarios of different systems.

Yu-Chin Cheng et al. proposes a generation method of attack scenarios [4]. Using attack patterns, attack state transition model, attack scenarios can be generated. Their approach focuses on just attack scenarios via network, but we provide a generation method of exceptional scenarios and alternative scenarios.

7. Conclusion

We have developed a frame base scenario language and a method of generating scenarios using differential scenario. With our method, we can get new scenarios of a certain problem domain using scenarios belonging to the same domain and differential scenario between two systems.

Differential scenario is derivable from different systems' normal scenarios of similar purpose and applicable to alternative/exceptional scenarios of one of the system. However, such a differential scenario cannot be applicable to dif-

ferent normal scenarios' alternative/exceptional scenarios. In order to automatically determine the applicability of the different scenario, we have a plan to use pre-conditions and post-conditions specified in a scenario just like the selection of rules applicable to verify the correctness of scenarios [17].

Acknowledgment

This research is partly supported by Grant-in-Aid for Scientific Research (C)(2)(22500039).

References

[1] C.B. Achour, "Guiding scenario authoring," Proc. Eight European-Japanese Conference on Information Modeling and Knowledge Bases, pp.181–200, 1998.
[2] D. Amyot, D.Y. Cho, X. He, and Y. He, "Generating scenarios from use case map specifications," Proc. 3rd QSIC, pp.108–115, Dallas, USA, 2003.
[3] I. Alexander and N.A.M. Maiden, Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle, pp.161–177, John Wiley & Sons, 2004.
[4] Y.C. Cheng, C.H. Chen, C.C. Chiang, J.W. Wang, and C.S. Laih, "Generating attack scenarios with causal relationship," Proc. IEEE International Conference on Granular Computing (GRC2007), pp.368–373, 2007.
[5] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, USA, 2001.
[6] C. Damas, B. Lambeau, P. Dupont, and A. Lamsweerde, "Generating annotated behavior models from end-user scenarios," IEEE Trans. Softw. Eng., vol.31, no.12, pp.1056–1073, 2005.
[7] C. Damas, B. Lambeau, and A. Lamsweerde, "Scenarios, goals, and state machines: A win-win partnership for model synthesis," Proc. 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp.197–207, 2006.
[8] C.J. Fillmore, The Case for Case, in Universals in Linguistic Theory, Holt, Rinehart and Winston, 1968.
[9] J.C.S.P. Leite, G. Rossi, F. Balaguer, V. Maiorana, G. Kaplan, G. Hadad, and A. Oloveros, "Enhancing a requirements baseline with scenarios," Proc. 3rd RE, pp.44–53, 1997.
[10] N.A.M. Maiden, M.K. Manning, and M. Ryan, "CREWS-SAVRE: Systematic scenarios generation and use," Proc. 3rd ICRE, pp.148–155, 1998.
[11] N.A.M. Maiden and M. Hare, "Problem domain categories in requirements engineering," Int. J. Human-Computer Studies, vol.49, pp.281–304, 1998.
[12] N.A.M. Maiden and S. Robertson, "Developing use cases and scenarios in the requirements process," Proc. 27th ICSE, pp.561–570, 2005.
[13] A. Mavin and N.A.M. Maiden, "Determining socio-technical systems requirements: Experiences with generating and walking through scenarios," Proc. 11th IEEE RE, pp.213–222, 2003.
[14] A. Ohnishi, "Software requirements specification database on requirements frame model," Proc. IEEE 2nd ICRE, pp.221–228, 1996.
[15] A.G. Sutcliffe and M. Ryan, "Experience with SCRAM, a SCenario requirements analysis method," Proc. 3rd ICRE, pp.164–171, 1998.
[16] A.G. Sutcliffe, N.A.M. Maiden, S. Minocha, and D. Manuel, "Supporting scenario-based requirements engineering," IEEE Trans. Softw. Eng., vol.24, no.12, pp.1072–108, 1998.
[17] T. Toyama and A. Ohnishi, "Rule-based verification of scenarios with pre-conditions and post-conditions," Proc. 13th IEEE RE2005, pp.319–328, 2005.
[18] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer, "Scenarios in system development: Current practice," IEEE Softw., vol.15, no.2, pp.34–45, March/April 1998.
[19] H. Zhang and A. Ohnishi, "Transformation between scenarios from different viewpoints," IEICE Trans. Inf. & Syst., vol.E87-D, no.4, pp.801–810, April 2004.

Masayuki Makino was received B. of Engineering and M. of Engineering degrees from Ritsumeikan University in 2006 and 2008, respectively. Currently, he is at JSOL Corporation.

Atsushi Ohnishi was born in 1957. He received B. of Engineering, M. of Engineering, and Dr. of Engineering degrees from Kyoto University in 1979, 1981, and 1988, respectively. He was a Research Associate of Kyoto University from 1983 to 1989 and an Associate Professor of Kyoto University from 1989 to 1994. Since 1994 he has been a Professor at Dept. Computer Science, Ritsumeikan University. He was a visiting scientist at Dept. Computer Science, UC Santa Barbara, California, U.S.A. from 1990 to 1991 and also a visiting scientist at College of Computing, Georgia Institute of Technology, Georgia, U.S.A. in 2000. His current research interests include requirements engineering, object oriented analysis, and software design techniques. Dr. Ohnishi is a member of IEEE Computer Society, ACM, Information Processing Society (IPS) Japan, and Japan Society for Software Science and Technology (JSSST).