ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 8, February 2018

Web Service based Filtering for Discovering Latent Requirements

Ryuichi Takahashi[†], Yasuo Tsurugai[‡], Yoshiaki Fukazawa[‡]
[†]Department of Computer and Information Sciences, Ibaraki University, Japan
[‡]Department of Computer Science and Engineering, Waseda University, Japan

***Abstract*— The discovery of latent requirements and the implementation of functions that satisfy such requirements are important to win market shares. Existing engineering methods stimulate analysts' imagination and creativity to support the discovery of such requirements. However, the results depend on the analysts and are unstable. Herein we propose a method using a web service composition technique to discover latent requirements practically. The proposed method is independent of analysts' ability.**

***Index Terms*— Requirement Engineering, Latent Requirement, Web Service Composition.**

I. INTRODUCTION

Users will not accept software that does not meet their needs. Consequently, requirement analysis, which is the initial process of software development, identifies users' needs. This process mainly discovers and models the functions that users' need and the system goals. If this process is neglected, development resources may be wasted. Furthermore, to gain market shares, it is important to realize functions that are not already provided by other systems because acquiring new users will be difficult. Discovering functions that do not currently exist is challenging, especially when customers are not aware of them. Herein a requirement that users are not aware of is referred to as a latent requirement, and a latent function is a function that achieves a latent requirement.

Approaches [1], [2] to discover latent requirement is to support the imagination and creativity of analysts. These approaches strive to provide inspiration to identify latent requirements and to organization information. These approaches depend on the analyst.

Another approach [3] is to mechanically generate candidates for requirements and functions, and manually evaluate whether the generate candidates are useful to acquire latent requirements. This approach is independent of analysts' imagination and creativity. However, screening candidates generated mechanically is burdensome because many candidates are useless.

In this research, we aim to reduce the number of candidates in the manual evaluation by improving the latter approach. We propose a method to filter both useless and costly candidates. Our method is independent of analysts' imagination and creativity to discover latent requirements and finds latent requirements at low cost.

II. PROPOSAL METHOD

To provide new applications, it is important to determine latent requirements. However, discovering latent requirements is difficult using methods that depend on analysts' imagination and creativity. Our method allows latent requirements to be discovered independent of analysts' imagination and creativity. However, the exhaustive composition of services produces vast results, and judging the usefulness of the results is too costly. In this research, we compose web services exhaustively and then filter the results based on the following criteria:

- Functions not provided by other systems
- Functions composed of fewer services
- Functions with low execution costs

Our method aims to reduce the burden on analysts while simultaneously discovering latent requirements and functions by providing potential requirement candidates while removing useless ones. It involves three steps (Fig.1). Step 1 generates many requirement candidates mechanically using web service composition techniques. Step 2 reduces requirements candidates using the proposed filters. Step 3 groups the requirement candidates so that analysts can easily evaluate their usefulness. Each step is explained in detail below.

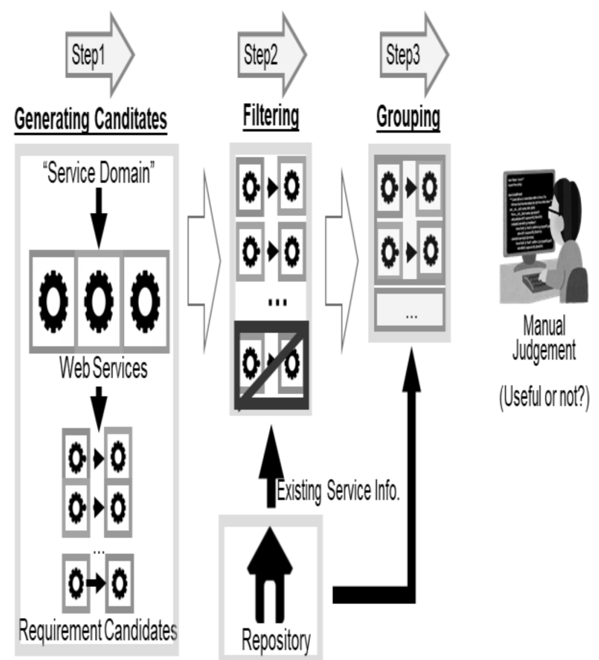

Fig.1 Overview of proposal method

A. Generation of requirement candidates

In the first step, all service execution sequences are retrieved from a set of web services. First, the set of services is obtained from the development target domain. Repositories can search services in service-oriented architecture field. Analysts obtain a set of services from the repositories using domain information as a clue. Examples of such a repository include UDDI [4] and ProgramableWeb [5].

Next, all service combinations are derived from their dependencies. The dependencies between web services are represented by a connection between the input and output. By passing the output of one service to another service, services can be connected to realize more complex functions. To pass data, the data types must match. The data type is defined in WSDL, which is language to describe the interface of web services. There are more domain specific representations than general programming languages. In this research, we use SDG (Service Dependency Graph) [6], [7] to represent service dependencies.

SDG can be represented by a directed graph between web services and data. Fig.2 shows an example of SDG, where squares and circles represent web services and data, respectively. An arrow from data to a web service means input, while an arrow from a web service to data means output. Each service can input/output several data, and each data can be passed as input to multiple web services. SDG can comprehensively represent the dependencies between web services.

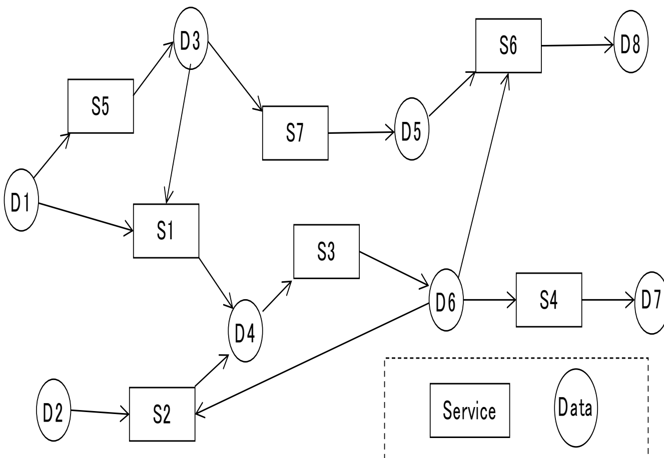

Fig.2 Example of SDG

Algorithm 1 generates the initial candidates by composing web services. First, it scans all web services in SDG and searches for a web service (*ws2*) that can provide input data of a certain web service (*ws1*). If *ws1* requires only one input, *ws1* and *ws2* are composed sequentially. If *ws1* requires two or more input, one or more services to provide input data must be collected. Web services, which provide each input, are searched by the retrieved SDG and unions of combinations of such web services are created. This algorithm finishes by retrieving entire SDG or reaching to maximum service length. All combinations are obtained as initial candidates.

Algorithm 1 Generating initial candidates

```
serviceNum ⇐ 0
Candidates ⇐ DomainWS
for serviceNum < maxComposedServiceNum && Candidates are updated do
    Composite ⇐ Candidates
    for all ws1 ∈ Candidates do
        for all ws2 ∈ DomainWS do
            if ws2.Outputs contains at least one of ws.Inputs then
                comp ⇐ ws2 combines ws1
                Composite.add(comp)
            end if
        end for
    end for
    Candidates ⇐ Candidates + Composite
    increment serviceNum
end for
```

Fig.3 shows the process to generate candidates in the above example. First, each single web service is listed. Tuple $< s_1, s_2 >$ means web service s_1 passes output to s_2 input and m means manual execution by users. In this case, seven are listed and they are executed by user inputs. Next, each web service is extended. Each web service is connected with the web service that provides its input to create a service union. In the case of S6, data D5 and D6 are needed. S6 makes unions with S3 and S7. When connecting < m, S6 > and < m, S3 >, input D6 are passed from S3. This creates a union {< m, S3 >, < S3, S6 >}. This means user executes S3 manually and the result is passed and latter services are executed automatically. However, D5 is also required to execute S6. These inputs are assumed to be given first by the user.

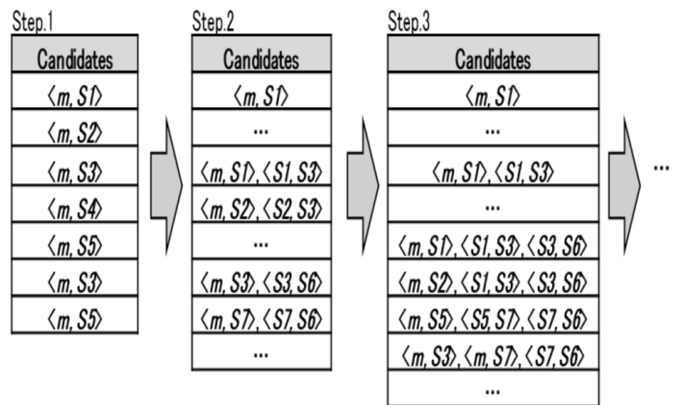

Fig.3 Process to generate requirement candidates

B. Filtering requirement candidates

The number of candidates generated by mechanically combining web services becomes enormous. We propose three kinds of filters. By using these filterings we reduce the generated candidates.

1) Filtering of existing functions

This filter aims to identify useful functions that are not implemented in other systems from the generated candidates. Therefore, candidates similar to existing applications are removed. To discover existing applications, we use the BPEL (Business Process Execution Language) descriptions stored

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 8, February 2018

in repositories of web services (e.g., ProgramableWeb). BPEL is a definition language of composition services. By comparing these descriptions and candidates, the similarity between candidates and existing applications can be evaluated.

Fig.4 shows an example of a BPEL description, which composes two services, "Search Hotel" and "Reserve Hotel". The *invoke* and *assign* elements are used to define dependencies between services. The *invoke* element specifies the execution of a specific service. The *inputVariable* and *outputVariable* in the *invoke* element are definitions of input/output data types. In addition, the *assign* element specifies the transfer of data between services. In this example, the data "HotelID" in the "Search Hotel" service is transferred to the "Reserve Hotel" service. In this way, it is possible to define cooperation between services by defining the transfer of data between them.

```
…
<sequence>
    …
    <invoke operation="Search Hotel" inputVariable="InputData1" outputVariable="OutputData1">
    </invoke>
    <assign>
        <copy>
            <from>$OutputData1.HotelID</from>
            <to>$InputData2.HotelID</to>
        </copy>
    </assign>
    <invoke operation="Reserve Hotel" inputVariable="InputData2" outputVariable="OutputData2">
    </invoke>
    …
</sequence>
…
```

Fig.4 Example of BPEL description

Algorithm 2 collects BPEL descriptions belonging to target domain from repositories. We compare each candidate with the service invocation sequence read from the invoke element of BPEL. If they match, the filter removes that candidate as an existing function. Also, regarding candidates having only one service, it means that the function is consistent with an existing single web service, so it is filtered.

Algorithm 2 Filtering existing functions

for all c ∈ Candidates **do**
  get all BPEL Files from repositories
  for all bpel ∈ BPEL Files **do**
    if c has same service invocation sequence with bpel **then**
      remove c from Candidates
    end if
  end for
  if c is constructed from single service **then**
    remove c from Candidates
  end if
end for

2) Filtering redundant services

This filter aims to remove candidates that include redundant service executions. In this research, a redundant service execution means a cyclic execution of services, which wastefully increases execution costs.

Fig.5 shows an example of redundant service. This example shows the dependencies among three Web services, "RecommendNearHotel", "SearchPhoneNumFromAddress", and "SearchAddressFromPhoneNum". By composing these services, we can obtain the phone number of a nearby hotel. However, it can convert repeatedly between address and phone number because these two services have cyclic dependencies. Even if these services are invoked more than once, the results do not change. Thus, the impact is negligible. Multiple invocations of such services are redundant, and removing candidates containing such service invocation reduces cost.

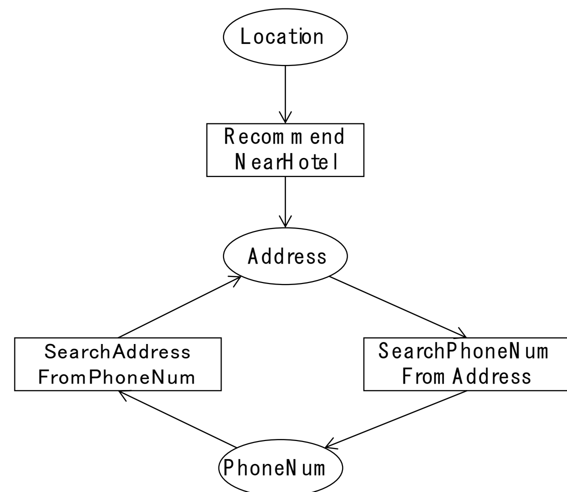

Fig.5 Example of redundant service execution

Not all services with cyclic dependencies are redundant. Fig.6 shows the dependencies between two services, "SearchNearbyHotel", "SearchNearbyTouristSpot". This example has a cyclic dependency. However, even if the data types match, the output data does not necessarily match. Due to the gradual shift in the search place, different results may appear. Although the output data is the same, a service with a secondary influence on the real world exists. To judge redundant service invocation, the data type, data value, and influence must be considered.

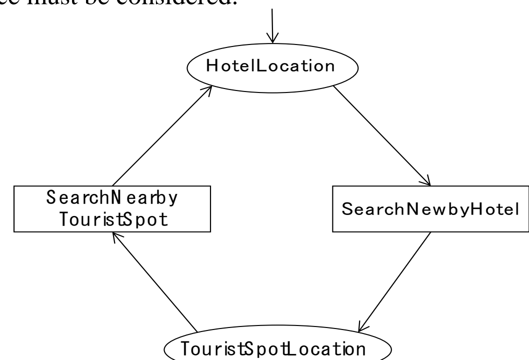

Fig.6 Example of not redundant service execution

IJEIT
ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 8, February 2018

Algorithm 3 removes candidates with redundant service invocation. First, all candidates are scanned to check if they contain cyclic dependencies. Circulation is determined on whether the output data type of one web service matches the output data type of a web service that has already been invoked. If a candidate contains a cyclic dependency, the data value and influence are checked. The check actually invokes the web services. In our method, each service constructing the circulation is invoked with test data, and a match of data values at the circulation point is evaluated. If the data values match, it means that the redundant invocation exists. we filter such a candidate. Invocation data are cached and reused in other verification cases. Caching reduces the judgment cost of redundant service invocation. Furthermore, the influence can be assessed by referring to the WSDL description. Even if there is a circulation, influential candidates are not filtered. By filtering candidates that include redundant service invocations, the number of candidates is reduced.

Algorithm 3 Filtering redundant services

```
for all c ∈ Candidates do
  cyclic ⇐ false
  GeneratedData ⇐ null
  add inputs of in GeneratedData
  for all ws ∈ c do
    if GeneratedData contains outputs of ws then
      cyclic ⇐ true
      if data cycle is not meaningful then
        remove c from Candidates
      end if
    end if
    add ws.output in GeneratedData
  end for
end for
```

3) *Filtering costly input*

This filter aims to identify functions with low user execution costs. User costs in this method mean the amount and complexity of input data provided by the users. Some web services require complex structured data or data with large number of digits like an ID. These inputs are difficult to provide manually and should be provided by other web services. A function requesting manual input of data that is difficult for the user to provide tends to have a low satisfaction even if it is useful. This filter removes such functions from the candidates.

Algorithm 4 removes candidates with costly inputs. This algorithm uses the BPEL description stored in repositories to judge whether user input is difficult.

Algorithm 4 Filtering costly inputs

```
ServiceInputs ⇐ null
UserInputs ⇐ null
for all bpel ∈ BPEL File do
  for all ws in bpel do
    add all ws's inputs provided by users in ServiceInputs
    add all ws's inputs provided by other services in UserInputs
  end for
end for
CostlyInput ⇐ ServiceInputs − UserInputs
for all c in Candidates do
  for all ws in c do
    for all uinput ∈ userInputs of ws do
      if uinput contains in CostlyInputs then
        remove c from Candidates
      end if
    end for
  end for
end for
```

First, it collects the BPEL descriptions and analyzes them to list data that can be provided by other web services. These data are considered that they are costly to provide manually. Candidates where users need input such data are removed.

C. *Grouping requirement candidates*

This step effectively presents candidates after filtering to requirement analysts. Because simply listing makes judgment difficult, similar candidates are grouped when presented to analysts. Then analysts evaluate whether the candidate is really useful.

The results are presented in a tree structure based on the execution sequence of the services. If a specific candidate is determined to be useless, then lower nodes (more extended candidates) can be considered useless collectively, reducing the time burden and increasing efficiency.

III. EVALUATION

In this section, we investigate the effectiveness of the proposed method by a user test.

A. *Preparation*

We prepared 19 web services. Then we examined whether latent functions can be found in reference to these web services. We targeted ten students in the Graduate School of Information Science. They used a combination of ten web services to design latent functions (composition service).

As a result, 97 composition services were discovered from the students' idea. We assumed that these 97 services are existing functions, and then implemented an experiment to determine how many other useful functions are found by our approach.

B. *Result*

Table.1 shows the results. Exhaustively combining 19 web services without filtering produced 5929 candidates. On the other hand, our proposed method with filtering reduced the number of candidates to 207.

Table.1 Number of candidates after filtering

Filtering methods	Number of Candidates
All candidates	5929
Use existing function filter	5886
Use redundant function filter	493
Use costly data filter	2386
Use all filters	207

Next, the ten subjects assessed the usefulness of the 207 candidates on a five-point scale (5: very useful, 4: useful, 3: unsure, 2: useless, 1: very useless). Of the 207 candidates, 111 received an average rating of 4 or more.

To evaluate the efficiency grouping, five students analyzed each candidate with and without grouping. We then compared the results. All the students indicated that grouping made the analysis easier.

IJEIT
ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 8, February 2018

IV. DISCUSSION

Using three types of filtering, our method reduced 5929 mechanically generated candidates to 207. This is a cost reduction of approximately 97 percent. If an analyst can evaluate the usefulness of each candidate in 30 seconds, about 2 days are necessary to screen 5929 candidates. However, only 2 hours are needed to screen 207 candidates. Considering that analysts must judge the useful of each candidate manually, our method yields a significant cost reduction.

In addition, 111 out of the 207 candidates (53%) were determined to be potentially useful and 71 were identified as actual unknown functions. However, these 71 candidates contained a lot of similarities, and were edited into seven latent functions. Although it is still necessary to improve the similarity judgment of the generated candidates, certain effects are obtained from the viewpoint of finding candidates that lead to latent functions.

The candidates that were removed by the filters were also analyzed. Some candidates that were deemed useful by analysts were filtered. In this approach, we mainly proposed filtering based on information of existing web services and a clear dependency information. It is likely that adding or improving the filtering methods will identify these candidates. In the future, it may be possible to add new filtering methods using a heuristic approach, artificial intelligence, or machine learning.

V.RELATED WORK

Previous works have proposed various requirement analysis methods. M. A. Boden [8] defined three ways to support analysts' imagination. R. B. Svensson et al. [9] compared these three methods to brainstorming to assess their effectiveness.

T. Bowmik et al. [10] proposed a method belonging to Combinatorial Creativity, which is one of the abovementioned three types. It aims to discover new requirements by collecting past developer's documents, stimulating the imagination of developers by combining words contained in them, and presenting the results to developers. Our method automatically generates candidate functions, and then developers assess the usefulness of the general candidates. It differs from previous studies in terms of the abstraction degree of the information presented and the intent is not to stimulate imagination and creativity of analysts.

K. Zachos et al. [11], [12] proposed a requirement analysis method using web services. In this method, analysts input an ambiguous requirement about the software. Then descriptions of a web services with close functions to the requirement are presented. By reading and understanding the descriptions, analysts can refine ambiguous requirements and discover new requirements. Their method supports generating wider and more flexible ideas by searching not only similar domain web services but also other domain services. It differs from the proposed method in that manually modification by analysts is necessary to modify the requirement. K. Zachos et al. [13] also compared the requirement analysis method using a web service and a method using a use case. The web service-based method discovered requirements that were not identified in the use case-based method.

Various methods have also been proposed for composing Web services. R. Tang et al. [14] proposes a method of defining patterns in the way of combining Web services and obtaining them from the execution log of the synthesis service. This method analyzes execution logs and extract service invocation sequences. By mining these sequence, they defined web service composition patterns. Our approach generates initial candidates based on input/output connections. This approach generates many useless candidates and needs filters to remove them. By using web service composition patterns, useful candidates may be generated easier. Our approach uses SDG to judge connectable services. SDG is proposed by Q. Ling et al. to express dependencies between web services. SDG uses data types of input/output to judge connectability of web services. Z. Gu et al. [7] proposes SDG+ which extends SDG. SDG+ can treat data dependencies between service specific data. For example, ID is very popular data and many of services adopt it to identify own data. However, ID is not necessarily compatible even for services of the same domain. Our approach aims to find latent functions. Since it is enough to judge the semantic compatibility of data, SDG is sufficient. If we extend to generate latent functions automatically, SDG+ may be useful to solve data compatibility.

VI. CONCLUSION

Mechanical service composition generates an enormous number of candidates. It is unrealistic to evaluate the usefulness based on the requirements of analysts for all combination of web services. In this study, we propose a method to support the discovery of new requirement using a service composition method. Candidates with a low usefulness and

High execution costs are removed using three types of filtering methods. A validation involving a user test confirms that our method reduces the useless candidates by 97%, and of the remaining candidates, 34% are useful and undiscovered. Our approach allows latent requirement to be discovered practically without depending on analysts' imagination and creativity.

Future research should focus on improving the filtering methods. One aspect is to improve the accuracy of similarity judgments between candidates and existing services. This should narrow the number of potential candidates. Another aspect is to devise new filtering methods. Current filtering methods are based on information of existing services and dependencies between services. It may be possible to devise a filter based on implicit relations using machine learning.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 8, February 2018

Further evaluations applied to other domains are also necessary.

REFERENCES

[1] J. Robertson, "Eureka! why analysts should invent requirements", IEEE Software, vol.19, no.4, pp.20–22, July 2002.

[2] N. Maiden, S. Jones, I.K. Karlsen, R. Neill, K. Zachos, and A. Milne, "Requirements engineering as creative problem solving: A research agenda for idea finding," Requirements Engineering Conference (RE), 2010 18th IEEE International, pp.57 – 66, IEEE Computer Society, 2010.

[3] J. Robertson and C.L. Heitmeyer, "Requirements analysts must also be inventors," IEEE Software, vol.22, 2005.

[4] N. Apte and T. Mehta, "UDDI: Building Registry-based Web Services Solutions", Pearson Education, 2003.

[5] "Programmable Web - apis, mashups and the web as platform.", http://www.programmableweb.com/

[6] Q.A. Liang and S.Y.W. Su, "AND/OR graph and search algorithm for discovering composite web services," Int. J. Web Service Res., vol.2, no.4, pp.48–67, 2005.

[7] Z. Gu, J.Z. Li, and B. Xu, "Automatic service composition based on enhanced service dependency graph," 2008 IEEE International Conference on Web Services, pp.246–253, 2008.

[8] M.A. Boden, The Creative Mind: Myths and Mechanisms, Basic Books, Inc., New York, NY, USA, 1991.

[9] R. Berntsson Svensson and M. Taghavianfar, "Selecting creativity techniques for creative requirements: An evaluation of four techniques using creativity workshops," 08 2015.

[10] T. Bhowmik, N. Niu, A. Mahmoud, and J. Savolainen, "Automated support for combinational creativity in requirements engineering," 2014 IEEE 22nd International Requirements Engineering Conference (RE), vol.00, pp.243–252, 2014.

[11] S. Jones, N. Maiden, K. Zachos, and X. Zhu, "How service-centric systems change the requirements process," 11th Workshop on Requirements Engineering: Foundation for Software Quality: REFSQ2005, 2005.

[12] K. Zachos and N. Maiden, "Inventing requirements from software: An empirical investigation with web services," Proceedings of the 2008 16th IEEE International Requirements Engineering Conference, RE '08, Washington, DC, USA, pp.145–154, IEEE Computer Society, 2008.

[13] N. Maiden, X. Zhu, S. Jones, and K. Zachos, "Does service discovery enhance requirements specification? a preliminary empirical investigation," 2006 Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'06 - RE'06 Workshop), vol.00, p.2, 2006.

[14] B. Upadhyaya, R. Tang, and Y. Zou, "An approach for mining service composition patterns from execution logs," Journal of Software: Evolution and Process, vol.25, no.8, pp.841–870, 2013.

AUTHOR BIOGRAPHY

Ryuichi Takahashi received the B.E., M.E. and D.E. from Waseda University, Tokyo, Japan in 2007, 2008 and 2012. During 2011–2017, he was assistant professor in Waseda University. He is now with Ibaraki University, Japan. His research interests include software engineering especially design of interactions for distributed systems.

Yasuo Tsurugai received the B.E. and M.E. from Waseda University, Tokyo, Japan in 2015, and 2017. And from April 2017, he now work as programmer at Sony Co., Ltd.

Yoshiaki Fukazawa received the B.E., M.E. and D.E. degrees in electrical engineering from Waseda University, Tokyo, Japan in 1976, 1978 and 1986, respectively. He is now a professor of Department of Information and Computer Science, Waseda University. His research interests include software engineering especially reuse of object oriented software and agent-based software.