Requirements Exploration by Comparing and Combining Models of Different Information Systems

Haruhiko Kaiya\textsuperscript{1}, Kazuhiro Adachi\textsuperscript{1}, Yoshihide Chubachi\textsuperscript{2}

\textsuperscript{1} Faculty of Science, Kanagawa University, Hiratsuka 259-1293, Japan
\textsuperscript{2} Advanced Institute of Industrial Technology, Tokyo 140-0011, Japan

Abstract. A people or an organization participates in several different activities simultaneously. Some activities are already supported by information systems, and others are not yet supported but supposed to be. In this paper, we proposed and exemplified a method for exploring the efficiency and synergy of the systems by comparing and combining system elements. We use any modeling languages such as use case diagrams and data flow diagrams for this comparison and combination. For the efficiency, elements in a model are shared or substituted by/for those in another. For the synergy, elements in several models are combined so as to resolve the problems of stakeholder. To validate and discuss our method, we applied it to three sets of activities: room and seat booking, driving and taxi, and rainy report and wiper control. Three different notations are used for each case: use case diagrams, activity diagrams and data flow diagrams.

Keywords: Requirements elicitation, Systems of Systems, Modeling languages

1 Introduction

Each people or organization normally participates in several different activities simultaneously. For example, some faculty member is writing technical papers while he is also preparing teaching materials in class rooms and attending administrational meeting in his faculty. Some activities have been already supported by information systems, but others have not. When we are going to introduce new information systems into latter type of activities, we have to take existing systems into account. However, we do not explicitly take them into account. Requirements engineers simply perform requirements elicitation from stakeholders. They can take the existing systems into account indirectly when the stakeholders states other activities and existing systems related to the activities.

Even when all activities are already supported by some information systems, it is valuable to analyze their relationships. Such analysis sometimes brings the efficiency and synergy of such systems. In some cases, features of the systems such as functions and data are duplicated among several systems. When we share such features, we can save both logical resources such as source codes or data and physical resources such as hardware devices. Even when some features of a system are not directly related to other features of other systems, the combination of such features sometimes lets us know requirements of which stakeholders were unaware. For example, a function of a system...
supporting technical writing may contribute to teaching preparation so that its teaching materials are always state of the art.

To summarize, we have the following research questions (RQs).

– RQ1: How to share elements among different systems or activities for efficiency?
– RQ2: How to substitute an element in a system or an activity for another element in another system or another activity?
– RQ3: How to discover unaware requirements by examining different systems and activities?

To answer these questions, we propose a requirements elicitation method. In the method, we first describe models of both existing and developing systems respectively. We then compare elements in a model with others in another model to find model elements which can be shared or substituted. We also examine the combination of such elements to discover requirements of which stakeholders were unaware. To examine such comparison and combination comprehensively, we use any kind of modeling languages. We use three different notations in this paper: use case diagrams, activity diagrams and data flow diagrams.

There already exists a research area called Systems of Systems (SoS) [1]. SoS researches normally focus on the relationships among a huge number of systems, and do not focus on specific comparison and combination of systems’ features. Our research focuses on a small number of systems because the number of activities where a person or an organization can participate is normal not so huge, i.e. around seven.

The rest of this paper is organized as follows. In the next section, we explain our requirements elicitation method by using use case diagrams. In section 3, we show other examples using activity diagrams and data flow diagrams respectively. We also discuss examples and the method itself. We then briefly review related works in section 4. Finally, we summarize our current results and show our future issues.

2 Method for eliciting requirements for systems together

Our requirements elicitation method consists of the following seven steps.

1. Identify stakeholders, hardware devices and physical phenomena. We call them actors.
2. Each actor usually participates in several different activities. Identify such activities.
3. Describe a model for each activity. If it already contains information systems, the systems of course should be described in the model. Information systems to be developed may be also described if they are planed.
4. Find the same or similar elements among models of several activities.
5. Share the elements among several activities.
6. Substitute existing elements for those to be developed if they are the same or similar.
7. Identify problems of actors which are not resolved. Discover the solution of the problems by combining elements in all activities.

We explain the steps above respectively by using an example.
Fig. 1. Use cases of booking a room in a hotel and a seat of a train

Fig. 2. Sharing elements and discovering new elements of room and seat booking
2.1 Step 1: Identify actors

We focus on a person who wants to book a hotel room. A hotel guest is focused here. The left hand side in Figure 1 shows the use case diagram of this activity.

2.2 Step 2: Identify other activities

The person usually has to travel to the hotel by some transportation systems. We thus focus on the activity to book the seat of a transportation system. The right hand side in Figure 1 shows its use case diagram. For simplicity, we assume the person will board a train for his/her travel. Although we have identified other activities such as checking tourist resorts or booking some activities such as diving or spa, we only focus on two activities here.

2.3 Step 3: Describe models

Use case models are already described in Figure 1. We may use other models such as class diagrams, state machines or goal models if we cannot find similarity among the models.

2.4 Step 4: Find the same or similar elements

We found two same use cases in these two diagrams: “Specify date”, “Make payment”. We regard “Choose a hotel” in room booking is similar to “Specify destination” in seat booking because the destination (a train station in this case) is almost fixed according to the chosen hotel. We also regard “Specify address” in room booking is similar to “Specify origination” in seat booking because the origination is usually a station close to the guest’s house.

2.5 Step 5: Share the elements

The same use cases in step 4 are simply shared as shown in Figure 2. We make include-relationships (or use-relationship) from similar use cases in seat booking to those in room booking.

2.6 Step 6: Substitute elements

In this case, there is no substitution because no use cases (functions) in a use case diagram cover the functionalities of use cases in another. We mentioned this issue in an example in section 3.2.

2.7 Step 7: Discover new solutions

Hotel staffs usually participate in this activity even if a booking system (booking web site) exists. No-show is one of the big problems in the booking management of hotel staffs. If both a room and a seat are booked together, hotel staffs can be confident of guest arrival. This new use case “confirm booking” may resolve this problem, and gives peace of hotel staffs’ mind. For the customers, the combined system can give some discount because the threat to no-show decreases.
**Fig. 3.** Activity diagram of a car navigation system

**Fig. 4.** Activity diagram of an application for calling a taxi
3 Examples

In this section, we show two distinct examples of applying our method to demonstrate the method works. We also discuss our future works on the basis of these examples.

3.1 Driving Navigation and Taxi Call

In this example, we focus on the following two systems, and try to share some actions in the systems.

- Car navigation system for driving (Figure 3)
- Application for calling a taxi (Figure 4)

We use activity diagrams because we assume these systems can share several actions so that they can be merged.

As shown in Figure 3, a driver first tells his/her destination to his/her navigation system. The system then tells the direction to go. The driver drives his/her car while he/she checks the navigation by the system. The navigation is given by audio and/or visual messages, but such issue is omitted in this diagram for simplicity.

In the context of an application for calling a taxi in Figure 4, four actors “Customer”, “TaxiApp”, “Navigation” and “TaxiDriver” exist. We assume the application

---

<table>
<thead>
<tr>
<th>Customer</th>
<th>TaxiApp</th>
<th>Navigation</th>
<th>Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify destination</td>
<td>Accept his destination</td>
<td>Accept his destination</td>
<td>Specify destination</td>
</tr>
<tr>
<td>Accept destination</td>
<td>Tell route</td>
<td>Accept reservation</td>
<td>Check navigation</td>
</tr>
<tr>
<td>Identify pickup location</td>
<td>Tune</td>
<td>Identify route</td>
<td>Drive his car</td>
</tr>
<tr>
<td>Find suitable Taxi</td>
<td>Notify success</td>
<td>Drive to the pickup location</td>
<td>Arrive at customer’s destination</td>
</tr>
<tr>
<td>Identify suitable Taxi</td>
<td></td>
<td></td>
<td>Arrive at his destination</td>
</tr>
<tr>
<td>Request reservation</td>
<td>Reservation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5. Activity diagram of an application for asking usual cars as a taxi
“TaxiApp” is installed in customers’ smartphones, and each customer can call the suitable (normally the nearest) taxi. The application “TaxiApp” communicates with car navigation systems in taxis, and it can identify each taxi’s location and vacancy. When the application finds a suitable taxi, it automatically sends a reservation request to the taxi. If the taxi driver “TaxiDriver” may accept the request, the reservation is succeeded. Although the taxi also uses its navigation system after picking its customer up, we omit it in the figure for simplicity.

By comparing these two diagrams, similar elements exists. Actions in “Navigation” are almost the same in the systems. In the navigation system of Figure 3, the driver performs both actions “specify destination” and “drive a car”. In the taxi application of Figure 4, a customer performs the former action, and a taxi driver performs the latter one. We also find the same action “accept destination” in both diagrams, but different actor performs the action respectively.

According to the same actions in “Navigation”, we combine these two diagrams into one activity diagram in Figure 5. In the diagram, we basically substitute “TaxiDriver” for “Driver” in Figure 3 like Uber1. We omit “TaxDriver” in Figure 5 for avoiding the complexity of the figure, but “TaxiDriver” may continue their works.

As a result, the possibility of catching taxi will increase because usual drivers can also play a role of taxi drivers. Although we have to take security and payment issues into account in the next step, we can find synergy of these two systems by our method.

3.2 Wipers of cars and local weather report

In this example, we focus on the following two systems. Because we assume the former system for rainy forecast does not exist, we explore how to develop it efficiently.

- Rainy report system for each small area, e.g. 1 km square2, in a big city like London or Tokyo (The system will be developed, i.e. not exist yet)
- Systems in a car such as wipers control and GPS

Recently, heavy rain in a small area becomes a big problem in Tokyo. Many people can keep away from such heavy rain if the former system exists. However, it is hard to forecast such rain in a small area without rainy logs of each small area.

The top area in Figure 6 shows data flow diagrams of a rainy report system and systems in a car. Although “Sense rain” is represented in a process, it requires some hardware devices for sensing rain. In addition, we have to install many instances of such a process around the city. The location of the process “Sense rain”, each rainy status and time are stored in a data store “rainy history”. We assume the cost of developing the system is not low mainly because it requires many devices around the city.

Note that the process “Report rain” forecasts rainy status for each small area on the basis of the rainy history. Each user then gets the forecast of his/her location via his/her smartphone or something like that. We omitted such issues both in Figures 6 and 7.

1 https://www.uber.com/
2 The average distance between train stations in a line “Yamanote Line” in Tokyo is about 1.2 km.
Drivers usually uses and controls wipers according to the strength of the rain. The status of the wipers thus reflect the strength of the rain. In addition, GPS system is installed in most cars today for driving navigation. We simply added such systems in a car at the bottom of Figure 6. The models in the figure let us identify “Weather” activates a process “Change visibility” in the same way as a process “Sense rain”. We also identify a process “Control wipers” transitively depends on the “Weather”. In addition, car’s location can be identified with the help of GPS. Therefore, we can assume “Control wipers” and “Identify location” can record the quasi-rainy history. We can thus substitute rain sensors around the city for the cars and their systems as shown in Figure 7. As a result, we can save the cost for installing sensors around the city. Data flow diagrams in Figure 6 helps us to find reusable information for the new system.

3.3 Discussion

There are several challenges to perform this method. First, it is not easy to choose the suitable modeling language. Currently, we have to describe models in different nota-
tions until we have identified similarities and good combinations. Second, finding the same or similar elements depends on the expertise of requirements analysts. Natural language processing will not be useful because labels and annotations in each model element are normally very short and domain specific. Extending each modeling notation is one of the practical ideas to find them. For example, we may put some semantic tags on each element by using stereo types. Third, focusing on existing problems is not sufficient to discover new requirements. To find needs and desire of which stakeholders are unaware, creativity techniques mentioned in the next section can be used but examining such techniques is our future issue.

4 Related Work
There are few approaches for sharing requirements and for exploring unaware requirements by comparing and combining existing or planning systems/services. Instead, we can find methods and techniques for satisfying given requirements by comparing and combining them. In [2], service composition is performed by using goal models so that required functionalities are satisfied. In [3], a requirements analysis method using UML for SoS is proposed. The method also focuses on how to satisfying given efficiency requirements. In [4], a technique for monitoring the deviation from given requirements in SoS is proposed. In a chapter [5], factory automation using SoS is focused. New idea discovery and exploration is not also focused on this chapter. Finding missing requirements is a little bit similar to our research goal. However, most researches [6], [7] in the area focuses on the completeness of requirements. In addition, they do not take the synergy of several systems into account.

There already exist creativity techniques [8], [9], and some of them have been applied to requirements engineering [10], [11]. However, such techniques are not directly applied to models and their elements. Although one research applied one creativity technique to a goal model [12], it only focused on a single system.

Software product line [13] focuses on comparison among systems. However, this research focuses on similar systems and their variabilities and commonalities. Our research focuses on comparison and integration among different systems and activities. In a paper [14], integration of different systems and activities were focused. However, the paper only focused on the negative impact of such integration, i.e. security threats.

5 Conclusion
In this paper, we proposed and exemplified a method for sharing, substituting elements among different systems by comparing the elements. We also focused on discover new requirements through the combination of the elements. We use any modeling notations for our comparison and combination. Using different notations helps us to find model elements that can be shared or substituted (Answers of RQ1, 2). For example in a rainy report system in section 3.2, focusing on data flows enables us to find the substitution easily. Problems in an activity are resources of unaware requirements. Because elements in different systems and their combination can become the solution of the problems, our method contributes to the discovery of unaware requirements (Answer of RQ3). However, it seems to be insufficient because not all requirements comes from problem
solving. Our future works are summarized in section 3.3 as discussion: choosing the suitable modeling language, finding same/similar elements, discover new requirements beyond the solution of current problems.

Acknowledgement This work was supported by JSPS KAKENHI Grant Numbers 18K11249, 16H02804, 15H02686, 17K00475 and 16K00196.

References


