A Visualization Tool for Scenario-based Software Development

Eiji Shiota and Atsushi Ohnishi
Department of Computer Science, Ritsumeikan University, Kusatsu, Japan

Keywords: Scenario Retrieval, Differential Scnenario, Scenario-based Requirements Elicitation.

Abstract: In a scenario-based software development, a lot of scenarios should be described in order to clarify the whole behaviors of the target software. By reusing scenarios of similar software systems, it becomes more efficient to newly describe scenarios of the target software. A differential scenario includes the difference between sequences of events of the two scenarios and the difference between nouns in the scenarios. If the nouns of the two scenarios are commonly used in the two scenarios, we regard the two scenarios specify the same or similar system. If the sequences of the events of the two scenarios are corresponding each other, we regard behavior of the two scenarios are similar. In this paper, we derive differential information including different words and events from two scenarios. Then, we propose a method of scenario retrieval using differential information between two scenarios. This method enables to detect similar scenarios for a given scenario. The proposed method and a prototype system for creating and visualizing differential scenario and a retrieval system will be illustrated with examples.

1 INTRODUCTION

Scenarios are important in software development (Cockburn, 2001), particularly in requirements engineering by providing concrete system description (Sutcliffe et al., 1998; Weidenhaupt et al., 1998). Especially, scenarios are useful in defining system behaviors by system developers and validating the requirements by customers (Alexander et al., 2004; Mavin et al., 2003). In scenario-based software development, incorrect scenarios will have a negative impact on the overall system development process.

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 of a banking system, another normal scenario represents money deposit, a third one represents wire transfer, and so on. Each normal scenario has several alternative scenarios and exceptional scenarios. In order to grasp all 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. In order to improve the productivity of scenario-based software development, reusing of scenario is a key to solve the above problem.

This paper focuses on retrieval of scenarios in order to improve the reusability of scenarios. For example, suppose a new development of a train ticket reservation system. In case of scenario-based development, several normal scenarios, such as reservation, cancellation, modification and so on should be specified. Furthermore, alternative scenarios and exceptional scenarios should be specified. In such a case, if scenarios of similar system, such as flight ticket reservation are reusable and if developers can easily generate new scenarios of the new system, the productivity will be improved.

If we can detect existing scenarios whose behaviors are similar with a scenario of a system to be developed, we may reuse the detected scenarios. Suppose a new train ticket reservation system. If we find scenarios of similar system, such as bus ticket reservation, we may use them.

Retrieval with keywords is an easy way, however, it is difficult to search for full-text scenarios. One of the reasons of the difficulties is the difference of abstraction levels of scenarios. Suppose a scenario of purchasing a train ticket. One scenario may consist of

Shiota E. and Ohnishi A..
A Visualization Tool for Scenario-based Software Development.
DOI: 10.5220/0005504702930300
In *Proceedings of the 10th International Conference on Software Engineering and Applications* (ICSOFT-EA-2015), pages 293-300
ISBN: 978-989-758-114-4

just one event of buying a train ticket. Another scenario may consist 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 abstraction levels of scenarios are different, it is quite difficult to correctly retrieve scenarios.

SCEL language for writing scenarios solves this problem, because SCEL provides a limited actions and their case structure as described in Section 2.3 and scenarios with SCEL keep a certain abstraction level of actions.

2 SCENARIO LANGUAGE

2.1 Outline

The SCEL language has already been introduced (Zhang et al., 2004). 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 (Fillmore, 1968). 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 of problem domains. The roles include agent, object, recipient, instrument, source, etc. (Fillmore, 1968; Ohnishi, 1996). 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 (Ohnishi, 1996). 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, SCEL. A

[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.

Figure 1: Scenario example.

title of the scenario is given at the first line of the scenario in Figure 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.

Each event is automatically transformed into internal representation. The details of transformation mechanism is in (Ohnishi, 1996). 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.

Table 1: Internal representation of the 1st event.

Concept: Data Flow

source	goal	object	instrument
user	reservation system	membership no. and name	*NOT specified*

3 DIFFERENTIAL SCENARIO

Systems that are designed for a 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 a 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 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 Figure 1 with the scenario of Figure 2 from top to bottom. First, we check the actors specified as viewpoints of the two scenarios. In the case of scenarios of Figure 1 and 2, “user” in Figure 1 corresponds to “citizen” in Figure 2 and “reservation system” in Figure 1 corresponds to “system” in Figure 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

[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.

Figure 2: Normal scenario of reservation of a meeting room.

each other. For example, the concept of the 2nd event in Figure 1 and the concept of the 5th event in Figure 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 Figure 1 can be transformed as shown in Table 2. The internal representations of the first five events of the scenario in Figure 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.

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

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

concept	agent/ source	goal	object
data flow	user	reservation system	membership no. and name
validate	*system*	*user*	*membership no. 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 Figure 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 no.
validate	*system*	*citizen*	*name and telephone no.*

each other. At this time we get "membership number and name" correspond to "name and telephone number." So, the 1st event in Figure 1 corresponds to the 4th event in Figure 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 Figure 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 Figure 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 Figure 2.

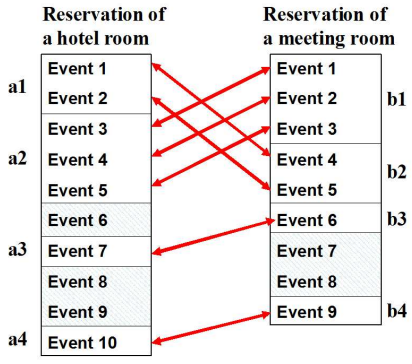

Figure 3: Corresponding events.

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 num. and name	name and telephone no.
available hotels	available room
retrieval information	reservation information
reservation number	room number
hotel room	meeting room
hotels	room

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

Finally, we can get the differential scenario between hotel reservation and meeting room reservation shown in Table 4, 5, and 6 and Figure 3. Table 5 shows events that appear in scenario A, but do not appear in scenario B. Table 6 shows events that do not appear in scenario A, but appear in scenario B. Figure 4 shows a script to make a new alternative/exceptional scenario of scenario B by applying to an alternative/exceptional scenario of scenario A.

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

Figure 4: Script applied to alternative/exceptional scenarios of scenario A.

4 SCENARIO RETRIEVAL USING DIFFERENTIAL SCENARIO

In scenario-based software development, several scenarios should be specified. Since such scenarios may be revised, there exist a lot of scenarios of different revisions. When a scenario is given, it may be difficult to find similar scenarios or related scenarios to

the given scenario by hand. We propose a retrieval method in order to get similar scenarios or related scenarios using the similar information of scenarios.

We assume that scenarios are analyzed based on the requirements frame in advance. As previously mentioned in Section 2, the requirements frame strongly depends on the problem domain. So, if case structures of verbs are different between two scenarios, we consider that these two scenarios are belonging to different domains each other. If all of the case structures are same, these scenarios can be classified into the same domain.

We propose two factors of the similarity between scenarios. One is related to same system. For example, a banking system provides several functions, withdrawal, deposit, loan, opening account, and so on. These functions are different each other, but both active objects, such as customer, bank clerk, ATM, banking system and inactive objects, such as bank card, cash, account in common appear in scenarios specifying behaviors of these functions of the banking system. The other factor is related to same or similar behavior. For example, behavior of train seat reservation and that of flight reservation are similar each other, although these systems are different.

4.1 Similarity of Scenarios by System

If same nouns are used in scenarios, these scenarios probably specify behaviors of the same system. For example, "library id," "e-library system," and "librarian" appear in different scenarios, these scenarios can be regarded as scenarios of the same system. On the basis of the above discussion, we give an equation in order to measure the similarity of system of scenarios as below.

$$\textit{Similarity of system between the two scenarios} = \frac{\textit{the no. of same nouns in events of the two scenarios}}{\textit{the total no. of nouns in events of the two scenarios}} \quad (1)$$

As for scenarios in Figure 1 and 2, nouns in the events of these scenarios are shown in Table 7.

The total number of the nouns is 19 and the same nouns are "database", "name" and "room rate." So the similarity of system between these two scenarios becomes $\frac{3}{19}$.

4.2 Similarity of Scenarios by Behavior

If scenario titles have a same verb, these scenario probably specify similar behaviors. For example, a scenario whose title is "a customer reserves a train seat" and another scenario whose title is "a user reserves a flight ticket" can be classified into similar scenarios from a behavioral viewpoint. However, a scenario whose title is "a customer purchases a train ticket" can be classified into similar scenarios with above ones. So, we think that scenarios are similar if titles of the scenarios have same verb, but this is not necessary.

Table 7: Nouns in the events of Figure 1 and Figure 2.

Scenario	nouns
Fig.1	available hotel(s), credit card company, (credit) card number, database, membership number, name, (retrieval) information, reservation number, (reservation) system, room rate, status of the card, user
Fig.2	available room, citizen, database, name, receipt, (reservation) information, room number, (room) rate, system, telephone number

Sequence of events in a scenario represents behaviors of users and system. If systems are different from each other, nouns in events become different, even if events specify similar behaviors. So, we use corresponding events in the differential scenario. If two scenarios are similar each other from the perspective of behavior, the ratio of corresponding events becomes high.

On the basis of the above discussions, we give the second equation in order to measure the similarity of behaviors of scenarios as shown in below.

$$\textit{Similarity of behavior between the two scenarios} = \frac{\textit{the number of corresponding events}}{\textit{the total number of events of the two scenarios}} \quad (2)$$

As shown in Figure 3, the total number of events is $10+9-7=12$ and the number of the same events is 7. So, the similarity of behavior between scenarios of Figure 1 and 2 is $\frac{7}{12}=0.58$ We consider that two scenarios whose similarity of behavior is greater than 0.5 are scenarios of similar behaviors.

In order to apply the differential information to another scenario of reservation of a hotel room, we also provide a script for application script shown in Figure 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.

4.3 Scenario Categorization

We map classified scenarios as shown in Figure 5. In Figure 5, there exist two axes. The horizontal axis means the similarity of behaviors, while the vertical axis means the similarity of systems. Scenarios of same vertical level are scenarios of similar behaviors and scenarios of same horizontal level are scenarios of same system. Overlapped scenarios mean that these scenarios have similar behaviors and belong to same system. The three scenarios of the left-side are categorized into scenarios of similar behavior, that is, reservation. Scenario of modifying flight ticket and its exceptional scenario are piled up, since these two scenarios belong to same system and have similar behaviors.

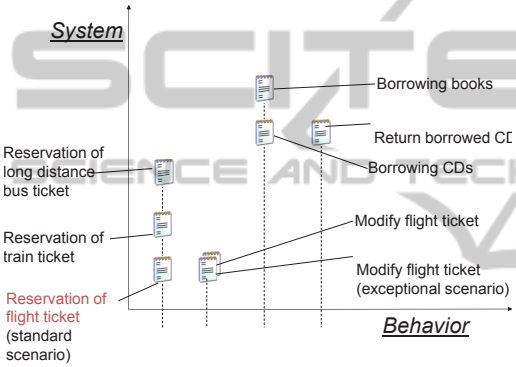

Figure 5: Retrieval result.

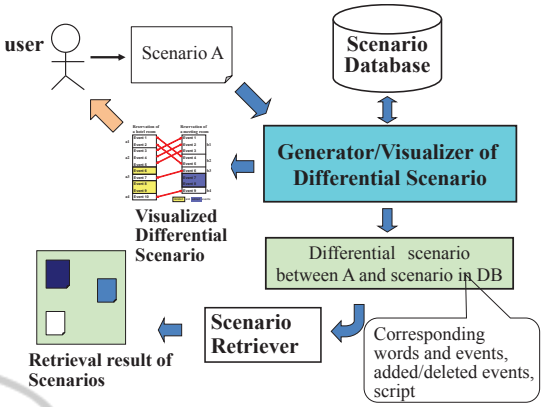

Figure 6: Outline of Scenario Visualization.

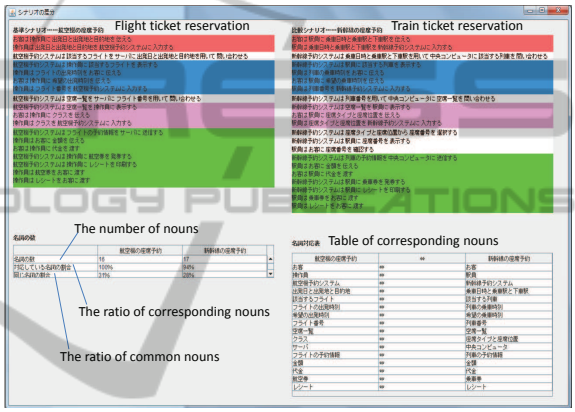

Figure 7: Visualized Differential Scenario between flight ticket reservation and train ticket reservation.

5 VISUALIZATION TOOL

We have been developing a prototype system based on the proposed method with Java on an Eclipse 3.6 Helios. This system is three man-month product and the number of source code line is about 3,000. Figure 6 shows the outline of the visualization of differential scenario. A user specifies a normal scenario of a new system. Our tool generates differential scenarios between the specified scenario and each scenario stored in a scenario database. If there exist 100 scenarios in the scenario database, 100 differential scenarios will be generated. Out tool can visualize each differential scenario. Another tool named scenario retriever generates scenario map with two axes, that is, the similarity of behavior and the similarity of system. With this map, the user can find similar scenarios to the specified scenario.

Figure 7 shows a differential scenario between a scenario of reserving a flight ticket and a scenario of reserving a train ticket. The same colored events of the two scenarios are corresponding to each other. There exist four corresponding blocks between the two scnearios. In the lower area of this figure, a table of the number of nouns in the two scenarios and a table of corresponding nous between the two scenarios are shown.

Figure 8 shows a scenario similarity map. A normal scenario of flight ticket reservation is located in the origin (left and bottom.) The horizontal axis means the similarity of system, while the vertical axis means the similarity of behavior. Similar scenarios to the flight ticket reservation in terms of system are displayed in the lower and bottom area. Similar scenarios in terms of behavior are displayed in the left area. With this map user can easily find similar scenarios to the specified scenario. For readers' convenience, the authors translated some Japanese messages of this map into English.

6 EXPERIMENT

To evaluate our method, we compare the classification of scenarios by hands with the classification

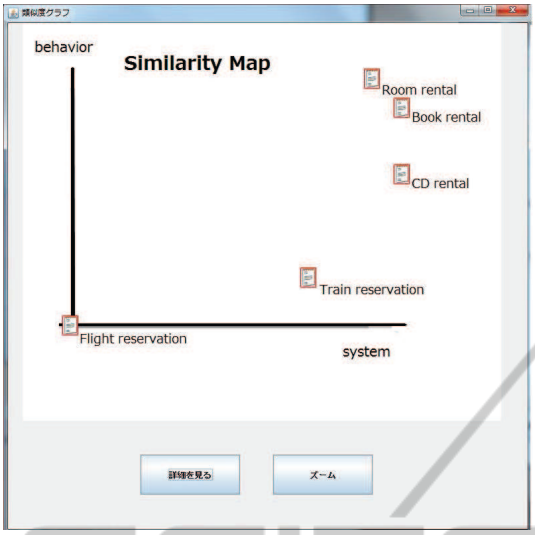

Figure 8: Visualized Scenario Retrieval Result.

Table 8: The scenario classification by proposed method and by students.

Scenario	by authors (same as by method)	by students
Train ticket reservation via internet	Different system, similar behavior	11/13
Flight ticket changing	Same system, different behavior	11/13
Flight ticket changing 2	Same system different behavior	11/13
Train ticket reservation	Different System, similar behavior	11/13
Flight ticket reservation	Same system, similar behavior	13/13
Bus ticket reservation	Different System, similar behavior	10/13
Claim for the loss on insurance	Different problem domain	13/13
Goods purchasing	Different problem domain	12/13

by the method. Thirteen graduate students at Computer Science department who well know both the scenario language and the problem domain classify eight scenarios for a standard scenario, while the same scenarios are also classified based on the proposed method. The scenario of reservation of flight ticket was adopted as a standard scenario in this experiment. Table 8 shows the comparison of the scenario classifications.

In this experiment, nine scenarios are classified in advance by the authors. The results by the authors are regarded as correct classifications. The classification results by our method are much the same as the result by the authors. This fact means our method can correctly classifyscenarios. The values of the column "result by students" mean the ratio of correct classification by the students. We investigated the reason why some students wrongly classified and found that they did not recognize the difference of systems correctly. After giving additional explanation of systems, the students adopted same classification of scenarios as classified by the proposed method.

Through the experiment, we found that differential scenario is useful to classify scenarios.

7 RELATED WORK

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 (Sutcliffe et al., 1998) or exploration (Leite et al., 1997) of requirements. The authors believe that it is also important to support both the generation and the classification of scenarios.

Brian Chance et al. give scenario types for the classification (Chance et al., 1999). However they do not propose how to classify given scenarios. Our approach for retrieving scenarios gives how to retrieve similar scenarios with a given scenario.

Colette Roland et al. give a classification framework of scenarios in CREWS project (Rolland et al., 1996). They classify scenarios from four viewpoints. These are contents, purpose, form and lifecycle. They can define the content of a whole scenario. We define contents of a scenario the sequence of actions specified in events and can derive contents of a part of a scenario. In this sense, we can detect similarities between fragments of two different scenarios.

Zhang Wei-ha et al. propose a classification method of scenarios, but their method depends on hazard scenarios and classified into five types in accordance with the complexity (Zhang et al., 2008). Gerrit Lahrmann et al. propose a classification method of scenarios, but their method depends on information logistics scenarios based on three types and 4 factors (Lahrmann et al., 2009). Our method classifies more general scenarios and enables to retrieve similar scenarios.

Martin Glinz proposes a very lightweight requirements modeling language (Glinz, 2010). With this language relations among objects including human, system and data can be visualized. The purpose of visualizing is modeling requirements at an early stage. In (Fill et al., 2013), a meta modeling platform named ADOxx is proposed. This platform provides the visual representation of modeling methods from the

area of requirements engineering. The purposes are both meta modeling and modeling a software. In case of our method, the purpose of visualizing is to improve the understandability of differential scenario and retrieval result of similar scenarios.

In authors' previous works, generation methods of exceptional scenarios and alternative scenarios with a normal scenario have been established. By retrieving a similar normal scenario using our proposed method and by applying the generation methods, we can easily get alternative/exceptional scenarios (Makino et al., 2012).

8 CONCLUSION AND FUTURE WORK

We have developed a frame base scenario language and a method of generating and visualizing differential scenario between two scenarios. We have established a retrieval method of similar scenarios with system/behavior for a given scenario using the differential scenario and a generation method of alternative/exceptional scenarios for a given scenario using the differential scenario. We have also developed visualization tools of differential scenario and retreival result of similar scenarios. The effectiveness of the differential scenarios is validated through the experiment.

In order to retrieve more efficiently similar scenarios with differential scenario, using pre-conditions and post-conditions just like the selection of rules applicable to verify the correctness of scenarios (Toyama et al., 2005) is left as our future work.

REFERENCES

Alexander, I. and Maiden, N. A. M. Scenarios, Stories, Use Cases, Through the Systems Development Life-Cycle. *John Wiley & Sons, Ltd.*, 2004, pp.161-177.

Chance, B. D. and Belhart, B. E., A Taxonomy for Scenario Use in Requirements Elicitation and Analysis of Software Systems. *Proc. IEEE ECBS'99*, 1999, pp.232-238.

Cockburn, A., Writing Effective Use Cases. *Addison-Wesley*, USA, 2001

Fill, H. G. and Karagiannis, D., On the Conceptualisation of Modelling Methods Using the ADOxx Meta Modelling Platform. *International Journal of Enterprise Modelling and Information Systems Architectures*, Vol.8, No.1, 2013, pp.4-25.

Fillmore, C. J., The Case for Case, in Universals in Linguistic Theory. *Holt, Rinehart and Winston*, 1968.

Glinz, M., Very Lightweight Requirements Modeling. *Proc. 18th RE10*, pp.385-386, 2010.

Lahrmann, G. and Stroh, F., Towards a Classification of Information Logistics Scenarios - An Exploratory Analysis. *Proc. IEEE 42nd Hawaii International Conference on System Sciences*, 2009, pp.1-10.

Leite, J. C. S. P. et.al., Enhancing a Requirements Baseline with Scenarios. *Proc. 3rd RE*, 1997, pp.44-53.

Makino M. and A. Ohnishi, A., Scenario Generation Using Differentail Acenario Information. *IEICE Trans. Inf. & Syst.*, Vol.E95-D, No.4, 2012, pp.1044-1051.

Mavin A. and Maiden, N. A. M., Determining socio-technical systems requirements, experiences with generating and walking through scenarios. *Proc. 11th IEEE RE*, 2003, pp.213-222.

Ohnishi, A., Software Requirements Specification Database on Requirements Frame Model. *Proc. IEEE 2nd ICRE*, 1996, pp.221-228.

Rolland, C. et al. A Proposal for a Scenario Classification Framework. *CREWS Report* 96-01, 1996.

Sutcliffe, A. G. and Ryan, M., Experience with SCRAM, a SCenario Requirements Analysis Method. *Proc. 3rd ICRE*, 1998, pp.164-171.

Toyama, T. and Ohnishi, A., Rule-based Verification of Scenarios with Pre-conditions and Post-conditions. *Proc. 13th IEEE RE2005*, 2005, pp.319-328.

Weidenhaupt, K. et al., Scenarios in System Development, Current Practice. *IEEE Software*, March, 1998, pp.34-45.

Zhang, H. and Ohnishi, A., Transformation between Scenarios from Different Viewpoints. *IEICE Trans. Inf, & Syst.*, Vol.E87-D, No.4, 2004, pp.801-810.

Zhang, W. et al.: Classification of Hazard Scenario and SDG Qualitative Identification Method. *Proc. the 7th ICSC2008*, 2008, pp.1223-1227.