Received: December 21, 2019

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Automatic Web Service Composition for SaaS Business Intelligence

Budi Harjo[1,2]* **Riyanarto Sarno[1]** **Siti Rochimah[1]**

[1]*Department of Informatics, Faculty of Information Technology and Communication, Institut Teknologi Sepuluh Nopember, Indonesia*
[2]*Department of Informatics Engineering, Faculty of Computer Science, Universitas Dian Nuswantoro, Indonesia*
* Corresponding author's Email: budi.harjo13@mhs.if.its.ac.id

Abstract: An atomic web service is not enough to meet complex user needs. This need can only be fulfilled by composing web services that perform operations automatically. The result of web service composition is a workflow. To the best knowledge of the authors, in the currently available methods existing a workflow output is not used to replace a workflow input. Also, web service backup is not provided. This can cause the transaction to stop because the web service that is being accessed cannot be invoked. Therefore, the authors propose a method that can utilize a workflow output to replace an input of another workflow and provide a web service backup. The supporting techniques used are *tf-idf* weighting and cosine similarity. The proposed method was applied to compose web services in a SaaS Business Intelligence application. The modules in Business intelligence are run using workflows that are composed based on the similarity between input parameters and output parameters required by the user with the web service metadata provided. The experimental results show that the proposed method can successfully produce workflows whose input can be replaced by other workflows and provide appropriate web service backup.

Keywords: Automatic, Semantic, Web, Service, Composition, SaaS, Business, Intelligence.

1. Introduction

A web service is a service application based on Software as a Service (SaaS) by utilizing Service-Oriented Architecture (SOA) [1]. It uses the Web Service Description Language (WSDL) as a tool to compose cross-platform and reusable web services, making it very valuable for users [2]. In order to meet user needs, web service composition (WSC) is needed. Several atomic web services can be combined to form a composite web service or a workflow. Several workflows can be combined to more complex workflows. Web service composition can be done using existing tools such as Business Process Execution Language (BPEL) Designer Project [3], Business Process Model and Notation (BPMN) [4], and several other methods that have been proposed by previous authors [5-17].

These tools can only be used manually [3, 4], i.e. the user must connect web service outputs to web service inputs one by one. With this simple method, the user cannot compare the semantic similarity between the different inputs or outputs to be selected, so that the user may not get a correct workflow. Several methods have been proposed to compose web services semantically, namely, Semantic Markup for Web Services (OWL-S) [5], Web Service Modeling Ontology(WSMO) [6], and Semantic Annotations for WSDL and XML Schema (SAWSDL) [7]. In the last few decades these semantic methods have encouraged researchers to develop automatic web services composition [8, 9]. These methods are intended to compose web services automatically with initial values of input parameters as precondition to get the requested output. However, these methods cannot find outputs that are searched by keyword from the web service that has been found ('branched search'). For example, the user requests the *TotalSales* and *ProductName* outputs, but at the stage of searching

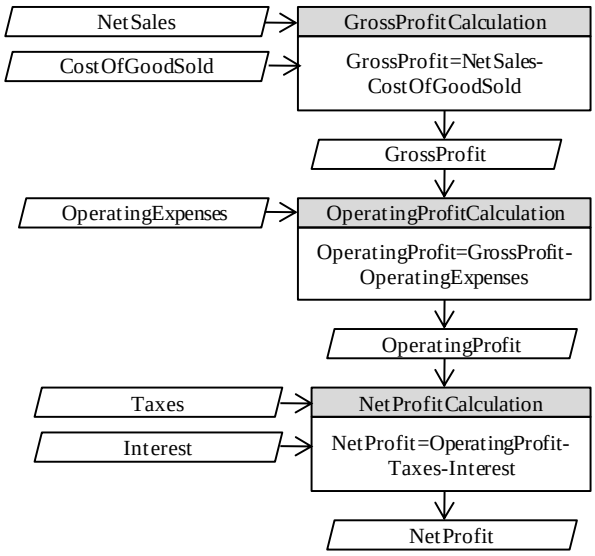

Figure. 1 Replacing a workflow input with a workflow output

only *TotalSales* can be found in the *Sales web service*, which has the *ProductId* keyword. Ultimately, these methods cannot find the *ProductName* output, because they cannot utilize the *ProductId* keyword to search for the *ProductName* output.

D. Elsayed, E. Nasr, A. El Din Ghazali, and M. Gheith also proposed a Web service composition method they called PGAQK [10]. This method uses a Parallel Genetic Algorithm (PGA) based on Q-learning, which they integrate with K-means clustering. However, this method cannot find outputs that are searched by keyword from the web service that has been found ('branched search').

To resolve the branched-search problem several authors have proposed methods. N. Arch-int, S. Arch-int, S. Sonsilphong, and P. Wanchai [11] proposed a graph-based method that can overcome the problem well. However, it cannot overcome more complex branched-search problems, i.e. replacing a workflow input with a workflow output, as shown in Fig. 1.

This problem often arises in sequential mathematical operations, especially in SaaS Business Intelligence (BI) applications. In the first case presented in this paper, as shown in Fig. 1, there are 3 workflows, each of which only has 1 web service, namely, *GrossProfitCalculation*, *OperatingProfitCalculation*, and *NetProfitCalculation*, respectively. The *GrossProfitCalculation* workflow has 2 inputs, namely, *NetSales* and *CostOfGoodSold*. The *OperatingProfitCalculation* workflow has 2 inputs, namely, *GrossProfit* and *OperatingExpenses*. The *NetProfitCalculation* workflow has 3 inputs, namely, *OperatingProfit*, *Taxes* and *Interest*. The *GrossProfitCalculation* workflow has 1 output, namely, *GrossProfit*. The *OperatingProfitCalculation* workflow has 1 output, namely, *OperatingProfit*. The *NetProfitCalculation* workflow has 1 output, namely, *NetProfit*. In fact, the *OperatingProfit* input of the *NetProfitCalculation* workflow can be replaced by the output of the OperatingProfitCalculation web service, but this cannot be worked out using the method proposed by [11]. A similar method was also proposed by F. Zhang, Q. Zeng, H. Duan, and C. Liu [12], but the weakness of this concept is also the same as those proposed by [11].

Some concepts about business processes have also been proposed by previous authors, namely, R. Sarno, W. W. Ayu, A. N. Fajrin, D. Manfaat, M. S. Arif, and I. Baihaqi [13], C. S. Wahyuni, K. R. Sungkono, and R. Sarno [14], and K. R. Sungkono, U. E. N. Rochmah, and R. Sarno [15]. The weakness of these concepts is the same as the weakness of the method proposed by [11, 12].

In order to get a good workflow, P. Wang, Z. Ding, C. Jiang, M. Zhou, and Y. Zheng proposed the Uncertainty Execution Effects method for composing web services. It matches the input and output parameters with existing web service metadata and also detects uncertain effects caused by values entered into the web service by using the Graphplan method [16]. This method can successfully shift the workflow to the appropriate web service based on the effects that occur. This is one of the methods that inspired the authors to develop the method proposed in the present paper. The authors apply a similar method to a BI module Sales Analysis to Make a Sales Projection and Calculate Bonus (Q18), however, without using the Graphplan method, because this method has the same weaknesses as [11-15].

'Web Service Similarity with Standardized Descriptions' is the title of our previous paper [17]. To make automatic web services composition possible, WSDL files, the program source files from the web services, and the input and output parameters are uploaded first. Upon entering the input and output parameters, they are automatically matched with the inputs and outputs of each of the available web services. Matching failed if no web service matched the input and output parameters. In fact, there are web services that can be found based on keywords found in web services that were found before. In addition, this method also does not provide web service backup.

At present, the SaaS Business Intelligence applications provided by different providers use services owned by each provider [18], so they

cannot be developed by users freely using web services that can be found spread across the internet. As a result users cannot compose web services that are currently being developed for free.

Therefore, we propose a method that aims to replace an input in the workflow with an output in another workflow in the BI application that we propose, so that the reuse of the workflow is realized. In addition, the method we propose can also provide a web service backup on the BI application that we propose, so that the system can directly use the web service backup without having to match the input and output parameters with the web service metadata again which takes a long time. These are our contributions to this research.

This paper is organized as follows. Related work and contribution is shown in introduction in section 1. Section 2 shows a research method of this research. Section 3 shows a proposed automatic web service composition. Section 4 shows a experimental result. Finally, section 5 provides conclusions and future work.

2. Research method

This section presents background knowledge on services language descriptions and planners for solving web service composition problems.

2.1 Business intelligence

Business Intelligence (BI) is a process for taking large amounts of data, analyzing data, and presenting high-level sets of reports that summarize data related to business actions, enabling management to make fundamental daily business decisions [18]. Based on the 2019 Magic Quadrant for Analytics and Business Intelligence Platforms [19], BI is led by several well-known vendors, namely Microsoft, Tableau, Salesforce, Qlik, SAP, and others. They compete to get the top position in serving users. Currently, the technology they use is Software as a Service (SaaS). SaaS allows users to access applications through the Internet that are up and running on the SaaS provider's server and to use them for free or for a fee based on usage [20-22].

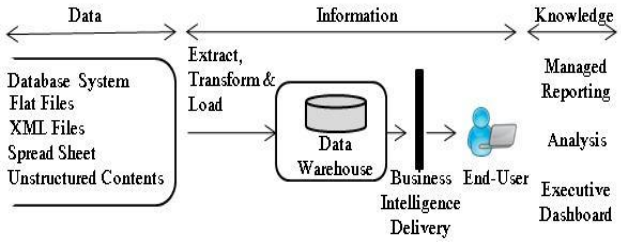

Figure. 2 Business intelligence architecture

BI applications offered by each provider are different, which can be seen from the BI architecture provided. However, in general the BI architecture is as depicted in Fig. 2.

In BI applications, one or more types of data can be used, as shown in Fig. 2. Furthermore, the data are extracted, transformed, and loaded for storage in a data warehouse. The data warehouse contains data used for online analytical processing (OLAP) and dashboards in BI [23]. In the BI application, the data warehouse is processed with OLAP to produce the information needed by the user.

2.2 Term frequency inverse document frequency

Tf-idf weighting was first introduced by G. Salton [24]. It stands for term frequency (*tf*) × inverse document frequency (*idf*). The *tf-idf* method is used to determine to what extent a word (term) is related to a document by weighting each word. It is often used as a weighting factor in information retrieval, text mining, and user modeling. Given a collection of terms $t \in T$ that appear in a set of N documents $d \in D$, each with length n_d, *tf-idf* weighting is computed as follows:

$$tf_{t,d} = \frac{tf_{t,d}}{n_d} \quad (1)$$

$$idf_t = \log \frac{N}{df_t} \quad (2)$$

$$W_{t,d} = tf_{t,d} \text{ x } idf_t \quad (3)$$

$$W_{di} = W_{d0} \text{ x } W_{di} \quad (4)$$

where $tf_{t,d}$ is the frequency of term t in document d, $tf_{t,d}$ is the inverse document frequency of term t, df_t is the document frequency of term t, $W_{t,d}$ is the weight (W) of term tin relation to document d, and W_{di} is the weight (W) of document d in index i.

2.3 Cosine similarity measurement

Cosine similarity measurement was used in this research to calculate the similarity between two elements in a web service that have two or more syllables. In the cosine similarity method two types of documents are distinguished [25].

The first type is the occurrence document and the second type is the query document [26]. The occurrence document can be described as follows :

$$\vec{d}=(w_{d0}, w_{d1}, ..., w_{dk}) \quad (5)$$

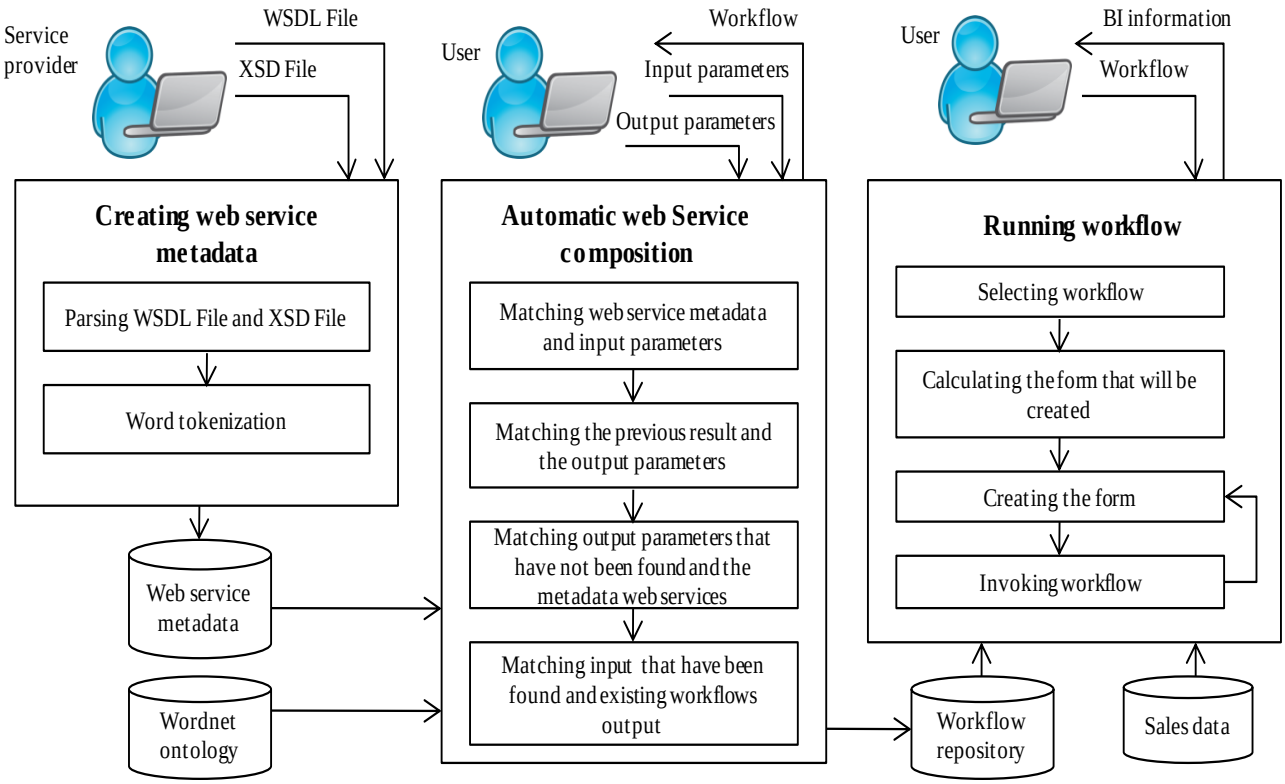

Fig. 3. Workflow of automatic web service composition

Table 1. Features and modules of Business intelligence application

Report	Executive Dashboards	Forecasting	What-If Analysis
Total Sales in a Year Range (Q1)	Sold Product Quantity in Current... (Q9)	Quantity (Q13)	Sales Projection (Q16)
Total Sales per Month in a Year (Q2)	Total Sales in Current Year (Q10)	Sales (Q14)	Calculate Bonus (Q17)
Total Sales per Product Id in a Year (Q3)	Total Profit in Current Year (Q11)	Profit (Q15)	Sales Analysis to ...(Q18)
Total Sales per State in a Year (Q4)	Sales Average Per Three Month (Q12)		Gross Profit (Q19)
Total Sales per State based on... (Q5)			Operating Profit (Q20)
Total Sales per Customer in a Year (Q6)			Net Profit (Q21)
Total Profit in a Year Range (Q7)			
Total Profit per Product in a Year (Q8)			

The query document is described as a vector shape:

$$\vec{q}=(w_{q0}, w_{q1}, ..., w_{qk}) \quad (6)$$

where w_{di} and w_{qi} ($0 \leq 1 \leq k$) are float numbers that indicate the frequency of each term in the document, while the dimensions of each vector correspond to all terms available in the document.

Based on the similarity vector, the similarity between the two vectors can be defined as follows:

$$Sim\ (\vec{q}, \vec{d})=\frac{\vec{q}.\vec{d}}{|\vec{q}|.|\vec{d}|}=\frac{\sum_{k=1}^{t} w_{qk} x w_{dk}}{\sqrt{\sum_{k=1}^{t}(w_{qk})^2} x \sqrt{\sum_{k=1}^{t}(w_{dk})^2}} \quad (7)$$

where $\vec{q}$ is vector q, $\vec{d}$ is vector d, $|\vec{q}|$ is the length of vector q, $|\vec{d}|$ is the length of vector d, and $Sim\ (\vec{q}, \vec{d})$ is the similarity between vector q and vector d.

3. Proposed automatic web service composition

In general, the method that is proposed in this paper has three processes, namely, 1) creating the web service metadata; 2) automatic web service composition; and 3) running the workflow, as shown in Fig. 3.

3.1 Creating the web service metadata

The first process shows that the provider creates the web service metadata that will be used to create composite web services or workflows. Anyone can register as a user to the application manager.

Table 2. Variants of *ProductInformation* web service

Service Code	Grounding		Model		Profile	
	Input Type	Output Type	Input Name	Output Name	Operation Name	Service Name
A.000	String	String	ProductId	Category	getProductInformation	SeekProductInformation
		String		SubCategory		
		String		ProductName		
A.001	String	String	ProductId	Category	getProductExplanation	SearchProducExplanation
		String		SubCategory		
		String		ProductName		
A.010	String	String	ProductId	Category	getProductInformation	SeekProductInformation
		String		SubCategory		
		String		ProductTitle		
A.011	String	String	ProductId	Category	getProductExplanation	SearchProducExplanation
		String		SubCategory		
		String		ProductTitle		
A.100	String	String	ProductId	Category	getProductInformation	SeekProductInformation
		String		SubCategory		
		String		ProductName		
A.101	String	String	ProductId	Category	getProductExplanation	SearchProducExplanation
		String		SubCategory		
		String		ProductName		
A.110	String	String	ProductId	Category	getProductInformation	SeekProductInformation
		String		SubCategory		
		String		ProductTitle		
A.111	String	String	ProductId	Category	getProductExplanation	SearchProducExplanation
		String		SubCategory		
		String		ProductTitle		

Table 3. Upload table

Field name	Type	Size
WebServiceId	Int	3
Userid	Varchar	20
UploadDate	Date	
WsdlFile	Varchar	100
XsdFile	Varchar	100
WebServiceName	Varchar	100
OperationName	Varchar	100
WebServiceAddress	Varchar	100
InputName	Varchar	50
OutputName	Varchar	50
InputType	Varchar	20
OutputType	Varchar	20

Furthermore, the application manager authorizes new users to create web service metadata, compose web services, and run workflows. To support this research, we replicated the dataset that was used in [11]. We wanted to use the actual dataset, but the server that stores it (http://www.webservicex.net/new/Home/Index) and the services repository (http://www.service-repository.com/) cannot be accessed. Therefore the authors replicated the dataset and stored it a *http://budiharjo.disertasi.com:4848/common/index.jsf*. The data the authors used for the BI application were data shared by Tableau Communications at *https://community.tableau.com/docs/DOC-1236.*

Making complex BI applications such as those made by vendors requires a lot of time and energy, therefore in this research a simple application was created to simulate the proposed method, as described in Table 1. However, the application can easily be developed further. In order to support the BI application, 147 web services were created using the Netbeans IDE 8.0.2 software. Each of them is able to produce wsdl and xsd files. They contain information about the web service concerned. To achieve the information, parsing files are needed. However, the information in the form of text contains prefixes, infixes, and suffixes that must be removed for calculating the similarity between texts. This removal is done using tokenization. The tokenization results are saved as web service metadata. The web service metadata were divided into 21 groups, which were given initial letter codes from 'A' to 'U'. From group 'A' to 'R' each had 8 web services encoded from '000' to '111'. For example, in Table 2, group 'A' contains product information web services and has 8 variants, whereas from 'S' to 'U' each group only has 1 variant.

Each web service metadata consists of 3 groups of elements, namely grounding, model, and profile. In this case, the terms used by OWL-S were adopted [5]. The web service metadata were stored in an upload table, as described in Table 3.

3.2 Automatic web service composition

The second process requires 2 types of parameters, namely, input parameters and output parameters. Input parameters are a collection of inputs that will be entered into the workflow by the user, while output parameters are a collection of outputs expected by the user in the workflow to be run. They will be matched with the web services metadata stored in the upload table (Table 3). The automatic web services composition flow is expressed in the TF-IDFwithCosineSimilarityMeasure algorithm provided in Appendix A and the AutomaticWSComposition algorithm provided in Appendix B. This process has 4 sub processes, namely, 1) matching web service metadata with the input names; 2) matching the previous result with the output names; 3) matching the output names that have not been found with the web services metadata; 4) matching the input names that have been found with existing workflows.

3.2.1. Matching input parameters with web service metadata

In The method allows more than 1 input parameter to be entered, for example in the first case, *OperatingProfit*, *Taxes*, and *Interest*. The output parameter is *NetProfit*. They are semantically matched with the web service metadata. Before being matched both are cleaned so that they do not contain prefixes, infixesor suffixes.

The web service metadata readings are expressed in the AutomaticWSComposition algorithm, step 1 and 2, where the system reads the Upload table that contains the web service metadata. The input parameter is added to the query array(*ArrayQI*) and the web service metadata are added to the document array (*ArrayD*). This step is expressed in the AutomaticWSComposition algorithm, step 4 to 12. Furthermore, *ArrayQI* and *ArrayD* are sent to the **TF-**IDFwithCosineSimilarityMeasure algorithm, which is a *tf-idf* algorithm that uses cosine similarity. *ArrayQI* and *ArrayD* must be tokenized before they are added to *ArrayTerm*. Hence, *ArrayTerm* is an array that contains members of *ArrayQI* and *ArrayD*. Each of member of this array will be semantically matched with all members of *ArrayQI* and *ArrayD*. The matching result is used to calculate the *tf* of the words presented by *ArrayTerm* in the document (*ArrayD*). To calculate *tf*, a similarity threshold of 0.8 was used here. Thus, *tf* will have a value of 1 if the similarity threshold is more than or equal to 0.8. To calculate *tf*, Eq. (1) is used.

The TF-IDFwithCosineSimilarityMeasure algorithm has several steps to look for similar words in the documents or *ArrayQI* and *ArrayD*, namely, a) calculating inverse document frequency; b). calculating weighted term document; c) calculating the length of each document; d) calculating the vector length; e) calculating the similarity between *ArrayQI* and *ArrayD* with the cosine similarity method.

3.2.1.1. Calculating inverse document frequency

The *tf* result is used to calculate *idf* as explained in the TF-IDFwithCosineSimilarityMeasure algorithm, step 27 to 36, where *df* is the number of *tf* that have values greater than or equal to 0.8.

3.2.1.2. Calculating weighted term document

Weighted Term Document is formulated by Eq. (3). It is expressed in TF-IDFwithCosineSimilarityMeasure algorithm, step 37 to 41.

3.2.1.3. Calculating the length of each document

The length of each document is calculated with Eq. (4). It is expressed in the TF-IDFwithCosineSimilarityMeasurealgorithm, step 42 to 46.

3.2.1.4. Calculating the vector length

The vector length is formulated by Vector Length = W_d x $W_{d,i}$. It is expressed in the **TF-**IDFwithCosineSimilarityMeasure algorithm, step 47 to 68.

3.2.1.5. Calculating the similarity of *ArrayQI* and *ArrayD* with cosine similarity

The cosine similarity of *ArrayQI* and *ArrayD* is calculated with Eq. (7). This is expressed in the TF-IDFwithCosineSimilarityMeasure algorithm, step 69 to 75.

3.2.2. Matching the previous result and the output parameters

This sub process needs the previous result, i.e. the result of matching the input parameters with the web service metadata to calculate the similarity between the output parameters and the web service metadata. This **sub process is expressed in the AutomaticWSComposition a**lgorithm, step 18 to 22.

3.2.3. Matching the output parameters that were not found with the web service metadata

This sub process serves to match the output parameters that were not found based on the results of matching the input parameters with the web service metadata. In the first case, all output parameters were found, so this sub process was skipped. This sub process is expressed in the **AutomaticWSComposition** algorithm, step 23 to 28.

3.2.4. Matching input parameters that have been found with existing workflows

This sub process serves to calculate the similarity between inputs that have been found with the outputs of each workflow stored in the workflow repository. This subprocess is expressed in the AutomaticWSComposition algorithm, step 29 to 43.

3.3 Running the workflow

The third process is shown in Fig.3. This process serves to run the workflow that was created by the user. It has 4 sub processes, namely 1) selecting workflow; 2) calculating form that will be created, 3) creating a form; and 4) invoking the workflow.

3.3.1. Selecting the workflow

This sub process serves to display all workflows created by the user. The user can choose one of them to run.

3.3.2. Calculating the form that will be created

The second sub process is used to calculate the form that will be created. The number of forms depends on the web service group separated by the '&&' symbol. For example, the workflow in the second case, *'57-58-61-62-59-60-63-64&&ZipCode_9-ZipCode_10-ZipCodes_13-ZipCodes_14-ZipCode_11-ZipCode_12-ZipCodes_15-ZipCode_16&& - ProductId_1-Zip-IdOfProduct_5-IdOfProduct_6-ProductId_3-ProductId_4-IdOfProduct_7-IdOfProduct_8'*.This workflow has 3 forms, because the workflow is separated by an '&&' symbol 2 times.

3.3.3. Creating the form

This sub process serves to create the form. The number and name of the inputs in the form are created based on the names of the inputs and the outputs in the web service.

3.3.4. Invoking the workflow

This sub process serves to call the workflow that has been chosen by the user. The workflow results can be used as input for the next workflow according to the method proposed in this paper.

4. Experimental result

There are 21 BI modules proposed as described in Table 1. The web service composition is automatically used to find the most appropriate workflow for each module. Fig. 4 shows the web service composition data input form for the module Net Profit (Q21). After running, the results are shown in Fig. 5.

In this research, to compare specifically with the results of other methods the authors decide to use the method [11], because the dataset and the method we used are similar to those used by them. The results of the comparison are explained in Table 4. It only displays 3 modules, because the other modules are the same type as them.

In the experiment, every atomic web service stored in a repository is coded from numbers 1 to 147. They are composed to be workflows or more complex workflows. The notation in the workflow that we use is not the same as the notation used by [11], but the meaning is the same, that is, the workflow generated consists of a standalone web service or more interconnected web services.

Some symbols are used in notation of workflows, including, the symbol '-' is used as a separator between similar web services in a workflow, for example, in the results of the module (Q1) for the results of the proposed method, meaning that the web service coded with number 17 has similarities to the web services behind it. The level of similarity starts from number 18 to number 24. Each web service coded from '18' to '24' is a backup of the web service coded '17'. Web service backups are not provided by workflows generated by the method [11].

Query Name: Net Profit (Q21)

Input Name Parameters

interest ▼ Add Row

Input Name
operatingProfit
taxes
interest

Reset

Output Name Parameters

netProfit ▼ If - ▼ - $ USA Then Calculate -

Output Name	Operator	Value	Output Name Effect1
netProfit	-		-

Reset

Submit Exit

Figure. 4 Web service composition data input form for the module Net Profit (Q21)

tf-idf

Term Name	Q OperatingProfit	D1 OperatingProfit 147	df	Log (n/df)
Operate	1.000	1.000	2	0.000
Profit	1.000	1.000	2	0.000

wtd

Q	D1
0.000	0.000
0.000	0.000

wdi

D1
0.000
0.000
0.000

Vector length

Q	D1
0.000	0.000
0.000	0.000
0.000	**0.000**
0.000	**0.000**

Result of Cosine similarity

D1 WsId
1.000 147

Composition name : Net Profit (Q21)
Final result : C20/OperatingProfit,Taxes,Interest

Figure. 5 Final results of cosine similarity measure for the module Net Profit (Q21)

Table 4. The results of the specific comparison

Name of Module	Parameters entered		Results of method [11]	Results of proposed method
	Input Name	Output Name		
(Q1)	BeginningYear, EndingYear	TotalSales	17	17-18-19-20-21-22-23-24
(Q3)	DemandedYear	ProductName,TotalSales	41&&ProductId_1	41-42-43-44-45-46-47-48&&ProductId_1-ProductId_2-IdOfProduct_5-IdOfProduct_6-ProductId_3-ProductId_4-IdOfProduct_7-IdOfProduct_8
(Q21)	OperatingProfit,Taxes, Interest	NetProfit	147	147‖C20/operatingProfit, taxes,interest

The symbol '&&' is used to separate two workflows composed in a more complex workflow, for example the results in the module (Q3) for the results of the proposed method, the first workflow '41-42-43-44-45-46-47-48' is joined with the second workflow 'ProductId_1-ProductId_2-IdOfProduct_5-IdOfProduct_6-ProductId_3-ProductId_4-IdOfProduct_7-IdOfProduct_8' using the symbol '&& '. This explains that the workflow that is run first is the first workflow, then the second workflow. In the second workflow there is a '_' symbol, the meaning is that to run the second workflow, the system has to match between the key field *ProductId* or *IdOfProduct* of first workflow and the key fields *ProductId* or *IdOfProduct* of the second workflow. They are stored in a web service that has codes '1' to '8'. *ProductId or IdOfProduct* matching is aimed to search for output name parameter *ProductName* not found in the first workflow. Workflow like this is generated by [11], but it does not have a web service backup as workflows generated by the proposed method.

Symbol '||' is used to separate 2 pieces of workflow, but only one of them can be selected, for example, in the results module (Q21) for the results of the proposed method. This workflow offers 2 choices. As the first choice the user can only run a workflow that has an atomic web service using *WebServiceId* '147' and as the second choice the user can run a workflow that has an atomic web service using *WebServiceId* '147', but the value of the *OperatingProfit* input is obtained from an existing workflow saved in the workflow repository with *CompositionId* '20'. If a workflow with *CompositionId* '20' contains an input value that must be obtained from another workflow, the system will also look for that workflow. This step also applies to the next workflow. Replacing a workflow input with another workflow output cannot be done in method [11].

In this research, as a comparison of results in general with other methods are described in Table 5. The comparison results described in Table 5 prove that only the proposed method can replace an input workflow with another workflow output and provide a web service backup.

4.1 Accuracy measurement

For measuring the accuracy of the web service composition system, the authors used a percentage of precision Eq. (8), recall Eq. (9), and F measure Eq. (10).

$$\text{Precision} = \frac{\text{TP}}{\text{FP+TP}} \text{ x100\%} \tag{8}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \text{ x100\%} \tag{9}$$

$$\text{F} - \text{Measure} = 2\text{x}\frac{\text{PrecisionxRecall}}{\text{Precision+Recall}} \text{ x100\%} \tag{10}$$

As shown in Table 6, the overall F measure score, i.e. the average accuracy of the proposed method, was 97.67%. The overall accuracy shown by the precision score was 95.45%, while the overall completeness of the web service search represented by recall was 100%. While data in the Healthcare domain is obtained from [11].

Mistakes (FP and FN) occurred when semantically matching parameters and web service metadata gave a result that was not expected by the authors. For example, the matching result of the words 'City' and 'Total' was 0.8. We think that this result is too high. This affects the use of other words in the parameters used.

Table 5. The results of the general comparison

The Methods	Replacing a workflow input with another workflow output	Web service backup
The Proposed Method	Available	Available
Process Execution Language (BPEL) Designer Project [3]	Not available	Not available
Business Process Model and Notation (BPMN) [4]	Not available	Not available
Semantic Markup for Web Services (OWL-S) [5]	Not available	Not available
The Semantic Web Service Ontology (WSMO)[6]	Not available	Not available
Annotation paths [7]	Not available	Not available
Efficient planners in large-scale service repository [8]	Not available	Not available
Planning-based semantic web service composition [9]	Not available	Not available
Parallel Genetic Algorithm (PGA) based on Q-learning [10]	Not available	Not available
Graph-based searching [11]	Not available	Not available
Composition context-based web services similarity measure [12]	Not available	Not available
Context Sensitive Grammar [13]	Not available	Not available
Weighted-Tree Declarative Pattern Models [14]	Not available	Not available
Heuristic linear temporal logic pattern algorithm [15]	Not available	Not available
Uncertainty Execution Effects [16]	Not available	Not available
Standardized Descriptions [17]	Not available	Not available

Table 6. Metric performance comparison of the proposed method

Domain	Number of WS	Number of queries	TP	FP	FN	Precision	Recall	F-measure
Business Intelligence	147	10	8	2	0	95.45%	100%	97.67%
Healthcare	90	24	220	10	30	95.65%	88.00%	91.66%

5. Conclusions and Future work

In this research , we provided 147 web services that can be composed automatically into workflows used in BI application. The proposed method can replace an input workflow with another workflow output and provide a backup web service on the web service composition automatically. The overall accuracy of the proposed method is better than [11], which is 97.67% compared to 91.66%.

The future research can be done by adding a Quality of Service (QoS) requirements and a facility to convert workflows to xml-based graphs. Thus, users can choose workflows that are faster, safer, cheaper, and easier to access.

Acknowledgments

We would like to thank Institut Teknologi Sepuluh Nopember and Dian Nuswantoro University for supporting the research.

References

[1] F. S. Hsieh and J. B. Lin, “A self-adaptation scheme for workflow management in multi-agent systems”, *J. Intell. Manuf.*, Vol. 27, No. 1, pp. 131–148, 2016.

[2] A. De Renzis, M. Garriga, A. Flores, A. Cechich, C. Mateos, and A. Zunino, “A domain independent readability metric for web service descriptions”, *Comput. Stand. Interfaces*, Vol. 50, pp. 124–141, 2017.

[3] Eclipse, “BPEL Designer Project,” 2018. [Online]. Available:

https://www.eclipse.org/bpel/. [Accessed: 01-Aug-2020].

[4] BPMN, "Object Management Group Business Process Model and Notation", 2019. [Online]. Available: http://www.bpmn.org/. [Accessed: 01-Aug-2020].

[5] M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott, S. Mcilraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic Markup for Web Services", *W3C Member Submission*, 2004. [Online]. Available: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/. [Accessed: 16-Jan-2020].

[6] H. H. Wang, N. Gibbins, T. R. Payne, and D. Redavid, "A formal model of the Semantic Web Service Ontology (WSMO)", *Inf. Syst.*, Vol. 37, No. 1, pp. 33–60, 2012.

[7] J. Köpke, "Annotation paths for matching XML-Schemas", *Data Knowl. Eng.*, Vol. 122, No. February 2017, pp. 25–54, 2019.

[8] G. Zou, Y. Gan, Y. Chen, and B. Zhang, "Dynamic composition of Web services using efficient planners in large-scale service repository", *Knowledge-Based Syst.*, Vol. 62, pp. 98–112, 2014.

[9] J. Puttonen, A. Lobov, M. A. Cavia Soto, and J. L. Martinez Lastra, "Planning-based semantic web service composition in factory automation", *Adv. Eng. Informatics*, Vol. 29, No. 4, pp. 1041–1054, 2015.

[10] D. Elsayed, E. Nasr, A. El Din Ghazali, and M. Gheith, "PGAQK: An adaptive QoS-aware Web Service Composition approach", *International Journal of Intelligent Engineering and Systems*, Vol. 11, No. 4, pp. 231–240, 2018.

[11] N. Arch-int, S. Arch-int, S. Sonsilphong, and P. Wanchai, "Graph-Based Semantic Web Service Composition for Healthcare Data Integration", *J. Healthc. Eng. 2017*, Vol. 2017, 2017.

[12] F. Zhang, Q. Zeng, H. Duan, and C. Liu, "Composition context-based web services similarity measure", *IEEE Access*, Vol. 7, pp. 65195–65206, 2019.

[13] R. Sarno, W. W. Ayu, A. N. Fajrin, D. Manfaat, M. S. Arif, and I. Baihaqi, "Context sensitive grammar for composing business process model variants", In: *Proc. of 2015 Int. Conf. Sci. Inf. Technol. Big Data Spectr. Futur. Inf. Econ. ICSITech 2015*, pp. 53–57, 2016.

[14] C. S. Wahyuni, K. R. Sungkono, and R. Sarno, "Novel Parallel Business Process Similarity Methods based on Weighted-Tree Declarative Pattern Models", *International Journal of Intelligent Engineering and Systems*, Vol. 12, no. 6, pp. 236–248, 2019.

[15] K. R. Sungkono, U. E. N. Rochmah, and R. Sarno, "Heuristic linear temporal logic pattern algorithm in business process model", *International Journal of Intelligent Engineering and Systems*, Vol. 12, No. 4, pp. 31–40, 2019.

[16] P. Wang, Z. Ding, C. Jiang, M. Zhou, and Y. Zheng, "Automatic Web Service Composition Based on Uncertainty Execution Effects", *IEEE Trans. Serv. Comput.*, Vol. 9, No. 4, pp. 551–565, 2016.

[17] B. Harjo, R. Sarno, and S. Rochimah, "Web Service Similarity with Standardized Descriptions", In: *Proc. of 2015 Int. Conf. Sci. Inf. Technol. Big Data Spectr. Futur. Inf. Econ. ICSITech 2015*, pp. 115–120, 2015.

[18] T. P. Liang and Y. H. Liu, "Research Landscape of Business Intelligence and Big Data analytics: A bibliometrics study", *Expert Syst. Appl.*, Vol. 111, No. 128, pp. 2–10, 2018.

[19] Sisense Inc, "2019 Gartner Magic Quadrant for BI and Analytics | Sisense", 2019. [Online]. Available: https://www.sisense.com/de/gartner-magic-quadrant-business-intelligence/. [Accessed: 01-Aug-2020].

[20] E. Loukis, M. Janssen, and I. Mintchev, "Determinants of software-as-a-service benefits and impact on firm performance", *Decis. Support Syst.*, Vol. 117, No. October 2018, pp. 38–47, 2019.

[21] S. Dutton, C. Marnay, W. Feng, M. Robinson, and A. Mammoli, "Moore vs. Murphy: Tradeoffs between complexity and reliability in distributed energy system scheduling using software-as-a-service", *Appl. Energy*, Vol. 238, No. January, pp. 1126–1137, 2019.

[22] I. Van De Weerd, I. S. Mangula, and S. Brinkkemper, "Adoption of software as a service in Indonesia : Examining the influence of organizational factors", *Inf. Manag.*, 2016.

[23] N. U. Ain, G. Vaia, W. H. DeLone, and M. Waheed, "Two decades of research on business intelligence system adoption, utilization and success – A systematic literature review", *Decis. Support Syst.*, Vol. 125, No. July, p. 113113, 2019.

[24] G. Salton and C. Buckley, "TERM-WEIGHTING APPROACHES IN AUTOMATIC TEXT RETRIEVAL", *Inf. Process. Manag.*, Vol. 24, No. 5, pp. 513–523, 1988.

[25] J. Ye, "Vector similarity measures of simplified neutrosophic sets and their application in

multicriteria decision making", *Int. J. Fuzzy Syst.*, Vol. 16, No. 2, pp. 204–211, 2014.

[26] B. T. McInnes and T. Pedersen, "Evaluating measures of semantic similarity and relatedness to disambiguate terms in biomedical text", *J. Biomed. Inform.*, Vol. 46, No. 6, pp. 1116–1124, 2013.

Appendix A:

TF-IDFwithCosineSimilarityMeasure algorithm.

TF-IDFwithCosineSimilarityMeasure Algorithm: Analyzing the relationship between a phrase/sentence and a collection of documents.

Input: ArrayQI and ArrayD
Output: WebServiceList1

1:**Procedure** ***TF-IDFwithCosineMeasure(ArrayQI,ArrayD)***
2: max=Maximum number of *ArrayD* elements
3: **for all** m ∈ *ArrayQI* **do**
4: **for all** a <*max* **do**
5 : *Initialized empty Array to ArrayQ,ArrayTerm,TermQD,ArrayDF,ArrayWDT,ArrayIDF,ArrayWD_WDi,Sum_ArrayWD_WDi,*VectorLength,SumVectorLength,SQRTSumVectorLength, *DocId,CosineMeasure*
6: *ArrayQ=split* $ArrayQI_m$ *with ","*
7: **for all** b ∈ *ArrayQ* **do**
8: $ArrayQ_b$ is stemmed, *then* $ArrayQ_b = B\{StemmedTex\}$
9**:** *Adding ArrayTerm element with* $ArrayQ_b$
10: **end for**
11**:** **for all** b ∈*ArrayD* **do**
12: $ArrayD_b$ is stemmed, *then* $ArrayD_b = B\{StemmedText\}$
13**:** *Adding ArrayTerm element with* $ArrayD_b$
14: **end for**
15: **for all** b ∈$ArrayTerm$ **do**
16: **for all** c ∈ *ArrayQ* **do**
17**:** *SimValue=similarity*(*ArrayTerm*$_b$, *ArrayQ*$_c$)
18: *Adding* TermQD *element withSimValue*
19: **end for**
20: **end for**
21: **for all** b ∈$ArrayTerm$ **do**
22: **for all** c ∈ *ArrayD* **do**
23**:** *SimValue=similarity*(*ArrayTerm*$_b$, *ArrayD*$_c$)
24: *Adding* TermQD *element withSimValue*
25 **end for**
26: **end for**
27: **for all** b < $n(ArrayTerm)$ **do**
28: TotalArrayDF=0.0
29: **for all** c < n(*ArrayD)* **do**
30**:** ***if***((*Array*TermQD$_{cb}$ ≥ 0.8) then
31: $TotalArrayDF = TotalArrayDF + 1.00$;
32: **end if**
33: *TotTermQD++;*
34: $ArrayIDF_b$ *=Math.log10(TotTermQD /TotalArrayDF)*
35: **end for**
36: **end for**
37: **for all** b < $n(ArrayTerm)$ **do**
38: **for all** c <(n(*ArrayD)+1)***do***
39: *Array*WTD$_{cb}$ = $TermQD_{cb}$ *x* $ArrayIDF_b$
40: **end for**
41: **end for**
42: **for all** b < $n(ArrayTerm)$ **do**
43: **for all** c <(n(*ArrayD)+1)***do***
44: *Array*WD_WDi$_{(c-1)b}$ = *Array*WTD$_{0b}$ * *Array*WTD$_{cb}$
45: **end for**
46: **end for**
47: **for all** b <(n(*ArrayD)+1)***do***
48: msum=0.0
49: **for all** c < $n(ArrayTerm)$***do***
50**:** *msum=msum+* *Array*WD_WDi$_{cb}$
51: **end for**
52: *Sum_Array*WD_WDi$_c$ = *msum*
53: **end for**
54: **for all** b < $n(ArrayTerm)$ **do**
55: **for all** c <(n(*ArrayD)+1)***do***
56**:** $VectorLength_{cb}$ =*Array*WDT$_{cb}$***x*** *Array*WDT$_{cb}$
57: **end for**
58: **end for**
59: **for all** b <(n(*ArrayD)+1)***do***
60: msum=0.0
61: **for all** c < $n(ArrayTerm)$***do***
62**:** *msum=msum+* $VectorLength_{cb}$
63: **end for**
64: $Sum\_VectorLength_c$*=msum*
65: **end for**
66: **for all** b <(n(*ArrayD)+1)***do***
67**:** $SQRTSumVectorLength_b$ *=Math.sqrt(*$Sum\_VectorLength_c$*)*
68: **end for**
69: *x=0.0*
70: *y=0.0*
71: **for all** b <(n(*ArrayD)+1)***do***
72: x=Sum_ArrayWD_WDi$_c$
73: y=($SQRTSumVectorLength_0$ x $SQRTSumVectorLength_{c+1}$
74**:** $ArrayCosineMeasure_b$*=x/y*

```
    end for
    text=""
    for all b <(n(ArrayCosineMeasure))do
        if( ArrayCosineMeasure_b>0.8) then
            text=
                text+','+string( ArrayCosineMeasure_b)
        end if
    end for
    adding WebServiceList1 element
 end for
end for
End Procedure
```

Appendix B:

AutomaticWSComposition algorithm.

AutomaticWSComposition Algorithm**:** Automatic web service compositionusing TF-IDF and cosine similarity measure

Input: InputName Parameters and OutputName Parameters

Output: A workflow that corresponds to InputName Parameters and OutputName Parameters

```
Data=read(upload table {WebSeviceId,Userid,WebServiceName, WebServiceAddress,Input-Name,OutputName,InputType OutputType})
Initialized ArrayWsId = Data_0 , ArrayWsAddress =Data_2, ArrayIN =Data_4, ArrayON =Data_5
Initialized Empty Array to WebServiceList1, WebServiceList2, WebServiceList3,ArrayD,ArrayDWs,ONNotFound, Result1,Result2,FinalResult
ArrayQI= split InputName text with ","
ArrayQN= split OutputName text with ","
for all a ∈ArrayIN do
    ArrayArrayIN=split ArrayIN_a text with ","
    if n(ArrayQI)=n(ArrayArrayIN )then
            ArrayD_a= ArrayIN_a
            ArrayDWs_a= ArrayWsId_a
    end if
end for
TF-IDFwithCosineSimilarityMeasure(ArrayQI,ArrayD)
if(n(WebServiceList1)==0) then
    print "Result of Matching InputNames with web service metadata is Null"
else then
    Result1= WebServiceList1;
    ArrayQI= ArrayQN
    ArrayD= WebServiceList1;
    TF-IDFwithCosineSimilarityMeasure (ArrayQI,ArrayD)
    if (n(WebServiceList1==0) then
        print "Result of Matching OutputNames with WebServiceLis2 is Null"
    else then
        for all a <n(WebServiceList1 )do
            if ( WebserviceLis1_a = "") then
                ONNotFound= WebserviceLis1_a
            end if
        end for
        ArrayQI= ONNotFound
        ArrayD= ArrayWsIdelement but not WebServiceList1 element
        TF-IDFwithCosineMeasure (ArrayQI,ArrayD)
        Result2= WebServiceList2;
        Data=read(findingresult table {WorkflowId,OutputName })
        Initialized ArrayWorkflowId= Data_0,ArrayONResult=Data_1
        ArrayResult2=split Result2 with ","
        for all a <n(ArrayResult2) do
            for all b <n(ArrayONResult) do
                if(ArrayResult2_a == ArrayONResult_b) then
                    ArrayResult2_a= ArrayResult2_a + "||" +String(ArrayONResult_a
                end if
            end for
    end for
Insert(findingresult table {CompositionId,CompositionName,InputName, OutputNameEffect,OutputName,Workflow })
```