sustainability

MDPI

Article

WSFeIn: A Novel, Dynamic Web Service Composition Adapter for Cloud-Based Mobile Application

R. Kanesaraj Ramasamy [1,*], Fang-Fang Chua [1], Su-Cheng Haw [1] and Chin-Kuan Ho [2]

[1] Faculty of Computing Informatics, Multimedia University, Cyberjaya 63100, Malaysia
[2] School of Computing, Asia Pacific University of Technology and Innovation, Kuala Lumpur 57000, Malaysia
* Correspondence: r.kanesaraj@mmu.edu.my

Abstract: Service-Oriented Computing (SOC) has been a cornerstone development in today's fast-paced world, covering the lifecycle of services and contributing to service delivery through distributed applications. SOC is striving to integrate cloud computing with complicated mobile apps. Dynamic and adaptable web service composition is the tip of the iceberg for SOA adoption. Dynamic service binding is vital for mobile computing due to the need for distributed mobile internet consumption at runtime. This study addresses SOC difficulties associated with web service composition, whose growth creates a paradigm change in identifying data type matching solutions. This allows for data type-level matching research to ensure high-quality web service creation. In this work, the composition process is divided into three phases: web service discovery, web service selection, and web service composition, where web service personalization and workflow reliability are emphasized. The final result is a complicated mobile app that runs without depleting device resources and an adaptable, reusable web service composition workflow. This improves matching at the data type level, an SOC pain point.

Keywords: web services; WSFeIn; web service composition; mobile applications

check for updates

Citation: Ramasamy, R.K.; Chua, F.-F.; Haw, S.-C.; Ho, C.-K. WSFeIn: A Novel, Dynamic Web Service Composition Adapter for Cloud-Based Mobile Application. *Sustainability* **2022**, *14*, 13946. https://doi.org/10.3390/su142113946

Academic Editors: Manuel Fernandez-Veiga and Zubair Baig

Received: 31 August 2022
Accepted: 21 October 2022
Published: 27 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

As the new era calls for excellence, web service (WS) has been a matter of interest for all as a platform for massive evolution from traditional web service into leaving imprints in cloud computing territory. Many researchers have shown interest in web service composition (WSC) which creates pathways by being the middleman, thus interconnecting web services for greater discoveries. Several WSs are coordinated in order to achieve a business requirement which results in a complex application. The process of WSC comes with several rooms for failure which include unavailability, poor performance, duplication in the functionalities, and bugs. Quality of service (QoS) becomes the milestone to select the best WS. When quite a few web service providers are available for composition, quality of service properties ties such as execution time, cost, or availability are considered vital in providing a clincher, leading towards the creation of QoS-aware composite WS [1,2].

The flexibility of dynamic binding of services is important in mobile computing where services are distributed, rendered, and consumed at the run time. Therefore, this paper aims to propose an adaptive and dynamic web service composition that can be implemented on the cloud-based mobile application. Through this implementation, the proposed composition method is able to run complex mobile applications in all sorts of mobile devices operating in real-time or dynamic environments.

On the other hand, the majority of existing researches mainly focus on reliability, efficiency, and testing web services. However, research has been idle on testing the reliability and efficiency of web service composition workflow [3–5]. Ensuring reliability in web service composition enhances the system workflow or business process, and indirectly boosts confidence in the business process. The abundance of the research [6] has been

carried out on a normal web and stand-alone system to investigate how business process can be improved. However, the research is limited on how adaptive WS can be implemented for mobile applications.

In a nutshell, this paper contributes the following:

1. Web service point (WSPoint) serves as a helping hand for users or companies to create their personalized web service composition based on QoS attributes.
2. An adapter, known as WSFeIn, facilitates checking the availability of the web service used for web service composition. If the web service is not available, then it chooses the next web service which has been proposed by the web service discovery phases and matches the data type dynamically.
3. An adaptive (WSFeIn) composition in a cloud environment is deployed and tested. Therefore, a cloud-based mobile application that uses WSFeIn consumes fewer resources and overcomes traditional mobile application deficiency that consumes more resources to run their applications.

The rest of the paper is organized as follows. Section 2 discusses phase 1: web service discovery (WSD), while Section 3 covers phase 2: web service selection (WSS) and web service composition (WSC). Section 4 includes testing and validation for all three main phases. Section 5 demonstrates the implementation of the architecture in a case study. Finally, the conclusion in Section 6 presents the overall summary and future work.

2. Web Service Discovery

Web service discovery stands as the sole function in determining foreseen services and appeasing the targeted consumers. The focal point here is to improve the searching mechanism. In WSD, WS from World Wide Web (WWW) is collected to build the cloud-based repository. This phase focuses on non-functional requirements as the functional requirement shall be focused on in the WSS phase. The WSD stage includes building a web service repository and customization using WSPoint.

Building Web Service Repository

There are several open web service repositories available in the WWW, including WebSeviceX.Net and Xmethods. These repositories are made available through methods that function on top of the WSDL URL through the website or expose it using UDDI methods. This opens the gateway for users to search through and choose the web service for the WSC process. Here, the aim is to filter WS using the non-functional requirement. However, one has to keep in mind that this is not a dynamic web crawler but rather a manual web service collector obtained from the developer. We are using a fixed dataset obtained from the Web Service Research Centre from The Chinese University of Hong Kong. The fixed dataset eases our testing procedure for our composition workflow in terms of efficiency and reliability. The searching process is streamlined through non-functional requirements by assuming all the web service information is pre-stored in the local repository.

A good quality WS offshoots before the process of matching search for functional requirements through this. On the contrary, if WS selection is preceded based on the functional requirement, followed by elimination of bad quality web services, this will burn in daylight as the number of search datasets that appear in the search space shall be larger. Therefore, good quality web service is determined by filtering the web service based on the functional requirement in WSC to enhance its performance.

Non-functional requirements in web service composition are based on quality of services (QoS). It is crucial to deliberate QoS non-functional requirements with a functional requirement in the web service discovery process. Seven quality attributes are cultivated in this work: availability, accessibility, integrity, performance reliability, regulations, and security. These attributes adhere to IBM's definition.

Non-functional requirements in web service composition are based on quality of services (QoS). It is crucial to deliberate QoS non-functional requirements with a functional

requirement in the web service discovery process. There are seven quality attributes including reliability, regulatory, and security. These attributes adhere to IBM's definition. According to [7–9] respectively, QoS is used to retrieve attributes that are used to calculate QoS points in various ways and the average of the total value is determined. We propose to rank selected QoS quality attributes by a certain weight, and it is more significant compared to the method using the ranking of QoS values with all same weight.

In Equation (1), WSPoint is calculated by assigning weightage to the QoS criteria.

β = Web service point (WSPoint)

λ = Quality of service (QoS) attribute value a, b, c = Weightage based on percentage

$$\beta = \lambda 1(a\%) + \lambda 2(b\%) + \lambda 3(c\%) \quad (1)$$

Once the rank is replaced with on-par QoS criteria, the WSPoint changes and triggers a rearrangement in the ranking of the web service. This is vital for the web service composition phase later as every domain of web service composition implemented has different requirements. For example, an early warning system gives more weightage to the availability of web service and also the reliability of a web service. At the same time, domains that require a large volume of transactions such as flight booking systems give more weightage to the performance of service or response time of a web service. Similarly, the entertainment industry (music or movies) requires high availability for most of the user-rated web services where the response time needs to be low. Since each domain has a different priority of QoS, assigning a weightage in the WSPoint calculation ensures the web service composition workflow meets the domain or customer needs (Figure 1).

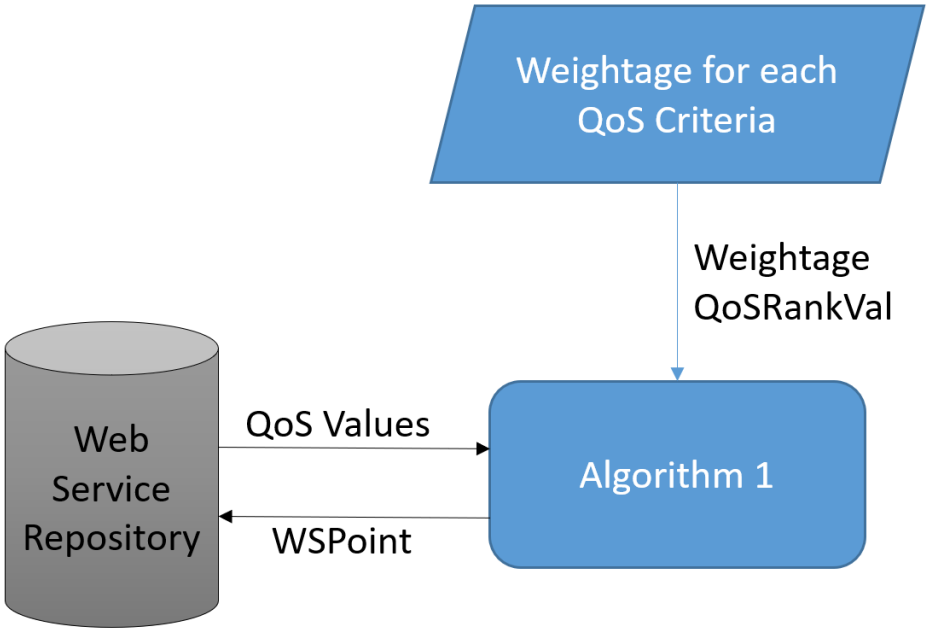


Figure 1. WSPoint calculation process.

Using Equation (1), the QoS attributes and the weightage for each QoS criteria are derived. For each of the QoS quality attribute, for example, performance, is extracted from the database where the response time is stored. Based on the QoS values and weightage set for QoS criteria, WSPoint is calculated. The steps are repeated every three days to ensure all the WSPoint are up to date. The three-day interval enables the test to work at its best at least twice a week. The more often it is carried out, the more accurate it is, but it becomes more costly (Algorithm 1).

Algorithm 1 WSPoint for WS in Web Service Repository

Input: Request for QoS Attribute Values and QoS Ranking Percentage
Output: WSPoint

1. QoS attributes value store in DB (values are updated every 3 days once)
2. QoSRankVal [3] = Weightage for each QoS attribute
3. For each service
4. For each QoS attribute
5. Top 3 QoS values retrieved from DB
6. TotalQoS = (QosValue1 + QosValue2 + QosValue3)
7. End for each
8. AverageQoS = $(\frac{TotalQoS}{3})$
9. WSPoint = ((AverageQoS [0] $\times$ $\frac{QoSRankVal[0]}{100}$) $\times$ 100%) + ((AverageQoS [1] $\times$ $\frac{QoSRankVal[1]}{100}$) $\times$ 100%) + ((AverageQoS [2] $\times$ $\frac{QoSRankVal[2]}{100}$) $\times$ 100%)
10. Store WSPoint in DB
11. End for each
12. Receives the list of web service match keyword from DB and sort in WSPoint ascending order

In Algorithm 1, a timer system is used to obtain the QoS quality attributes, availability, and response time every 3 days (line 1). Tracking QoS in WS helps to safeguard the state of the web service. The weightage for each QoS attribute is accepted by the developer (line 2). For each web service, the latest three records for each of the QoS attributes are collected (line 4) and added altogether (line 6). This is followed by the calculation of the average QoS attribute (line 8). Based on the average, the WSPoint for each web service using Equation (1) is computed (line 9). The WSPoint is gathered and stored in the database (line 10). When a user requests WS using a keyword, the search will ignore the entire web service which is not available and match the keyword to the method name based on the REGEX expression. Subsequently, the search result will be sorted in ascending order based on WSPoint values. These aids in retrieving the fastest response time and best-rated web service shall have a smaller number.

Once the rank is replaced with on-par QoS criteria, the WSPoint changes and triggers a rearrangement in the ranking of the web service. This is vital for the web service composition phase later as every domain of web service composition implemented has different requirements. For example, an early warning system gives more weightage to the availability of web service and also the reliability of a web service. At the same time, domains which require a large volume of transactions such as flight booking systems give more weightage to the performance of service or response time of a web service. Similarly, the entertainment industry (music or movies) requires high availability for most of the user rated web service where the response time needs to be low. Since each domain has a different priority of QoS, assigning a weightage in the WSPoint calculation ensures the web service composition workflow meets the domain or customer needs. Using Equation (1), the QoS attributes and the weightage for each QoS criteria are derived. Each QoS quality attribute, for example, performance, is extracted from the database where all the response time is stored. Based on the QoS values and weightage set for the QoS criteria, WSPoint is calculated. The steps are repeated every three days to ensure all the WSPoint is up to date. The three days interval enables the test to work at its best at least twice a week. The more often it is carried out, the more accurate it is but it also becomes more costly.

3. Dynamic Web Service Selection and Adaptive Web Service Com-Position

3.1. Web Service Selection

Web service selection plays an undeniable role in web service composition. This process imposes a direct impact on the quality of composition service. As a subject of interest, many researchers have gravitated towards focus on quality of service (QoS) web

service selection in recent years. According to [10], researchers have presented many algorithms based on integer programming (IP), mixed integer linear pro-programming (MILP), multi-dimension multi-choice 0–1 knapsack problem (MMKP), Markov decision programming (MDP), genetic algorithm (GA), and particle swarm optimization (PSO), and have attempted to address problems in their research. However, the emphasis on QoS is highlighted in our first phase –WSD.

In this phase, the elbow room is to filter WS outlined in phase 1 based on the keyword criteria provided by the user. Keywords are specified based on the method name in each web service as recommended by [11]. Generally, searching a description or web service name for functional requirements without a filter criterion is not feasible as the possible number of search choices can be large. Developers rarely include a description for the web service published in free repositories and service names are general and each service can have more than one method. Therefore, to obtain an on the dot search result, an on target method has to be included to aid in the filtration of the keywords and it is based on its functionality.

To search through methods in the web service repository, regular expression (REGEX) is used to express the filter criteria. REGEX is a sequence of characters that define a search pattern, mainly used in pattern matching with strings [12]. REGEX is useful in computing and has emerged to provide both basic and extended standards for grammar and syntax; modern REGEX heavily augments the standard. REGEX processors are found in several search engines, search and replace dialogs of several word processors, and text editors, and in the command lines of text processing utilities [12]. By specifying keyword filter criteria via REGEX, the processors are able to efficiently select the relevant keyword method to obtain the result matching the REGEX.

Figure 2 describes a case where a user is required to provide a keyword which is the name of the desired function such as “SMS”, “address”, and “weather”. Then, the system searches through the web service repository using the REGEX and generates an XML file with search results. This XML file will be used in the selection unit. The XML file for the selection unit works as the placeholder to hold the matching criterion between the user input and the matched strings. As such, the adaptor need not search through the entire list of search keywords. Consequently, the search time needed for the adaptor is reduced.

Using Algorithm 2, an XML is generated using the following nodes. The lists of nodes that are obtained are shown in Table 1. Using Algorithm 2, user input is obtained from the developer (line 1). A filtration process is conducted using REGEX (line 2) and saved in a LIST data type format (line 3). Each method is invoked to retrieve the input and output data type (line 4).

Table 1. Description for Web Service Selection XML nodes.

Node	Description
wsURL	Web Service URL
wsID	ID for the web service in local cloud database (unique ID for each web service)
methodName	Method name as uploaded by developer
Outputs	Output data type for the selected method
Inputs	Number of inputs
Input	Input data type for the selected method
seq	Sequence ID where the WS are sorted based on WSPoints.

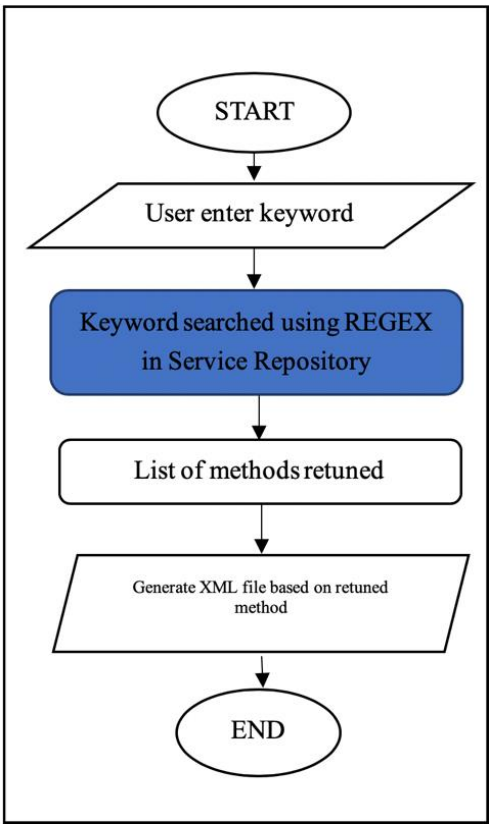


Figure 2. Flowchart on searching methods.

Algorithm 2 Search Keyword saved in XML File

Input: User Keyword
Output: XML File with Method Information

1. Obtain user input
2. Read all method names and filter using REGEX()
3. List<string> methodname
4. Each method is invoked to retrieve input and output data type
5. Call GenerateXML()
6. GenerateXML(){
7. Create new XML
8. Start writing XML
9. For each method
10. Write service name
11. Write webservice URL
12. Write method name
13. Write output data type
14. Write inputs data type
15. Write sequence ID
16. End for each
17. Save XML
18. }

Next, the generate XML function is called (line 5) to create an XML file with service name, web service URL, method name, output, input data type, and the sequence of web service (line 10–15) (see Figure 3). In the final step, the XML file is saved in the server (line 17). The purpose of storing is to save time in the web service composition phase as repeated requests need not be searched from the beginning. Thus, the stored file functions as a "cache" and retrieves the method and obtains the scheme information when needed.

This action can speed up the performance of the adapter in a real-time environment to replace a failed web service with same functionality.

```
<?xml version="1.0" encoding="utf-8"?>
<selectedService>
  <service>
    <wsURL>http://seq1.azurewebsites.net/Service.asmx?WSDL</wsURL>
    <wsID>2</wsID>
    <methodName>updatemap_string</methodName>
    <output>System.String</output>
    <inputs>
      <input>System.Int32</input>
      <input>System.String</input>
    </inputs>
    <seq>0</seq>
  </service>
  <service>
    <wsURL>http://seq1.azurewebsites.net/Service.asmx?WSDL</wsURL>
    <wsID>1</wsID>
    <methodName>updatemap_int</methodName>
    <output>System.String</output>
    <inputs>
      <input>System.Int32</input>
      <input>System.Int32</input>
    </inputs>
    <seq>1</seq>
  </service>
</selectedService>
```

Figure 3. Shows the sample XML file generated using the search string "updatemap".

3.2. Web Service Composition Adapter

Service composition can be executed by composing elementary or composite services. In composing web services, the sequence of events behind any business process is implemented by a few web services. This leads to workflow management, where the logic is comprehended by composing autonomous applications. These composite services can in turn be recursively composed with other WS into higher-level solutions. Furthermore, the recursive composition feature allows the development of new solutions rapidly by reusing existing business services. This form of reusability is one of the most significant features of SOA. Web service composition creates a stepping stone for enterprises, increasing their business agility to frequent changes in user requirements by integrating existing business services [13].

Although web service composition is beneficial to many domains, it also poses some limitations that can lead to failure of the composition. A web service composition becomes a failure if the web service used for the composition is not available. Some of the methods as presented in web service discovery can be used to replace a web service in composition. Each method or adapter is developed to solve this problem capable of selecting the next best web service from the repository. Among the issues of concern is the requirement matching, where the selected web service does not match with the current requirement in the composition workflow or QoS issues of the selected web service are not of the right quality. Good quality WS are characterized by their high availability. This can ensure the web service composition workflow does not fail. In this paper, attention is given to the data type matching issue when a new web service is selected to replace a failed web service in the web service composition. When a web service is selected based on the requirement but does not match on data type, it will still not work in the workflow. To address this issue an adapter, as shown in Figure 4, can be placed in between the workflow and selection unit to ensure the web service selected by the selection unit matches their composition workflow.

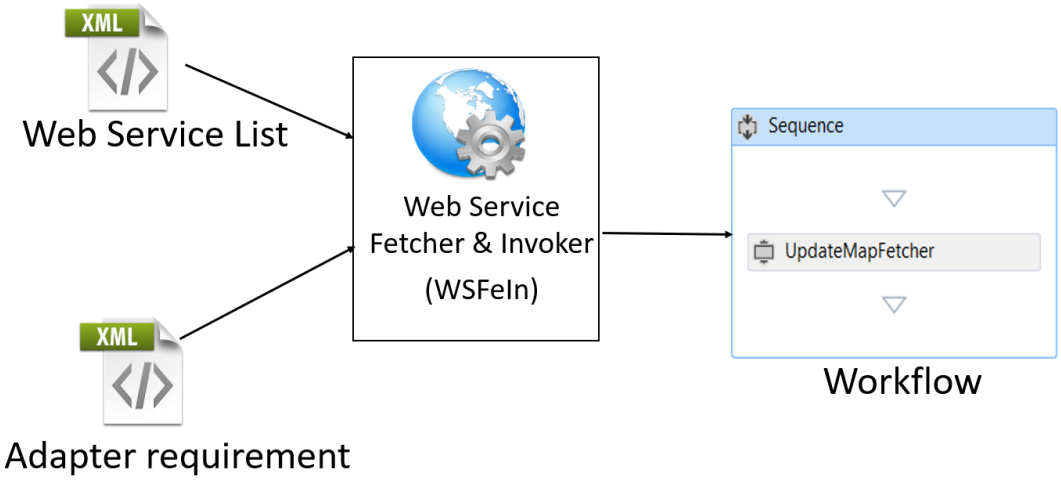


Figure 4. Web Service Fetcher and Invoker.

The term "Web Service Fetcher and Invoker" describes the function of this adapter (WSFeIn). WSFeIn can retrieve the web service from the selection unit's XML file and then use method invocation to check the web service's availability before service composition. Two distinct XML files can be obtained from the unit of selection. In Figure 3, we see an example of the first type of XML file, which contains a list of all the WS that were considered but were ultimately eliminated from further consideration. The second kind of XML file is one that the user has modified by selecting the data type needed to return from the web service via the administrator system. As seen in Figure 4, the workflow process is requesting for a string to operate on, but the web service is only returning integers. After that, the administrator can make the necessary changes, and the XML customization file will be updated automatically. WSFeIn will then take a look at the requirement file and set up the web service return data type.

The WSFeIn adapter is created as a web service and hosted in the admin server. In this case, the adapter is hosted in Microsoft Azure as a cloud service. Algorithm 3 explains the way this adapter is developed to work in between the selection and composition unit. In this work, the selection and composition are narrowed to string and integer. Other data types such as Boolean, Char, Long, and Float use the same method to convert or match the data type. In Algorithm 3, the process begins by reading the XML file (line 1). The input nodes (line 3) are read followed by the output node. In line 7, a ping test is conducted to ensure the web service is available to use, or else the next available web service is collected using the while loop (line 2 and line 8). The input and output node information is extracted from the XML file. In the ping test, the WDSL file is downloaded to test and ensure the WSDL file is available (line 23). In line 10, the invoke function is triggered to call the first function in the XML file. The return value is stored in a variable (return values). In line 4, based on the output data type obtained from the output node, the variable is converted to that corresponding data type (line 12 and line 16) and returned to the workflow.

Our formalization of WSFeIn, which is built on top of the web service choreography interface, is based on the grammar presented in [14]. (WSCI). In WSCI, you may define things like conditional and looped executions, as well as simultaneous executions of activities. In addition, [14] also suggests using process algebra for formalizing WSCI. An example sourced from the author is shown below.

```
Exc = context {
            Process P
Exceptiom{
onTimeout t to Q
            }
}
```

Will be translated to:

[[Exc]] (begin,end) = [[P]] (begin,raised,cTO)
| | Timer (eoTO,cTO,end)
| | eoTO?().raised!().catch!().0
| |[[Q]](catch,end)

Figure 5 shows a formal definition for the process of WSFeIn. Based on the above example by [14], we defined the flow of the WSFeIn and followed by the formalized definition.

A = web service list
b = selected web service from the web service list
c = the process after the web service is invoked
d = adapter requirement for the next process
x = availability of service in workflow composition

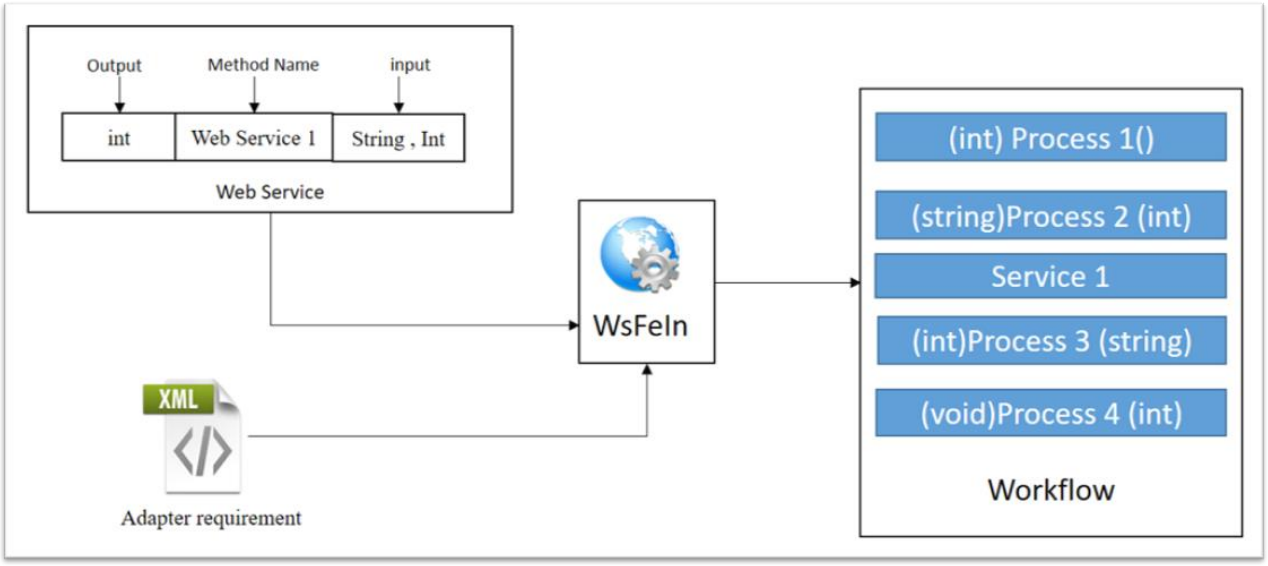


Figure 5. How WSFeIn works.

Definition 1. *If (x==0) then*
web service is the web service composition workflow is not available if (x==1) then
web service is the web service composition workflow is available).

Definition 2. *boutput = cinput*
if boutput is same with cinput datatype then
selection process end and web service used for web service composition.

Definition 3. *boutput f = cinput*
$\sum_{x=0} [f(y)] = d(c)$
if boutput is not same with cinput datatype then
the boutput will be converted to cinput based on the adapter requirement.

Based on the definition above, we created the following formal definition to show how WSFeIn works within the workflow. Referring to the above formalization, the workflow will perform the request to check the availability of web service. If the web service is not available (0), then the adapter (A) will be triggered to request a new web service. This web service will be checked for the datatype match with the workflow requirement. If the match fails, the data type will be converted to the desired datatype based on the workflow requirement.

```
WSWorkflow = workflow/service/request?().
Workflowservice/availability!().
(
A = WSFeIn/newService/request?().
WSFeIn/checkService/datatypematch?().
WSFeIn/checkService/reply!("match")
(
WSFeIn/convert?().
)
)
```

Referring to the above formalization, the workflow will perform the request to check the availability of web service. If the web service is not available (0), then the adapter (A) will be triggered to request a new web service. This web service will be checked for the datatype match with the workflow requirement. If the match fails, the data type will be converted to the desired datatype based on the workflow requirement.

Figure 6 shows three different scenarios of web service composition by [15]. In the first workflow, the WS is working in normal conditions with no issues reported but in workflow 2, there is one web service that fails during the composition triggering the failure of depended-on web service and the whole composition fails. As for workflow 3, there is an adapter standby for each service. If there is a failure in any web service, the dedicated adapter shall react to re-place the web service with a suitable one. In this project, all the web services presented are assumed to be of good quality as they are filtered based on functional requirements and sorted based on WSPoint. With this, the proposed adapter focuses only on data type matching while functional requirements and QoS issues are handled in earlier phases.

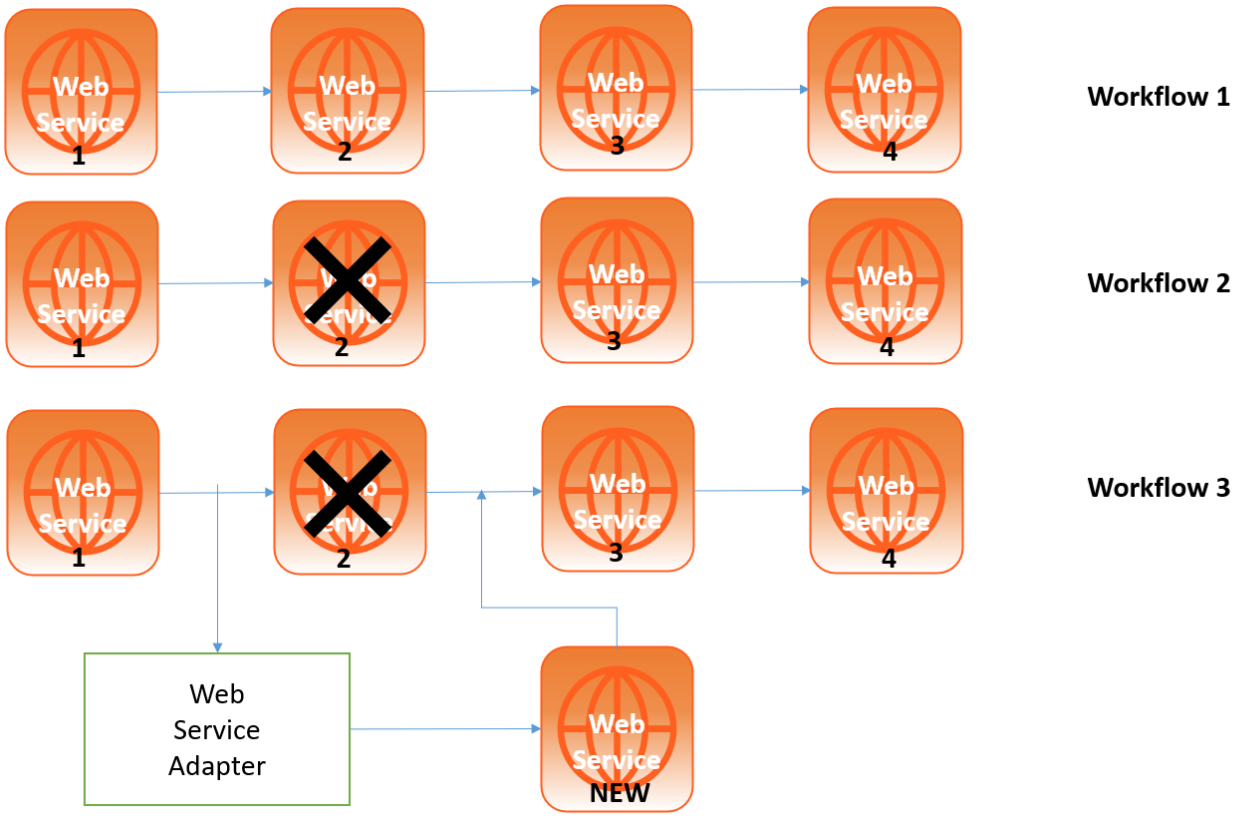


Figure 6. Different scenarios of web service composition.

Algorithm 3 WSFeIn Matching Algorithm

Input: Data type from Adapter Requirement XML
Output: Converted output data type

```
1.  Open and Read XML
2.  Begin Do
3.  Read NODE Inputs
4.  Var finalOutput = Read NODE Output
5.  Read input value from Workflow
6.  String URL = Read NODE wsURL
7.  Bool servicestatus = IsAddressAvailable (URL);
8.  End While (servicestatus==FALSE)
9.  Begin IF (servicestatus==true)
10. Var returnValues = invokeService()
11. If (finalOutput == "System.String") Then
12. returnValue = returnValue.ToString();
13. return returnValue
14. Else IF (finalOutput == "System.Int32") then
15. Integer a = 0;
16. a = Convert.ToInt32 (returnvalue);
17. return a;
18. End IF
19. End IF
20. public bool IsAddressAvailable (string address)
21. {
22. Begin Try
23. DownloadData (address)
24. return true;
25. End Try
26. }
```

3.3. Cloud-Based Mobile Application

Mobile applications comprise two types: traditional mobile application (TMA) and cloud-based mobile application (CBMA). TMA runs 100% on the phone resources. The CBMA application uses the cloud processing power and the mobile device as input and output devices. Both TMA and CBMA are tested to ascertain the claim that CBMA saves mobile device resources.

Tests are conducted in a set by running iterations from 10 loops to 1,000,000 loops for both types of applications. A complex mobile application is defined as one that uses high CPU usage and takes a longer time to complete a process. Algorithm 4 shows the method used in TMA to perform iteration (line 4–7). The user input (line 4) is a value used to submit and test the number of iterations.

Algorithm 4 Traditional Mobile Application

Input: Number
Output: Total Number of Loops

```
1.  Void main() {
2.  double Count = 0;
3.  double calculate = 0;
4.  Begin For Loop Condition(Count < User Input)
5.  Calculate = calculate +1
6.  Count ++
7.  End For Loop
8.  }
```

In TMA, all the inputs and outputs including processing are carried out within the mobile device itself. This causes the mobile device to have limited applications running or limited processing at one time.

The CBMA uses Algorithm 5, where only the end-point address is declared to call the web service and the device is used to retrieve the user input and call the web service to the iterations. In Algorithm 4, the iteration loops are run within the application itself, while in Algorithm 5, the iterations are carried out in the cloud environment (line 5). Both applications use the same input and return similar output, however, the CPU usage and time taken to complete the iterations are different as the resources available to process differs as Microsoft Azure cloud service offers 1 Gigabyte of storage. The CPU is shared allowing 60 min per day and 1-gigabyte random access memory (RAM). The server is located in South Central United States.

Algorithm 5 Cloud-Based Mobile Application.

```
Input: Number
Output: Total Number of Loops
1. Void main() {
2. double Count = 0;
3. double calculate = 0;
4. Define EndPoint For Web Service
5. Calculate = Call WebService (User Input) }
```

Using CBMA, the input is collected from the mobile device and sent for processing in the cloud. The output is sent to the mobile application from the cloud. With this, processing power increases, and more complex mobile applications can run on the device. The process uses the power of the CPU from the cloud while the device CPU is used only to retrieve user input and to display the output. Based on Algorithms 4 and 5, Table 2 shows the CPU usage and total response time required for the application to obtain the user input and return the result.

Table 2. Traditional mobile application versus cloud-based mobile application.

Number of Iterations	Traditional Mobile Application		Cloud-Based Application	
	CPU (%)	RT (Seconds)	CPU (%)	RT (Seconds)
10	37.87	3.886	34.23	2.784
100	37.87	4.184	34.23	2.894
1000	37.87	4.754	34.23	3.326
10,000	37.87	4.997	34.23	3.69
100,000	37.87	5.1	34.23	4.091
1,000,000	37.87	5.503	34.23	4.291
10,000,000	37.87	7.09	34.23	4.321
100,000,000	37.87	8.055	34.23	4.801

The total response time required by TMA is longer compared to CBMA (Figure 7). Although the CPU usage is higher than TMA, the time to complete the task is comparatively less. This shows that although the CPU usage is lower compared to CBMA, it takes a longer period of execution, which means more resources (e.g., CPU, RAM, and battery) are being drained. However, in CBMA, the CPU usage is high only for a short period which can translate to longer battery life. The resource usage closely correlates to the number of processes. Mobile applications such as early warning systems (EWS), which involve coverage of thousands of stations in charge of floods and landslides, will require higher

CPU processing power to retrieve all the station status at a given point especially when the mobile applications are required to refresh every 10 min in real-time.

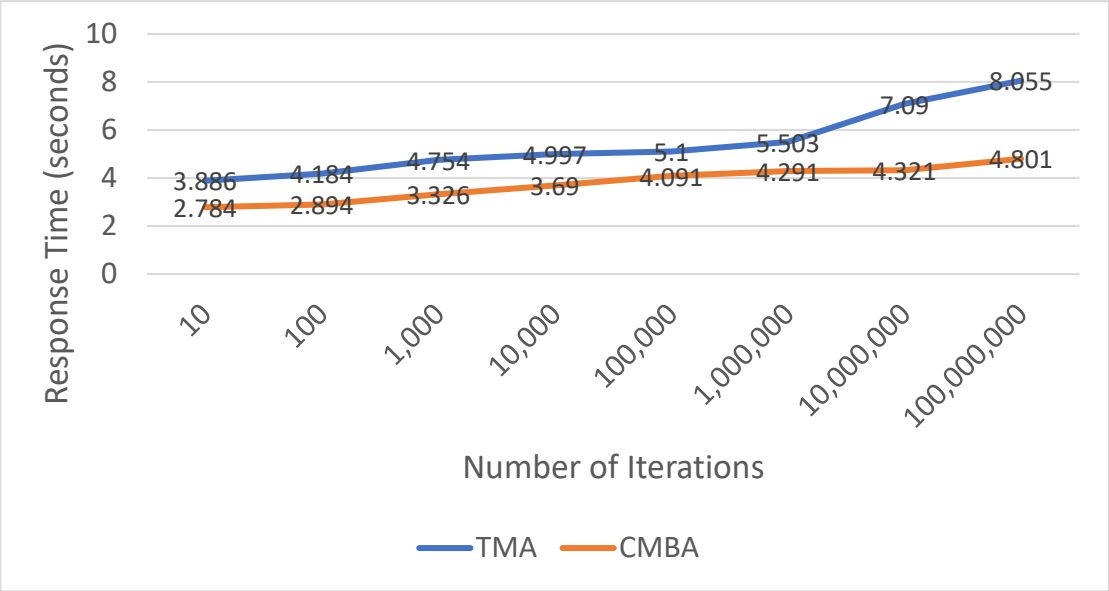


Figure 7. Comparison graph between TMA and CBMA.

4. Testing and Validation

To design this web service repository, a database is built based on the design as shown in Figure 8 to store all the web services. The extracted method from each web service is illustrated as shown in Figure 8. This table is built for QoS criteria of performance, availability, and user rating. An advantage of this database design is that it can be expanded based on the QoS criteria selected for the system implementation in the future.

Table Name: **servicetbl**

SID	ServiceLink	Pingstatus	servicePoints	usrRating	totaltime

Table Name: **methodtbl**

mid	SID	methodName

Table Name: **avaitbl**

avaiID	serviceID	avaiResult	avaiTime

Table Name: **usrRating**

ratingID	serviceID	Rating	ratingtime

Figure 8. Database schema for web service repository.

4.1. Comparison between Equal QOS Ranking and Weightage QOS Ranking

Three types of tests are carried out in this phase for comparison. The first test is on the difference between the result with the implementation of WSPoint using the basic method where only the QoS calculation is carried out with equal QoS weightage. For this test, a system where a search result is produced for the keyword "weather" is implemented. As shown in Figure 9a, all QoS criteria are equally calculated, and results are displayed. The same dataset is used with the WSPoint calculation algorithm and shown in Figure 9b. This

comparison was carried out to demonstrate that the result presented after using WSPoint is different compared to the equal weightage QoS calculation. The arrangement of the selected web service is critical for the next phase as the adapter follows the sequence of the recommendation.

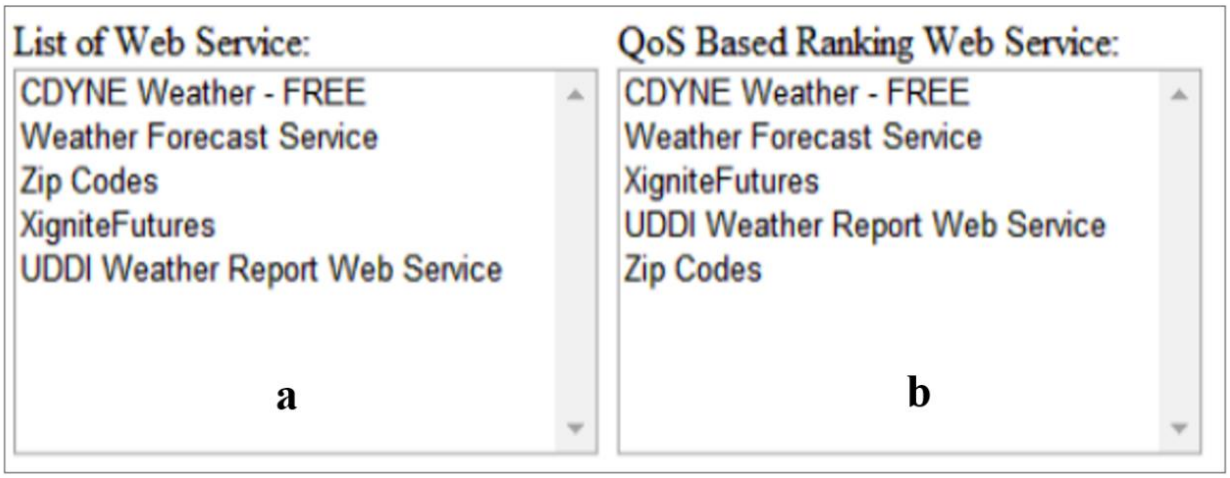


Figure 9. Comparative results between (**a**) normal searching of web service and (**b**) search using WSPoint.

In Table 3, the QoS quality attributes used are summarized. It also shows the maximum (max) and minimum (min) change based on the designated QoS used to convert the actual QoS value to percentage. The overall result shows that when the QoS weightage is provided, it differs from the search result. A lower WSPoint is accepted to be a higher quality web service. This is because the WSPoint increases as the response time increases, thus, the rating becomes of higher value. A rating with higher response time represents a weak web service. Meanwhile, a web service that takes lesser time to respond is rated as a good quality web service.

Table 3. Details on how QoS criteria is being used.

Types of QoS	Max Value	Min Value	Metric
Response Time (RT)	-	-	Milliseconds
User Rating (UR)	1	5	Rating
Availability (AV)	1	0	Rating

The result shows that the list of WS is the same but the arrangement of WS is different. Based on the proposed arrangement, the WSFeIn adapter selects the next web service if the first web service fails at any time. As such, it is recommended that a good quality web service list and matched functional requirement at the same time ensures the web service list meet user personalization. In order to test if the changes are significant, a user ac-acceptance survey is conducted on six companies from various domains. This is shown in Section 4.2.

4.2. Influence of QOS Weightage in Searching Result

In the above test, it is known that providing weightage to QoS to calculate WSPoint makes a difference and improves the search result. Next, a test is conducted to determine how QoS weightage influences web service search results. By looking into the same web service, the weightage for the QoS is adjusted to prove that results do change accordingly. Two examples are taken into consideration where one company requires high user-rated web services such as the entertainment industry and another company that prefers reliabil-

ity such as EWS. In this second test, the entire WS is the result when using the keyword "SMS". A total of 11 methods are selected from 2 different types of web services.

4.3. User Acceptance Testing

In order to ensure the test result meets the user's needs, acceptance testing is conducted. A set of six different companies are selected from different domains to execute this test. The respondents for the tests are companies with IT support who are familiar with web services. This ensures evaluations are performed by users who have proficiency in web servers and server administration, and are competent to answer the questions posted in the survey. Table 4 shows service points vs. percentage set for QoS criteria. Table 5 shows the result of the test performed based on which it can be concluded that with same sets of WS is able to provide the right web service based on the company policy. Rather than providing a list of web services to be used that are of good quality, another list of web services is provided which has been personalized to the company's needs.

Table 4. Service points vs. percentage set for QoS Criteria.

Test Case	Scenario	Top 2 Web Service ID Sorted by Service Point Ascending Order	
		Service ID	**Service Point**
1	RT:60% AV:30%UR:10%	38 53	570.95 576.84
2	RT:30% AV:60%UR:10%	38 53	286.17 288.92
3	RT:10% AV:60%UR:30%	53 38	96.97 97.12

Table 5. Test case set up for WSFeIn reliability test.

Test Objective	To check the icons are displayed based on status
Test Data	Two Web Service (A and B) with same functionality but in different servers and input data type Web Service A—East Asia Web Service B—South Central United States Web Service A—Input (Integer, Integer) Web Service B—Input (String, Integer)

The setup for this test is achieved by implementing the searching system (Figure 10) in Microsoft Azure Cloud Service allowing users to access it from anywhere. This is followed by a short briefing on the system. The users are allowed to search their preferred keyword and set the QoS weightage based on their requirements. As the result is produced, the IT developer or IT engineer with experience in the IT field around 5 to 7 years from the specific companies is required to verify the web service to be used.

Referring to Figure 11, different companies are viewed from various domains with all the companies having different priorities for quality criteria. Businesses that focus on traveling and e-learning are more interested in user rating while businesses that develop applications for security and accountancy are concerned with performance in terms of response time and availability of web service rather than user rating. Thus, different companies possess different criteria for their service availability and security in accordance with their business and user needs.



Figure 10. User acceptance test web service searching system.

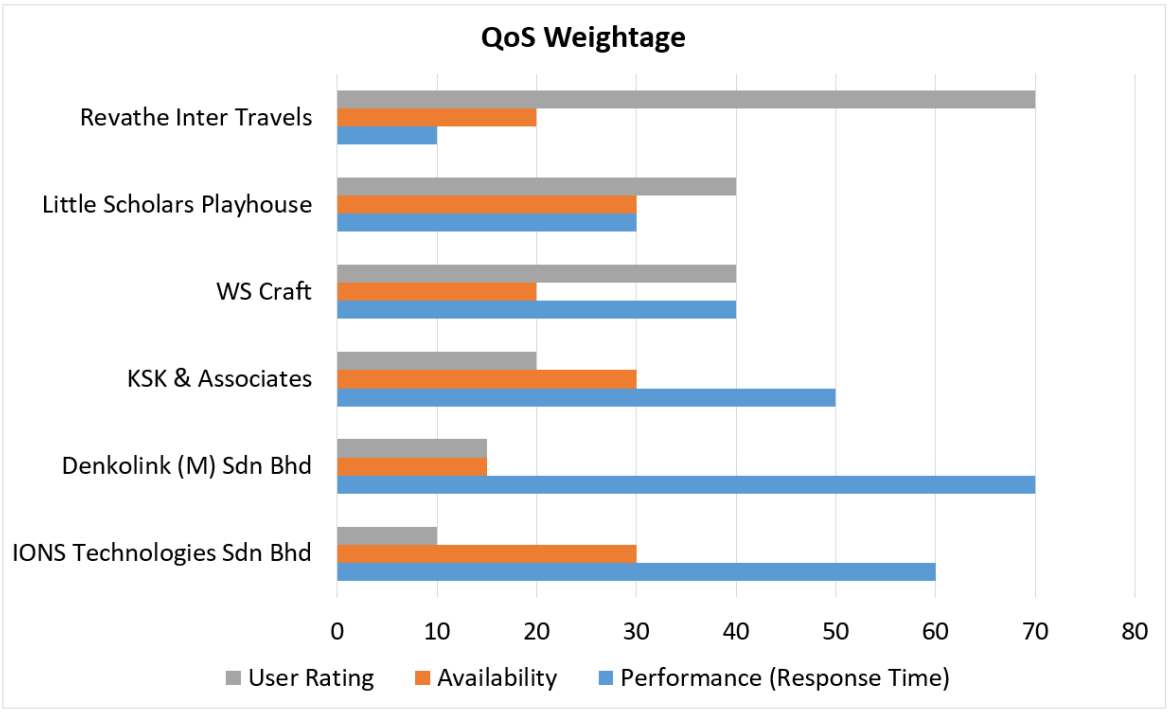


Figure 11. QoS weightage set by companies.

In sum, companies are indeed interested in using WS for their system development. As a way to support such an interest, it will help to provide them with a web selection tool with QoS criteria as part of their search process. This can ensure that the search incorporates QoS during searching. Use of this web selection tool can save time and effort, and it is easier to retrieve the web service compared to a web search which makes their searching process more convenient. Most of the respondents agree that web service composition saves more time in terms of development. Based on the scenarios written on the questionnaire, they can be built using a web service composition workflow. This method provides comfort to the development in different platforms as the process can be used in web service composition workflow.

4.4. WSFeIn

According to [16], the reliability of a web service composition workflow is calculated when the cream of the crop amongst web services is selected during design time which will offer the best performance time during composition. Web service composition is more reliable if we can store the functionally matched WS that allows switching in runtime. Thus, the reliability of the web service composition workflow is improved without slowing down the execution (increasing the execution time). Table 5 shows the test case set up for WSFeIn reliability test.

Based on the test result shown in Table 6, it can be concluded that the probability of failure for WSFeIn is 25%. The main reason for the failure of web service composition is when there is no more web service in the discovered list. The way to overcome this failure is to have a new web service added to the web service repository. Since the development is carried out using Windows Workflow, Visual Studio is designed to identify mistakes caused by typing before the program compilation and run [17]. In addition, the reliability of web service composition is dependent on technical implementation and helps to achieve and sustainable development [18]. Reliability of a web service composition is not measured by the quality of result returned by a service in the composition workflow, but rather by the stability of the service. This is measured by the uptime of the service and the ability to retrieve results despite the increase in requests. Other tests are also carried out that focus on the efficiency test for web service composition workflow. The value for WSFeIn is generated based on an average of five tests. Table 6 shows the execution time of WSFeIn.

Table 6. Test Cases For WSFeIn.

TC#	Step	Expected Result	Actual Output
1	Web Service A: Running Web Service B: Running	Workflow Provide Output	Workflow Provided Output
2	Web Service A: Disabled Web Service B: Running	Workflow Provide Output	Workflow Provided Output
3	Web Service A: Running Web Service B: Disabled	Workflow Provide Output	Workflow Provided Output
4	Web Service A: Disabled Web Service B: Disabled	Workflow Fail	Workflow Failed with error Message: Failed to invoke the service. Possible causes: The service is offline or inaccessible

5. Case Study: Early Warning System for Landslides and Floods in Malaysia

The case study is related to an early warning system for landslides and floods in Malaysia based on the agreement between the Government of Malaysia and the Government of Japan with JICA and a competent agency of the Government of Malaysia (Record of Discussion between the Government of Malaysia and Japan International Corporation Agency in Japanese Technical Cooperation, 2011). This project is a collaboration project between Universiti Sains Malaysia (USM), Multimedia University (MMU), and Universiti Tenaga Nasional (UNITEN). The actual output of this project is presented in a web-based system, but for this proposed work, it is implemented as a mobile application to demonstrate the effectiveness of the proposed method on a mobile device.

5.1. Requirements

Both the functional and non-functional requirements for this case study are extracted from the JICA discussion document. To begin with, the application requires loading of the Malaysia map when the application is installed. All the stations' statuses are to be displayed on the map itself within the single screen. The status of the station is displayed in different colors as shown in Table 7. The system is required to show the indicators and when the status change, it requires an action to switch the indicators accordingly.

Table 7. Station status indicator.

Color	Icon	Status	Action
Red		Danger	Send SMS to authorities and show signal on map
Yellow		Warning	Send SMS to authorities and show signal on map Show signal on map
Green		Normal	No action

For non-functional requirements, the system is required to be available at all times and reliable even when hazards occur. The identified QoS criteria need to be taken into consideration in the development of this prototype as shown in Table 8.

Table 8. QoS criteria.

QoS Attribute	Description
Availability	The system should be online 24/7 and able to obtain data and show a notification.
Reliability	The system should have a backup that will ensure when the failure occurs the system can recover with minimal interrupt
Performance	The total round-trip time required to send and obtain a request for each, and every task should be fast.

In this prototype development, real-time data from the USM server are not obtained. Instead, the actual data from the UPM server are copied and used in the EWS prototype. The design process of EWS implements all three phases of this study. A custom web service is used to update the map signal and retrieve data from the database server. There were two WSs developed with the same functionality but with different data types as input. This WS is later updated in our web service repository and goes through the WSPoint calculation as shown in Chapter 4. Both WS are placed on two different servers which are the South Central United States server and the Southeast Asia server. Since the EWS is hosted using Microsoft Azure Cloud Service, the advantages of the cloud are exploited by placing the servers in different regions. Therefore, we can obtain different WSPoint as the location of the server will influence the performance (response time) of a web service. This also increases reliability as in case of a failure in the server in a particular site, the alternative server can take over—thus reducing downtime of the service. For this prototype, the QoS criteria are being set, and its weightage is shown in Table 9. Based on this information, the WSPoint is calculated

Table 9. Used QoS Criteria and their weightage.

QoS Criteria	Weightage
Performance (Response Time)	60%
Availability	30%
User Rating	10%

The first step in web service composition is to design the Business Process Model Notation (BPMN) diagram. Figure 12 shows the BPMN diagram for the early warning system that is designed specifically for this project. Based on Figure 12 Windows Workflow 4.0 is used to design the web service composition flow. The data collected from the sensor

is omitted when building our web service composition flow and replaced with manual insertion of data into the database. This workflow is linked to the adapter (WSFeIn) algorithm that was proposed in Section 3. The WSFeIn is set up using the web-based administrator to ensure the output is following the workflow which is designed for the Early Warning System. Upon completion, the workflow is published in Microsoft Azure Cloud Service as a WSDL file. A Windows 10-based application is developed so that the WSDL file from the cloud service is consumed. The process shall be carried out in cloud service and only each station status is sent to the mobile application. Based on the result, the icons as shown in Table 7 shall be displayed.

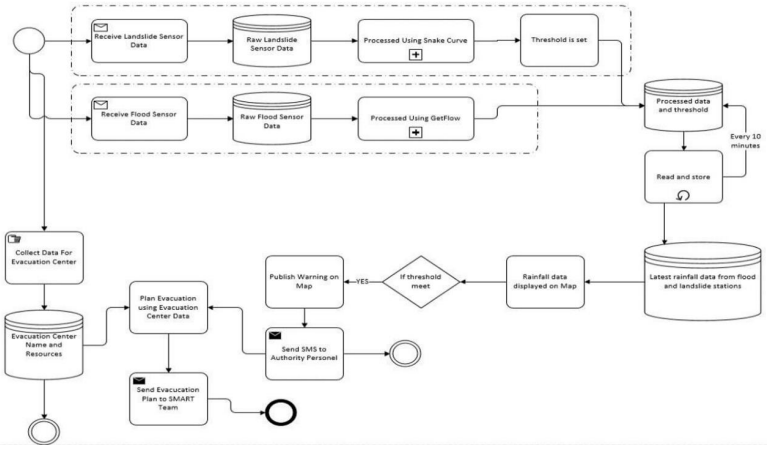


Figure 12. BPMN for early warning system for landslides and floods in Malaysia.

5.2. Non-Functional Requirement Testing for Case Study

Figure 13, shows the screenshot from windows phone emulator which demonstrate the system running in mobile application.

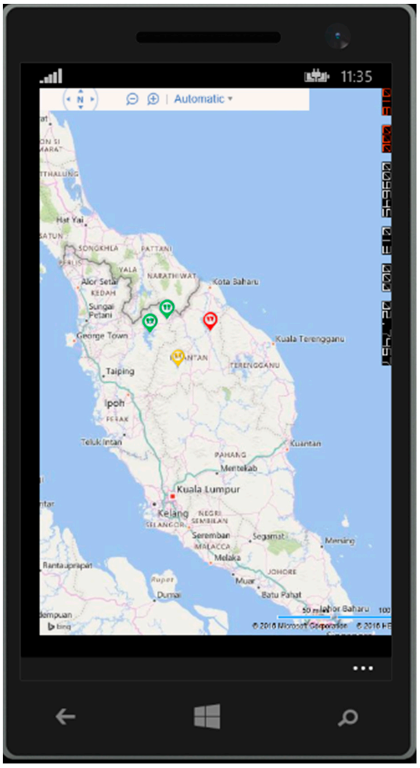


Figure 13. Test result for the functional requirement for mobile phone application.

6. Conclusions

In a nutshell, this research focuses on web service discovery, web service selection, and web service composition. A thorough identification of the proposed methods is analyzed and selected to be improvised and adapted based on requirements and appropriate criteria. The first phase is the web service discovery phase which retrieves and stores the WS from various repositories into a cloud-based web service repository. In this phase, WS from different repositories are collated and stored on the cloud-based repository database. The emphasis is on non-functional requirements which are quality of service. Each service is tested for availability and response time every three days to ensure the latest data are available on all the web services. The implementation also introduces the QoS weightage for the personalization of quality attributes for each organization.

In the web service selection and composition phases, the developer is required to search the web service based on the keywords. This is accomplished using REGEX to search for the method names from the cloud-based repository. The result is presented in WSPoint in ascending order for the composition unit to be used for web service composition workflow. This allows the developer to set up an adapter if there is any special need for data type in the web service composition workflow. Based on the configuration, the WSFeIn adapter can convert the value to the required data type for the workflow. If there is any failure midst composition, WSFeIn takes on a role to select the next web service as proposed by the selection unit and ensure the workflow success. Finally, the implemented method is applied to an early warning system which is developed using the phases described above and tested with a WSFeIn adapter to ensure adaptability in real-time. The scope for research is on an adaptive and dynamic web service composition which is able to adapt new web services to the web service composition workflow in real-time. This can be improved further by implementing semantic-based search that can improve the searching accuracy in the web service repository. One of the composition issues is solved that is related to the data type matching which improves the reliability. However, matching the schema is also important as a web service which comes with the same functionality and might have a different number of inputs. Some web services will fail if all the inputs are not provided with a value.

Author Contributions: Conceptualization, R.K.R.; Formal analysis, R.K.R.; Methodology, R.K.R.; Project administration, C.-K.H.; Supervision, F.-F.C. and S.-C.H.; Validation, R.K.R.; Writing—original draft, R.K.R.; Writing—review & editing, F.-F.C., S.-C.H. and C.-K.H. All authors have read and agreed to the published version of the manuscript.

Funding: The APC was funded by Multimedia University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Parejo Maestre, J.A.; Segura Rueda, S.; Fernández Montes, P.; Ruiz Cortés, A. QoS-aware Web Services Composition using GRASP with Path Relinking. *Expert Syst. Appl.* **2014**, *41*, 4211–4223. [CrossRef]
2. Zo, H.; Nazareth, D.L.; Jain, H.K. Service-oriented application composition with evolutionary heuristics and multiple criteria. *ACM Trans. Manag. Inf. Syst. TMIS* **2019**, *10*, 1–28. [CrossRef]
3. Gao, H.; Huang, W.; Duan, Y.; Yang, X.; Zou, Q. Research on cost-driven services composition in an uncertain environment. *J. Internet Technol.* **2019**, *20*, 755–769.
4. Xie, Y.; Guo, Y.; Mi, Z.; Yang, Y.; Obaidat, M.S. Loosely coupled cloud robotic framework for QoS-driven resource allocation-based Web service composition. *IEEE Syst. J.* **2019**, *14*, 1245–1256. [CrossRef]
5. Sheng, Q.Z.; Qiao, X.; Vasilakos, A.V.; Szabo, C.; Bourne, S.; Xu, X. Web services composition: A decade's overview. *Inf. Sci.* **2014**, *280*, 218–238. [CrossRef]
6. Feng, X.; Ren, Y.; Hu, J.; Wu, Q.; Jia, Y. A model for service composition with multiple QoS constraints. In Proceedings of the 2007 International Conference on Computing: Theory and Applications (ICCTA'07), Kolkata, India, 5–7 March 2007; pp. 208–213.

7. Ramalingam, R.; Muniyan, R.; Dumka, A.; Singh, D.P.; Mohamed, H.G.; Singh, R.; Noya, I.D. Routing Protocol for MANET Based on QoS-Aware Service Composition with Dynamic Secured Broker Selection. *Electronics* **2022**, *11*, 2637. [CrossRef]
8. Arunachalam, N.; Amuthan, A. A survey on QoS aware global optimization for dynamic Web service composition. In Proceedings of the 2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN), Pondicherry, India, 29–30 March 2019; pp. 1–6.
9. Fan, X.Q.; Fang, X.W.; Jiang, C.J. Research on Web service selection based on cooperative evolution. *Expert Syst. Appl.* **2011**, *38*, 9736–9743. [CrossRef]
10. Phalnikar, R.; Khutade, P.A. Survey of QoS based web service discovery. In Proceedings of the 2012 World Congress on Information and Communication Technologies, Trivandrum, India, 30 October–2 November 2012; pp. 657–661.
11. Mitkov, R. (Ed.) *The Oxford Handbook of Computational Linguistics*; Oxford University Press: Melbourne, Australia, 2022.
12. Chen, T.; Flores-Lamas, A.; Hague, M.; Han, Z.; Hu, D.; Kan, S.; Wu, Z. Solving string constraints with Regex-dependent functions through transducers with priorities and variables. In *Proceedings of the ACM on Programming Languages, 6(POPL)*; ACM: New York, NY, USA, 2022; pp. 1–31.
13. Brogi, A.; Canal, C.; Pimentel, E.; Vallecillo, A. Formalizing web service choreographies. *Electron. Notes Theor. Comput. Sci.* **2004**, *105*, 73–94. [CrossRef]
14. Ramasamy, R.K.; Fang, F.C.; Su, C.H.; Chin, K.H. Adaptive web service selection based on data type matching for dynamic web service composition. In Proceedings of the 5th International Conference on Computing and Informatics, ICOCI 2015, Istanbul, Turkey, 11–13 August 2015; pp. 488–494. Available online: https://repo.uum.edu.my/id/eprint/15603/1/PID105.pdf (accessed on 30 August 2022).
15. Cotfas, L.A.; Diosteanu, A. Software Reliability in Semantic Web Service Composition Applications. *Inform. Econ.* **2010**, *14*, 48–56.
16. Hanabaratti, K.D.; Rodd, S.F. Effective Replanning Web Service Composition Algorithm for Workload Scheduling under Cloud Environment. *arXiv* **2022**, arXiv:rs.3.rs-1421764/v1.
17. Ghazi, M.H. Diagnosis of Errors in Stalled Inter-Organizational Workflow Processes. Ph.D. Thesis, Nova Southeastern University, Fort Lauderdale, FL, USA, 2022.
18. Woltmann, S.; Kittel, J. Development and implementation of multi-agent systems for demand response aggregators in an industrial context. *Appl. Energy* **2022**, *314*, 118841. [CrossRef]