AN INTEGRATED THREE-FLOW APPROACH FOR FRONT-END SERVICE COMPOSITION

***Mei Ting Lim*[1] *and Moon Ting Su*[2*]**

[1,2]Department of Software Engineering
Faculty of Computer Science and Information Technology
University of Malaya, 50603 Kuala Lumpur, Malaysia

Email: meitinglim23@siswa.um.edu.my[1], smting@um.edu.my[2*](corresponding author)

DOI: https://doi.org/10.22452/mjcs.sp2019no3.1

ABSTRACT

End-User Service Composition (EUSC) aims to enable end-user programmers who are not professional developers develop applications by composing or aggregating existing web services. Despite the effort, studies have shown that, end-user programmers are not able to deal with the technical complexities involved in EUSC. One way to deal with this issue is Front-End Service Composition (FESC), which allows end-user programmers to compose web services at the presentation layer of an application by configuring user interface (UI) widgets that represent the back-end web services. However, there are not many studies on FESC and the existing ones suffer two main problems, namely, lack of control flow and/or application flow visualization, and they still require end-user programmers to have certain technical knowledge in the service composition process that these end-user programmers generally do not possess. Following that, this study proposes an integrated three-flow approach (application flow, control flow and data flow) to deal with the current limitations of FESC. The approach generates the GUI of web services automatically, thus allowing the UI of the application to be developed at the same time the required web services are assembled. The approach allows end-user programmers to explicitly configure the three different types of flows involved in service composition. A proof-of-concept prototype, QuickWSC, that incorporates the three-flow approach was developed. It adopts a side-by-side multiple-view design to support visual configuration of the three flows in an uncluttered yet synchronized manner that adhered to established design guidelines. A user evaluation study was conducted on QuickWSC. The results show that it is easy to compose web services by explicitly specifying the three flows, that the configurations of the three flows integrated in the two views helps in composing application from web services, and that no technical knowledge is required to use QuickWSC.

Keywords: End-User Service Composition (EUSC), Front-End Service Composition (FESC), Application User Interface, End-user programmers

1.0 INTRODUCTION

"End-user programmers" are people who write programs to assist themselves in accomplishing their primary tasks; they taught themselves to program and are not experienced in programming languages [1]. Studies have shown that there has been a significant increase of end-user programmers that develop their own software applications in comparison to professional software developers [2]. This has been attributed to the deployment of Web 2.0 technologies [1]. Another contributing factor is the impeding cost and time involved in engaging professional software developers to adapt software to meet changes of requirements [3]. As a result, it is important to empower end-user programmers to develop and adapt the software themselves at their levels of skills and contexts. That is the main goal of End-User Development (EUD) [3]. EUD is defined as "a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact" [3]. EUD integrates Human Computer Interaction, Software Engineering, Computer Supported Cooperative Work, and Artificial Intelligence to form user-centric designs that are suitable for end-user programmers.

With the proliferation of web services, EUD has expanded to include End-User Service Composition (EUSC), where end-user programmers develop applications by composing or aggregating existing web services [4]. In EUSC, the end-user programmer who composes the web services is also the person who is going to use the composed web services [5]. Nevertheless, studies have shown that end-user programmers cannot deal with the technical complexities involved in EUSC [6]. Front-End Service Composition (FESC) was introduced to enhance the intuitiveness of the web service composition process for end-user programmers [7]. FESC is characterized by composition of web services through the composition of their user interfaces (UIs) [7]. However, existing studies on FESC (as presented in Section 2.2) suffer two main problems, namely, lack of control flow and/or application flow visualization, and some still require end-user programmers to have certain technical knowledge in the service composition process that these end-user programmers generally do not possess. Even though end-user programmers welcome the opportunity to assemble or compose web services, they face a number of conceptual and usability issues of service composition [6] [8]. Common conceptual issues are: do not know what service composition is [6] [8]; confuse between design time and runtime (where during design time data is entered into the input fields of web services and expecting results to be shown) [6][8]; having difficulty in specifying the execution order of the web services and the logic of the application [6]; do not understand some technical terms [6]; worry about the security of sensitive information provided to the web services that require this information [6]. Common usability issues are having difficulty in positioning web services to create an organized visual layout and unsure of whether they have done the right things [6].

The aim of this study is to produce a new web service composition approach that leverages close integration between the development of the UI of an application and the composition of the web services required by the application, to enable end-user programmers to compose the service-based applications by using existing web services without much difficulty. Following that, this study proposes an integrated three-flow approach (application flow, control flow and data flow) that enables the UI of the respective application and the composition of the web services required by the application to be constructed concurrently. The approach allows end-user programmers to specify or define, visualize and modify the three diferent types of flows involved in web services composition through a graphical integrated view. This has the potential to ease the composition process and it does not require any technical knowledge.

The proposed approach was implemented in a proof-of-concept prototype, QuickWSC. QuickWSC was evaluated by end-user programmers in a user evaluation study and the results show that it is easy to compose web services by explicitly specifying the three flows, the configuration of the three flows in the integrated views helps in composing application from web services, and that no technical knowledge (such as communication protocol [9], component-based software [10] and event-based communication [10]) is required to use QuickWSC.

The remaining part of this paper is organized as follows. Section 2 reviews the web services composition approaches and the related work in EUSC and FESC. Section 3 describes the research methodology for this study. Section 4 details the proposed approach. Section 5 presents QuickWSC, its architecture design and UI, justification for a multiple-view design, and the navigational flow support design. Section 6 details the evaluation results. Section 7 discusses the results and comparison with existing work on FESC. Section 8 concludes the study.

2.0 LITERATURE REVIEW

Generally, web service composition approaches can be categorized into three main groups: static, dynamic, and semi-automated. This section briefly describes these categories and the web service composition approaches that fall under each of the category together with some exemplar studies. This section also describes FESC and the existing work in this area.

2.1 Web Service Composition Approaches

2.1.1 Static Web Service Composition

In static web service composition, the aggregation of services takes place at design time [11]. The users manually select the primitive services, design the composition logics, data and control flows [4], and the primitive services are bound to the process at design time [12]. Static composition is suitable in situations where business partners and service functionalities requirements remain fairly constant [11]. Static web service composition approaches for end-user programmers can be further categorized into workflow, spreadsheet-based, wizard-and-form-based [4], and reuse of pre-recorded composition logic.

Workflow approach allows users to define the sequence of connecting web services by using a graphical workflow diagram [4]. It has been applied in Baya [13], Flow Editor [14], Hypermash [15], Co-Taverna [16] and VIEW [17].

A recent systematic review of EUSC activities and tools reveals that workflow diagram editor is the most popular tool for end-user programmers [4]. Visual workflow composition UI makes the composition process easy and user-friendly [14]. Many of the implementations of workflow approach employ drag-and-drop feature [13]–[15],[17]. Some common ways of creating workflows in a visual editor are pipeline method [13] [15] and block [16]. Pipeline method connects web services and includes their inputs and outputs, while blocks are pre-developed blocks that support certain tasks or functions [1].

Spreadsheet-based service composition allows users to compose services in spreadsheet environment [4], where the users use spreadsheet formulas to achieve coordination among the services [18]. Exemplar studies are AMICO:CALC [18], Marmite [19], Mashroom [20] and Vegemite [21]. Some of the spreadsheet-based service compositions (such as Vegemite [21] and Mashroom [20]) introduce functions that enable the use of services in aggregating data.

Wizard-and form-based service composition approach solicits key information (such as the location of primitive services, order of invoking services, and so on) of the service composition from the users [4] by using forms and wizards. An example is Easy SOA [22].

Reuse of pre-recorded composition logic approach records the composition logics and configuration done by users and then reapplies them in other composite services that bear similarities to the existing one [4]. For example, in web scripting or web macros, repetitive common tasks in a web browser are recorded and replayed. One specific example is CoScripter [23].

2.1.2 Dynamic Web Service Composition

Dynamic web service composition automatically creates composite services based on the user's request and context [4]. To do that, the execution system needs to support automatic services discovery, selection and binding [11]. In dynamic composition, the determination and replacement of constituent services take place during runtime [11] or deployment time [12]. It is particularly useful when runtime changes of requirements are frequent and when services cannot be predicted at design time [11].

Some of the approaches employed in dynamic service composition are: use of high-level graphical language to define composite service, visualization, wizard-based and natural language processing [4]. Other approaches make use of semantic technologies and AI planning techniques, abstract model (such as in FUSION and OWLS-Xplan) and entity-relationship model (SWORD) [11].

2.1.3 Semi-automated Web Service Composition

Although dynamic service composition automates some of the tasks involved in assembling services, it limits the freedom of users in the process of service composition. A study on establishing requirements for EUSC tools shows that an appropriate amount of user's involvement in service composition is required [5]. A high degree of automation in service composition is generally not desired [24].

Several research tried to leverage both manual and automatic composition to assist users in the composition process [11]. An example of semi-automated service composition can be seen in a template-based service composition [25] that allows users to compose services by selecting pre-defined templates and modifying the tasks workflow in the selected templates according to their needs. The underlying system suggests suitable services for the tasks once the template modification is done. Another example is DoCoSoc [26] that uses a user-provided Service Oriented Architecture domain model to automate the service composition. This requires expertise in Model Driven Architecture and is not suitable for end-user programmers.

2.2 Front-End Service Composition

FESC enables service composition at the presentation layer, where applications are developed by composing web services using their UIs rather than the application logic or data [27]. The idea originated from graphical UIs integration, which refers to integrating components by combining their presentation front-ends rather than their application logic or data [28]. FESC allows the end-user programmers to play the role of a service composer and an application designer at the same time [27], and this is an advantage for the end-user programmer. A number of FESC tools are described next.

To integrate it into an application, ServFace Builder allows a user to drag a service operation from its Service Component Browser to its composition canvas [6] [27] [29]. It then automatically generates the corresponding service UI. ServFace Builder allows a user to design the data flow between connected services by connecting the UI element from where the data is obtained to the UI element that serves as the destination of the data. The data flow is represented by an arrow linking the two UI elements. Service operations can be dragged to the same page or different pages to create a multi-page application. The user can connect two pages to create a page transition, and this signifies the page flow or application flow. The order of execution of the service operations, or control flow, is implicit, following the application flow. ServFace Builder requires service developers to provide web services annotations to improve the visual appearance of the resulting applications [29].

MashArt [9] proposes to create composite web applications by integrating data, application, and UI components. It allows the modelling of the three types of components by using a unified model. It combines event-driven philosophy of UI and control-flow-based philosophy of service orchestration. Basically, the service components are linked to the UI components via connectors while the events are attached to the UI components. It supports various types of components such as RSS and Atom feeds for data component, SOAP and RESTful web services for service components and JavaScript UI components. Core functionality service component models which are reusable are provided to users. The limitation of MashArt is it targets advanced web users as it requires the users to have the understanding of communication protocol [30].

CRUISe [10] employs service-oriented paradigm for web-based UIs development. In CRUISe [10], UI components are provided as reusable services, namely, User Interface Services (UIS). They can be selected, configured and exchanged dynamically based on the model context. These reusable UI components integrate the UI logics at the presentation layer. The data or application logics are provided by back-end services. With a homogeneous access layer, the backend services can be bound to UI services. The UI components concept eases the development, maintenance and upgrading of UI. Integration of services is carried out on the client side to achieve a lightweight service orchestration at the presentation level. It also supports dynamic adaptations, for example, UIS reconfiguration and exchange. However, CRUISe is not suitable for end-user programmers as it requires the users to have the knowledge of component-based software and event-based communication.

One study proposed a Service Creation Environment (SCE) comprising a widget-based abstraction layer and a two-step service composition mechanism [31]. A widget is a reusable GUI that is linked to one or more functionalities of a service [31]. Every widget has a description file that contains abstract description and implementation description. The abstract description is used to explain the functionality of the widget. The implementation description refers to the index URL that provides access to the functionality. The GUI can be generated semantically based on the widget description. The first step of the mechanism includes GUI generation based on selected widgets and the creation of a composite service through automatic semantic matching and linking of all connectable widgets. The connectable links and GUI elements involved are shown to the user for further modification. In the second step, the user can manually personalize the services composed. The emphasis on semantic service composition restricts it to minor customization such as removing the unused generated links. Besides that, there is no proper organization of application flow and no visualization of the control flow of composed services.

Anoher study developed a Lightweight Service Creation Environment (LSCE) based on a data-driven service creation approach to compose services for mashup applications [32]. In this study, service is the basic data unit, and a Service Data Model (SDM) was created to support service description, data transformations, visualization, and extension of the services. The study developed an IFrame implementation for the SDM. The LSCE provides a drag-and-drop workspace for developing applications by drawing dataflow graphs, which are also known as Service Process Graphs (SPGs). SPGs would be parsed into JSON-based script before been sent for execution purpose. Being data-driven, LSCE does not provide a clear visualization of application flow and control flow.

Table 1 summarizes the existing work on FESC presented in this section. In summary, there are two main problems with the existing work. Firstly, lack of control flow and/or application flow visualization. Secondly, some still require the users to have certain technical knowledge (such as communication protocol [9], component-based software [10] and event-based communication [10]) that end-user programmers in general do not possess.

Table 1: Summary of existing work on FESC

Related Work/Tool	Approach	Features	Limitation
ServFace Builder [6] [27] [29]	Wizard-and form-based	*UI generation:* Based on service description and attached annotations *Type of flow:* Application/page flow, control flow and data flow *Multiple view:* Not applicable *Other:* UI elements represent web services	- Lack of control flow visualization
MashArt [9] [30]	Workflow	*UI generation:* Reusable UI components provided by component developers *Type of flow:* Control flow and data flow *Multiple view:* Not applicable *Other:* 3 types of components (data, application, and UI components)	- Requires knowledge on communication protocol - Lack of control flow and application flow visualization
CRUISe [10][30][33]	Wizard-and form-based	*UI generation:* Auto searching for suitable UI components captured as reusable services in database *Type of flow:* Control flow and data flow *Multiple View:* Visualization of information in multiple views *Other:* Reusable UI services	- Generic UI components which cannot be modified - Requires knowledge on component-based software and event-based communication - Lack of control flow and application flow visualization
SCE (Widget-based two-step service composition mechanism [31]	Wizard-and form-based	*UI generation:* Based on description file of the data model of GUI widgets *Type of flow:* Data flow *Multiple view:* Not applicable *Other:* Attach service to widgets (reusable GUI); provide suggestions for linking connectable widgets	- Generic GUI widgets which cannot be modified - Lack of control flow and application flow visualization
LSCE (Data-driven service creation approach) [32]	Wizard-and form-based	*UI generation:* Based on annotation template provided by service provider *Type of flow:* Control flow and data flow *Multiple view:* Not applicable *Other:* Visual elements in IFrame represent services	- Lack of control flow and application flow visualization

3.0 RESEARCH METHODOLOGY

The review of the literature shows two main problems of the current work on FESC and we developed a three-flow approach to address them. The three-flow approach was implemented in a proof-of-concept tool, QuickWSC. We adopted a multiple-view design in QuickWSC's UI to incorporate the three-flow approach. We also followed the design guidelines/rules for multiple-view presentation and interaction, and applied recommended techniques to conform to the design guidelines. To evaluate QuickWSC and the three-flow approach implemented in QuickWSC, we recruited end-user programmers to use QuickWSC to build an application by aggregating existing web services based on a predefined scenario. The evaluation was carried out in two phases, namely, a pilot study and a user evaluation study. The purpose of the pilot study was to gather participants' feedback to improve QuickWSC and the design of the user evaluation study.

This section presents the design of the pilot study and the user evaluation study, together with the research methods employed. The details of the multiple-view design of QuickWSC and the corresponding techniques employed are in Section 5.2.1.

3.1 Design of the Pilot Study

To evaluate QuickWSC and three-flow approach implemented in QuickWSC, we recruited end-user programmers to use QuickWSC to build an application by aggregating existing web services based on a predefined scenario.

The predefined scenario: "A man, while traveling between two cities, suffers from an electrical breakdown in his car. He turns on the QuickWSC client installed on his PDA/laptop (alternatively, the driver could make a call to an operator at the control center who will use the application on the caller's behalf). He then enters the registration number of his vehicle, problem of vehicle and driver information to Vehicle Insurance service to check the insurance status. Upon getting the confirmation from the Insurance service, he sends the location as well as the problem from which the car has been suffering via Mechanic service to find out the nearest mechanic to the car's location who is capable to fix the stated problem i.e. electrical breakdown. He can search for the nearest workshop's address via the Workshop service by providing the location, in case the mechanic could not handle the breakdown and might need to tow-away the car to a workshop."

Participant recruitment criteria: To ensure end-user programmers were recruited as the participants of the pilot study, the following criteria were used:

1. Do not have a well-trained programming knowledge but is able to use a computer competently.
2. Do not have web service composition knowledge.

Data collection instrument: The data collection instrument of the pilot study contains four sections. Table A.1 (Appendix A) shows the sections and their purposes, the questions asked to the participants in the questionnaire section and the purposes of asking these questions. The participants carried out the evaluation based on the sequence of the four sections in the data collection instrument.

Procedure of study and research methods: We contacted an acquantance from Faculty of Science to help to disseminate the information about the recruitment of participants for the study and to obtain the contact details of students who volunteered to take part in the study. The first author contacted the students and arranged for individual evaluation session to be conducted with each participant. During the session, the participant was given a set of the data collection instrument and informed about the things to be done. The participants were asked to engage in *think-aloud protocol* which required them to say out whatever that came into their mind as they were building the application by composing the relevant web services. This included what they were looking at, thinking, doing, and feeling. While the participants were performing the web services composition and execution, direct *observation* was carried out by the first author. The session was also video-recorded (with participants' permission) to enable further analysis where necessary. After completing the task of composing the application, the participants were asked to complete a *questionnaire* comprising of open-ended and close-ended questions. The data collection instruments were collected from the participants at the end of the evaluation session. The think-aloud protocol, observation and analysis of recorded video were used to find whether the participants faced any difficulty when using QuickWSC and whether they were able to compose the web services by using QuickWSC.

3.2 Design of the User Evaluation Study

The design of the user evaluation study is similar to the design of the pilot study with the following exceptions:

i) QuickWSC was refined by synchronizing the data flow between the two canvases;
ii) The printed user guide in Section 1 of the data collection instrument was changed to a screencast video.

The procedure of the user evaluation study is the same as the pilot study except that the participants were recruited through introduction by those who participated in the pilot study and by randomly approaching students at the Faculty of Science, University of Malaya.

4.0 THE PROPOSED APPROACH

We proposed and developed an integrated three-flow approach (application flow, control flow and data flow) that enables the UI of the respective application and the composition of the web services required by the application to be constructed concurrently. Fig. 1 illustrates the approach.

The approach extracts web services information from the URLs of Web Service Description Language (WSDL) files. The extracted web services information is saved into a local database and is used to generate the respective web services client stubs and web service client programs for the purpose of the execution of the web services. These two processes are executed by Java servlets. A web service client stub acts as a remote procedure call that provides the entry point between a web service client program and the web service server [34]. A web service client program is a remote client that contacts the web service and invokes the web service's methods [34].

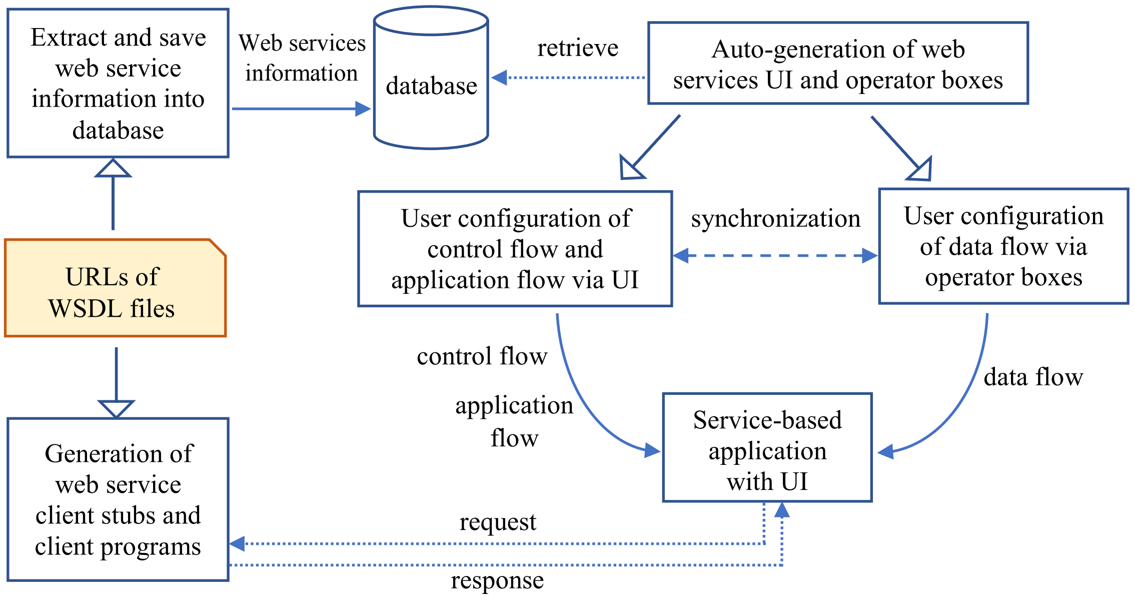
Extract and save web service information into database
Web services information
database
retrieve
Auto-generation of web services UI and operator boxes
URLs of WSDL files
User configuration of control flow and application flow via UI
synchronization
User configuration of data flow via operator boxes
control flow
application flow
data flow
Service-based application with UI
Generation of web service client stubs and client programs
request
response

Fig. 1: Integrated three-flow approach

The approach retrieves web service information from the database and auto-generates the UI elements and operator boxes of the web services end-users selected for the composition process. It allows the end-user programmers to explicitly configure the three flows: the data flow between connecting web services via the operator boxes as the data mapping between these web services; the control flow or the execution order of the web services included in the composition; and the application flow that determines the order of transition of UI pages in a multi-page application. Three types of control flow patterns are provided: sequence (sequential control flow), merge (convergence of two or more services into a single subsequent service) and split (divergence of a service into two or more parallel services each of which executes concurrently). The composed application will be saved as HTML.

Control flow and data flow are the two essential types of composition constructs in web services composition [12]. Our approach also includes application flow to specify the order of the web services' UI for the composed application.

Control flow could be implied by the application flow in simple applications, but in more complex applications, web services might need to be executed concurrently, requiring control flow to be separated from the application flow.

The three flows are used as the composition logic for the web service composition process and are integrated into the UI of the composed web services. The composition logic is embedded in the web services' UI for execution purpose. Input data can be provided through the UI of the composed web services and the execution of the web services can be initiated. This will result in the invocation of the respective web service client program, the web service client stub sending a request to the web service server and subsequently returning the response from the server to the client program for the output to be shown in the services' UI.

The approach uses web services' names that are meaningful to the end-user programmers instead of the services' technical names to help them in choosing the suitable web services for the composition. They would also be assisted in this aspect through the early visualization of the UI of the composed application during design time with the actual effects being reflected instantly on the scene. The UI elements serve as concrete mediums for end-user programmers to design the composition logic of the application by using the three flows made visible in a multiple-view design (Section 5.2.1). The multiple-view design requires the display of the information in the different views to be synchronized since they are presenting the same web services despite on different aspects. Besides that, the data flow configured in one of the view must be synchronized to the data mapping between the web services that appears in the UI of the application in order to transfer the data between the web services correctly during runtime.

In summary, the approach leverages instant integration, visualization and synchronization of the application UI development and web service composition, to simplify the process of web service composition. Through concrete visualization of the web service composition and the application UI and their synchronization, the complexity of composing and modifying assembled applications can be reduced to cater for end-user programmers who have no technical knowledge.

The approach addresses some of the common conceptual and usability issues faced by end-user programmers in web services composition [6] [8]. In particular, our approach

(i) enables end-user programmers to gain basic understanding of web services composition.
(ii) eliminates the need to differentiate between design time and run/execution time as it allows end-user programmers to key in input data into the UI fields of the web services and to execute the services to produce the output during the design time as well.
(iii) reduces the difficulty that might be faced by end-user programmers in specifying the execution order of the web services and logic of application by providing an intuitive graphical click-drag-release mechanism in selecting the required web services and specifying the control flow and application flow. It also allows end-user programmers to perform a simple two-click mechanism when specifying the data flow between two web services. This is done by first clicking on the output field of the first web service and then clicking on an input field of the subsequent web service.
(iv) presents minimal technical terms to the end-user programmers where no other technical term was used apart from "web services" and "composed services".
(v) produces an organized visual layout of the UI of the selected web services to reduce end-user programmers difficulty in positioning the layout of the web services.
(vi) provides instant visual update of the results of the end-user programmers' actions to reduce their uncertainty of whether they have done the right things.

Nevertheless, at this point of time, our approach does not address the end-user programmers concern of the security issue of the sensitive information provided as input data to certain web services.

5.0 QUICKWSC

We developed QuickWSC, a service creation environment, as a proof-of-concept prototype based on the proposed approach. QuickWSC was developed using Java for backend implementation, and JavaScript and CSS for interface implementation. The server used is Apache Tomcat Server and the system database is hosted on Microsoft SQL Server. This section explains the architecture design and UI of QuickWSC, and the multiple-view design adopted.

We identified the user requirements of QuickWSC based on our analysis of existing FESC tools and their limitations as shown in Table 1, and the common conceptual and usability issues faced by end-user programmers [6][8]. QuickWSC would provide the following features and strive for ease of use.

1) Automatic generation of UI of web services
2) Explicit visual configuration of three types of flow (application flow, control flow and data flow) involved in service composition
3) A multiple-view design
4) Execution of web services during design time and runtime
5) Minimal use of technical terms
6) Organized visual layout of UI of selected web services.
7) Instant visual update on actions taken and synchronization between views
8) Clear visualization of UI and the three flows

5.1 Architecture Design of QuickWSC

Fig. 2 shows the architecture design of QuickWSC. QuickWSC consists of two main parts, namely, Web Service Composition System and Execution System. The Web Service Composition System is responsible for the composition process and it comprises three subsystems: Web Service Retrieving Subsystem (WSRS), User Interface Generation Subsystem (UIGS), and Workflow Generation Subsystem (WGS). The Execution System runs the invocation process of the composed web services and it consists of Servlet Execution Subsystem (SES).

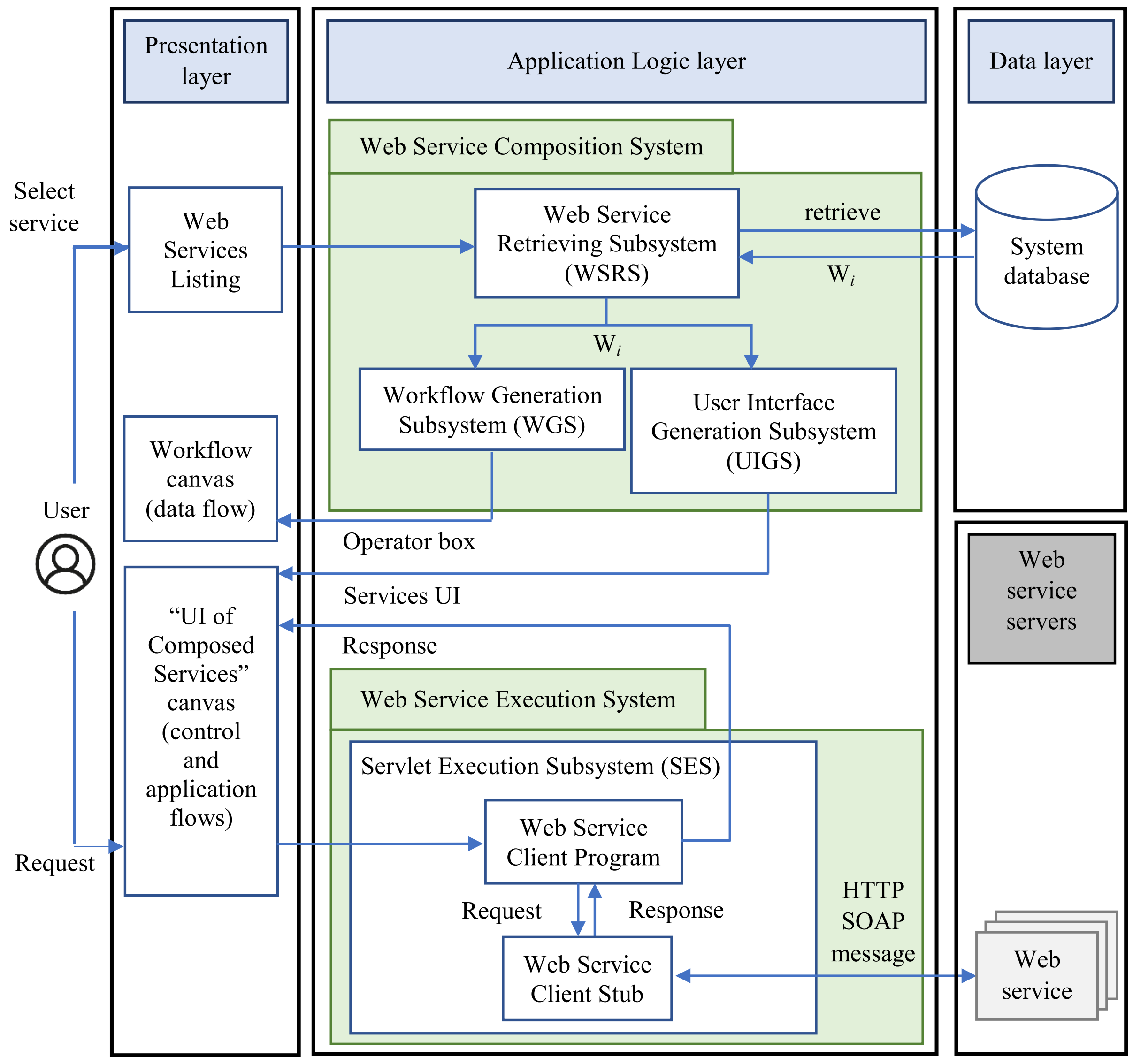
Presentation layer
Application Logic layer
Data layer
Web Service Composition System
Select service
Web Services Listing
Web Service Retrieving Subsystem (WSRS)
retrieve
W_i
System database
W_i
Workflow Generation Subsystem (WGS)
User Interface Generation Subsystem (UIGS)
Workflow canvas (data flow)
User
Operator box
Web service servers
"UI of Composed Services" canvas (control and application flows)
Services UI
Response
Web Service Execution System
Servlet Execution Subsystem (SES)
Web Service Client Program
Request
Request
Response
HTTP SOAP message
Web Service Client Stub
Web service

Fig. 2: Architecture design of QuickWSC

Web Service Retrieving Subsystem (WSRS) - WSRS retrieves the respective web service information (W_i) from the system database when a user selects the web service from the list of web services. The WSRS will return an object, *Wi, comprising of {wsid, wsop, wsdesc, wsinmsg, wsoutmsg, inelm, outelm}*, where,

wsid is the web service id in the system database;
wsop is the web service name;
wsdesc is the web service description;
wsinmsg is the web service input message;
wsoutmsg is the web service output message;
inelm is the web service input parameters and data type;
outelm is the web service output parameters and data type.

The web service information was extracted from the WSDL file of the web service. The *Wi* will be sent to UIGS and WGS.

User Interface Generation Subsystem (UIGS) - UIGS generates the UI for the respective web service based on the web service object from WSRS. A UI element will be generated for each input data required by the web service based on the type of the input data as shown in Table 3. For example, if the input data type is String or numeric, a TextBox will be generated. The results or output of executing the web service will be shown in a table form. All the input UI elements and the output table will be enclosed in a container box and shown in the "UI of Composed Services" canvas. The generated graphical UI serves as a concrete medium for the respective web service and provides a visualization of its input requirements and output. The whole composed services' UI is generated in HTML format and shown in the "UI of Composed Services" canvas.

Table 3: UI element type generation logic

Input Data Type	**UI Element Type**
String	TextBox
Numeric (int, float, decimal, double)	TextBox
Boolean	RadioButton
Range	Dropdown Box
Time	Time Picker
DateTime	DateTime Picker

Workflow Generation Subsystem (WGS) - WGS generates the operator box for the respective web service based on the web service object from WSRS. Each operator box represents a primitive web service and shows the names of its input and output parameters, and their data type in text format. Operator boxes were developed using jquery.flowchart (JavaScript Jquery plugin). The operator boxes are shown in the Workflow canvas.

Servlet Execution Subsystem (SES) - When the end-user invokes any operation of the web service, SES will execute the respective web service client program. The process communication is through XML HttpRequest. The program will send a request to web service client stub to invoke the corresponding method (Fig. 2). The runtime system of web service client stub sends a SOAP message to web service server. When the web service server receives the SOAP message, the runtime system of web service will execute the web service and send the response back to the web service client stub. Web service client stub extracts the SOAP message and sends the response to web service client program in the requested format. Client stub acts as a proxy between the client program and the web service. The system will create the web service client stub for every web service. Every method/operation in a web service will be created as an individual web service client program in the system.

5.2 User Interface of QuickWSC

Fig. 3 shows the UI of QuickWSC which consists of three frames. The right frame is the web services listing. It shows the available web services in QuickWSC. The center frame is the web service composition workplace ("User Interface of Composed Service" canvas) for users to compose web services. Users are allowed to arrange the application/page flow and control flow within this canvas. The order of the numbers (1, 2, 3, 4) shows the application flow (namely, the sequence of UI pages of the composed services). End-users can also arrange the web services into different vertical levels (for example as labelled by 'A', 'B', 'C' in Fig. 3) to configure the control flow of the composed services. The order of the alphabets (A, B, C) shows the control flow (namely, the sequence of execution of the constituent services). The left frame is the workflow workplace ("Workflow" canvas). It shows the operator boxes that represent the web services selected for the composition. An operator box shows the corresponding web service's input and output parameters and their data types. In this canvas, the users can configure the data flow between the web services by connecting the output parameter/field of a web service to an input parameter of the second web service. This makes the output data from the first web service to flow as an input to the second web service, resembling a data flowchart. An example of the data flow of the service compositon is shown by the blue connecting lines in Fig. 3. In short, the UI of QuickWSC presents the three different flows (application flow, control flow and data flow) in an integrated view.

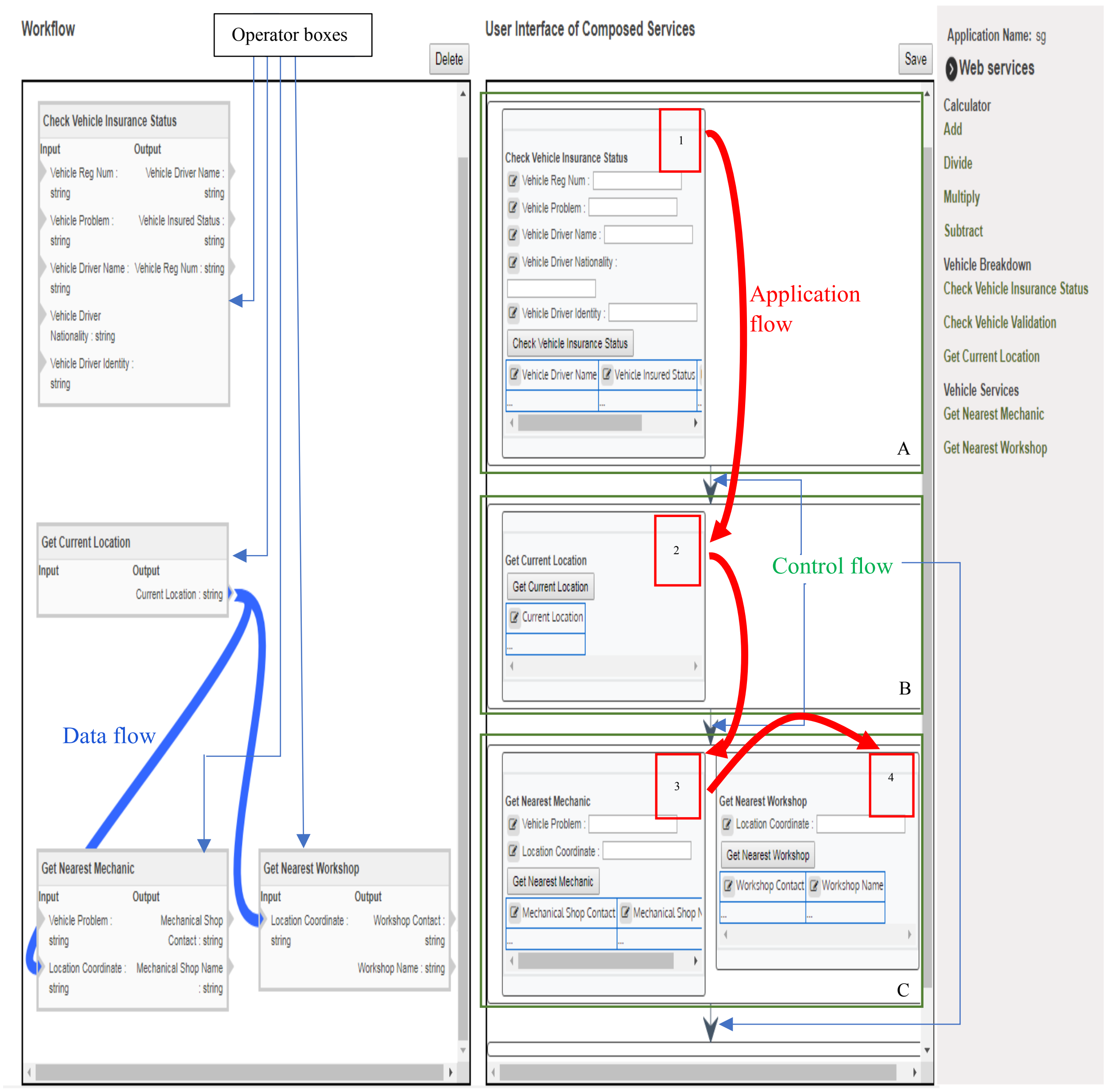

Fig. 3: User interface of QuickWSC

5.2.1 Multiple-View and Navigational Flow Support Design

Apart from a multiple-view design, cascading view or single view could be used in the design of a EUSC/FESC system. Nevertheless, QuickWSC adopted a multiple-view design in its UI in order to incorporate the proposed integrated three-flow approach and we provide our justification in this section. CRUISe system also adopted a multiple-view design [33].

A multiple-view design uses two or more different views to support the study of a single conceptual entity [35]. Our integrated three-flow approach calls for a diversity of views that are complimentary, thus fullfilling two guideline rules that advocate a multiple-view design [35]. A multiple-view is applicable when there is a diversity of attributes, models, user profiles, levels of abstraction or genres (rule of diversity) [35]. Our approach allows the users to explicitly configure the three different types of flows required in web service composition. Instead of cramping all the three flows in one single view, QuickWSC splits the configuration of data flow from the configuration of the control flow and application flow, by putting the first into a separate canvas (Workflow canvas) and the latter two into another canvas known as "UI of Composed Services" canvas. This is done to reduce unnecessary information overload to the users. Since control flow that chains the execution order of web services is more closely related to the application flow that depicts the order of pages transition, they are put in the same canvas. Data flow that depicts which web services' output serve as which web services' input, is of a different genre, and therefore captured in another canvas. The benefit of having the data flow in another canvas is particularly obvious when the same output of a web service serves as the input to more than one web services.

According to the rule of complementarity, multiple views is applicable when different views bring out correlations and/or disparities [35]. Having multiple views can help to show otherwise hidden relations. QuickWSC auto-generates corresponding operator box in the Workflow canvas for each web service selected for the composition. An operator box shows the basic information of the respective web service such as its name, input and output parameters together with their data types. When using the operator boxes to configure the data flow, the users will be able to identify compatible output data type to serve as input for another web service. When generating the operator box, QuickWSC also generates the UI elements of the respective web service in the "UI of Composed Services" canvas. The data flow connections configured by the users in the Workflow canvas show the expected transfer of data between the web services when the users execute the services, and this is not visible in the "UI of Composed Services" canvas, especially in the case of an output of a web service serving as input for multiple web services.

Having justified the use of a multiple-view design, the following explains the decisions made on QuickWSC's view presentation and interaction. QuickWSC follows four design guidelines/rules for these aspects [35]. First, QuickWSC goes for presenting the multiple views side-by-side instead of sequentially, to save users' time in looking at the two views (canvases) by showing them side-by-side (rule of space/time resource optimization). Understanding the relationships among views can be difficult for the users and perceptual cues can be used to make the relationships more obvious to the users (rule of self-evidence). Two perceptual cue techniques are applied in the design of QuickWSC to help the users in understanding the relationships between the Workflow and the "UI of Composed Services" canvases: brushing and navigational slaving. Brushing technique refers to user highlighting or selecting items in one view and the system highlighting the corresponding items in another view. Brushing technique is applied in both canvases. When a user moves the mouse pointer over the UI of a web service in "UI of Composed Services" canvas, QuickWSC will highlight it and the respective operator box in the Workflow canvas with a green boundary, and vice versa (Fig. 4). This helps the user to identify the corresponding entities between the two canvases.

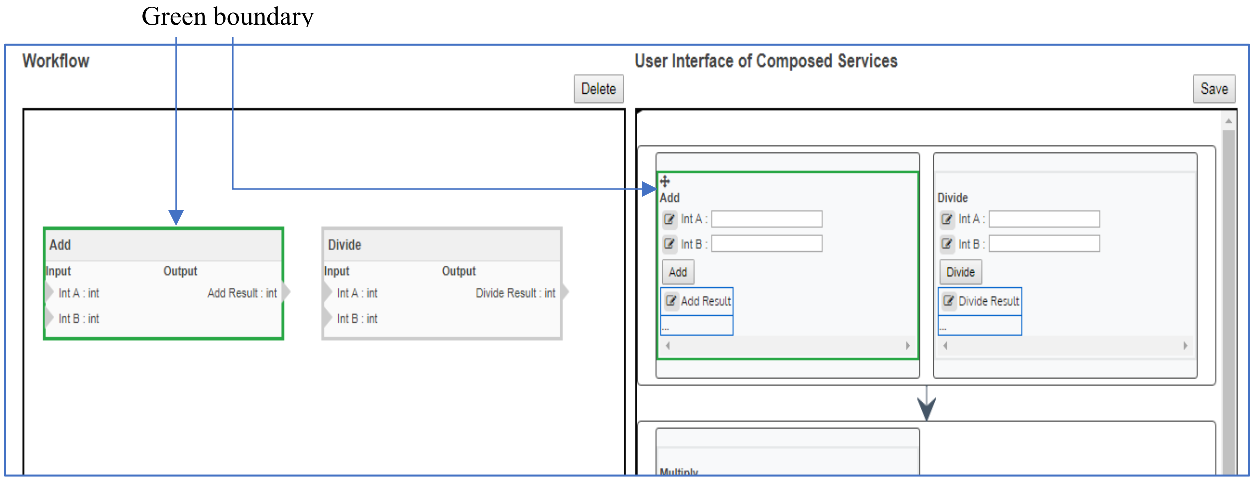

Fig. 4: Highlighting corresponding items in the two views

The navigational slaving technique refers to propogating movements in one view automatically to other views. The application of this technique in fact achieves the design required by the rule of consistency and will be explained in the following. Rule of consistency requires making the interfaces for multiple views consistent, and making the states of multiple views consistent. For example, if the objects or regions are shown/highlighted in one view, the corresponding objects or regions in the related view should also be shown. Consistent views facilitates learning and consistent states helps in object comparisons. QuickWSC applies the consistency rule to the Workflow and "UI of Composed Services" canvases. The operator boxes in the Workflow canvas and the respective UI elements of the corresponding web services in the second canvas are positioned at the same horizontal coordinate points (y-axis) in the two canvases. The navigational slaving technique is applied to the horizontal and vertical scrollbars of both canvases, resulting in the display of the corresponding regions in both the two canvases when the users scroll either one of the canvases using its scrollbars. Changes in "UI of Composed Services" canvas are reflected in Workflow canvas.

Rule of attention management is about using the perceptual techniques to focus the users' attention on the right view at the right time. It is a challenge to ensure the users' attention is at the right place at the right time when there are multiple views so that they are not distracted away from the view. QuickWSC uses color highlighting technique for attention management, where it highlights the items of focus with a green boundary in both views (Fig. 4).

6.0 EVALUATION

This section presents the results of the evaluation conducted to evaluate QuickWSC and the three-flow approach. This includes the results of the pilot study and the user evaluation study.

6.1 Results of Pilot Study

Two students from the Faculty of Science, University of Malaya, participated in the pilot study. The results are summarized below: Both of the participants were able to compose the services based on the given scenario. One took 13 minutes to compose the services. The other took 10 minutes. From the observation, it was noticed that the participants knew what they were supposed to do but they were not sure on how to do it. For example, the participants knew that they had to drag the service from the web services listing frame and drop it in the "UI of Composed Services" canvas, but they verbally confirmed with the researcher regarding the dropping position. Besides that, they knew that they had to specify the data flow, but they did not know how to create the needed connections in the Workflow canvas. One of the participants suggested to synchronize the data flow between the two canvases.

6.2 Results of User Evaluation Study

This section presents the results of the user evaluation study: the participants' background, the results of observation, success rate of participants in composing the application using QuickWSC, and the participants' opinion on the features of QuickWSC.

6.2.1 Participants' Backgrounds

Twenty students comprising of two master and eighteen undergraduate from the Faculty of Science, University of Malaya, took part in the user evaluation study (Question 1). The average participation time was 20 minutes. The participation took place in the Human Computer Interaction Lab, Faculty of Computer Science and Information Technology, University of Malaya.

The participants' levels of programming experience (Question 2) and computer skills (Question 3) are as below: The majority of the participants (fifteen) have not learned programming before. Twelve of these participants have basic computer skills such as Internet, email, hardware, software concepts, word processing, formatting, presentations, graphics, multimedia, and spreadsheets. The remaining three have intermediate computer skills including Internet, email, hardware, software concepts, terminology, word processing, formatting, tables, presentations, graphics, multimedia, spreadsheets, and databases. One participant self-learnt Python, C++, R and Arduino, but the participant told the researcher that he could not write a complete program with the languages. Another four participants also chose the option indicating they learned programming on their own and could write simple programs. However, when asked by the researcher, they mentioned that they attended a one-day programming class during their matriculation study, where codes were given to them by the tutor, and they were able to execute the codes but did not learn about programming theory. Three of these four participants have intermediate computer skills and one have basic computer skills.

6.2.2 Observation Results

Choosing web services from the web services listing frame: After reading the given scenario, the majority of the participants (seventeen) were able to choose the web services from the web services listing frame without any difficulty. One participant was not sure whether the chosen services were the ones required by the scenario. Another two felt it was something new to them as they had never explored web services and needed some time to find them. Despite that, these three participants were able to choose the correct web services from the web services listing frame.

Success rate in composition: The majority of the participants (fifteen) successfully composed the services based on the given scenario, by configuring the three flows. Two of these fifteen participants said that they were not familiar with the three-flow concept when using QuickWSC for the first time, but were able to figure out how it works and were able to configure the three flows easily after the first experience. Only one of the fifteen participants was confused with the concept of application flow and control flow even though he managed to compose the services successfully. All of these fifteen participants also agreed that the three flows could be clearly visualized in the integrated views when composing the services. Five participants failed in composing the web services. They did not manage to configure the application flow and control flow, but were able to configure the data flow.

6.2.3 Evaluation Results for Features of QuickWSC

For Questions 1 – 12 in Part B of the questionnaire, participants were asked to rate the features of QuickWSC using a 5-point Likert scale (1 - "Strongly Disagree", 2 – "Disagree", 3 – "Neither Agree nor Disagree", 4 – "Agree" and 5 - "Strongly Agree"). Participants were asked to circle their chosen option for each of these Likert scale questions and to state the reason if he or she chose the option of 3 or below. Fig. 5 shows the distribution of the participants' opinion on the ease of use (Questions 1 – 5) and synchronization (Question 6) features of QuickWSC.

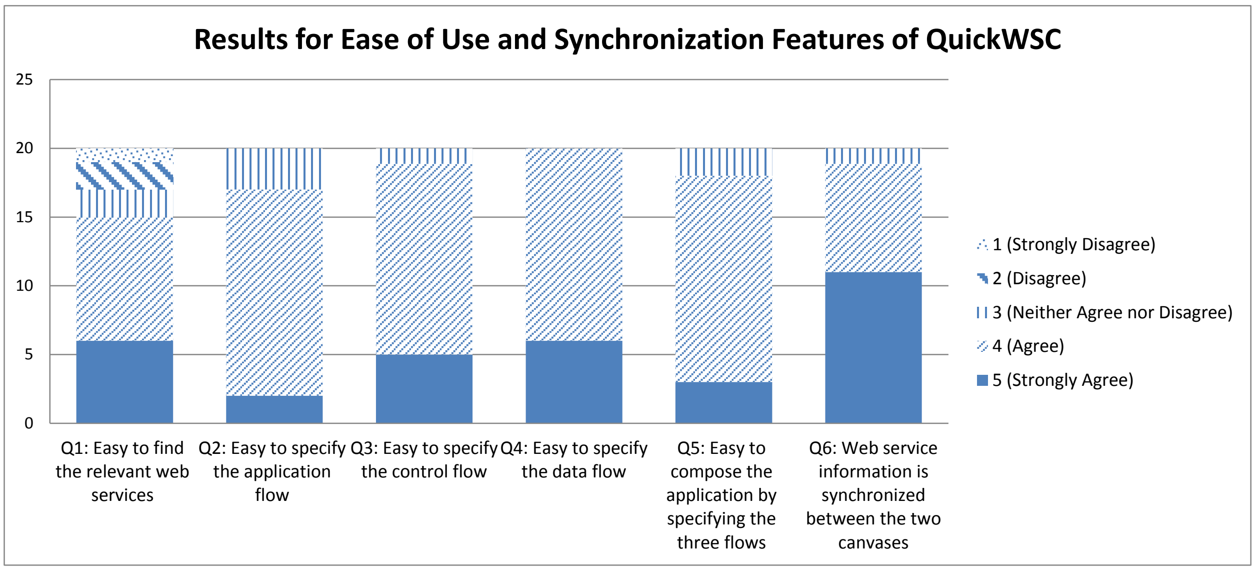

Fig. 5 : Results for ease of use and synchronization features of QuickWSC

Question 1 states “It is easy to find the relevant web services for the scenario. If your response is 3 and below, please state the obstacles of finding the relevant web services.”. Fifteen (75%) participants agreed or strongly agreed that it was easy to find the relevant web services in QuickWSC. Two participants were undecided on the ease of finding the relevant web services giving the reasons that they were new to web services and not sure whether they were choosing the correct web services. Three participants disagreed or strongly disagreed that it was easy to find the relevant services. The reasons they gave were they did not know what is web services and one of them have never explored web services.

Question 2 states “It is easy to specify the application flow in the “UI of composed services” canvas. If your response is 3 and below, please state why it is not easy to specify the application flow in the respective canvas.” Seventeen (85%) participants agreed or strongly agreed that it was easy to specify the application flow. Three (15%) participants were undecided on this aspect. One of them gave the reason of unfamiliarity with the tool when using it for the first time. Another was confused with arranging the application flow in the canvas. The third was confused between the application flow and control flow.

Question 3 states “It is easy to specify the control flow in the “UI of composed services” canvas. If your response is 3 and below, please state why it is not easy to specify the control flow in the respective canvas.”. All except one of the participants agreed or strongly agreed that it was easy to specify the control flow. The one who was undecided on the ease of specifying the control flow was confused between the application flow and control flow.

Question 4 states “It is easy to specify the data flow in the “workflow” canvas. If your response is 3 and below, please state why it is not easy to specify the data flow in the respective canvas.”. All participants agreed or strongly agreed that it was easy to specify the data flow.

Question 5 states “It is easy to compose the application by specifying the three flows (application flow, control flow and data flow) in the integrated two views (workflow canvas and UI of composed services canvas). If your response is 3 and below, please state why it is not easy to compose the application by specifying the three flows in the integrated two views.”. Eighteen (90%) participants agreed or strongly agreed that it was easy to compose the application by specifying the three flows (application flow, control flow and data flow) in the integrated two views of Workflow canvas and “UI of Composed Services” canvas. The two undecided participants stated that this was new to them and they were not familiar on their first use of this tool, but would be easy to use once they were familiar with it.

Question 6 states “The web service information is synchronized between the “workflow” canvas and “UI of composed services” canvas. If your response is 3 and below, please state what web service information is not synchronized between the two canvases.”. Nineteen (95%) participants agreed and strongly agreed that the web service information was synchronized between the Workflow canvas and “UI of Composed Services” canvas. Only one participant was undecided on this aspect because she was wondering if reversing the data flow was applicable in the system and has any effect during the design time.

Fig. 6 shows the distribution of the participants’ opinion on the visualization (Questions 7 - 11) and execution (Question 12) features of QuickWSC. Question 7 states ““UI of composed services” canvas provides a clear visualization of the GUI of the web services on an application page. If your response is 3 and below, please state why the “UI of composed services” canvas does not provide a clear visualization of the GUI of an application page.”. All the participants agreed or strongly agreed that the “UI of Composed Services” canvas provides a clear visualization of the GUI of the web services on an application page.

Question 8 states ““UI of composed services” canvas provides a clear visualization of the application flow of the composed services. If your response is 3 and below, please state why the “UI of composed services” canvas does not provide a clear visualization of the application flow of the composed services.”. Question 9 states ““UI of composed services” canvas provides a clear visualization of the control flow of the composed services. If your response is 3 and below, please state why the “UI of composed services” canvas does not provide a clear visualization of the control flow of the composed services.”. All the participants agreed or strongly agreed that the “UI of Composed Services” canvas provides a clear visualization of the application and control flows of the composed services.

Question 10 states “Workflow” canvas provides a clear visualization of the data flow of the composed services. If your response is 3 and below, please state why the “Workflow” canvas does not provide a clear visualization of the data flow of the composed services.”. Question 11 states “The three flows (application flow, control flow and data flow) are displayed clearly in the integrated two views (workflow canvas and UI of composed services canvas) during design time. If your response is 3 and below, please state which part(s) is/are not displayed clearly in the integrated two views during design time.”. All the participants agreed or strongly agreed that Workflow canvas provides a clear visualization of the data flow of the composed services, and the three flows were displayed clearly in the integrated two views during design time.

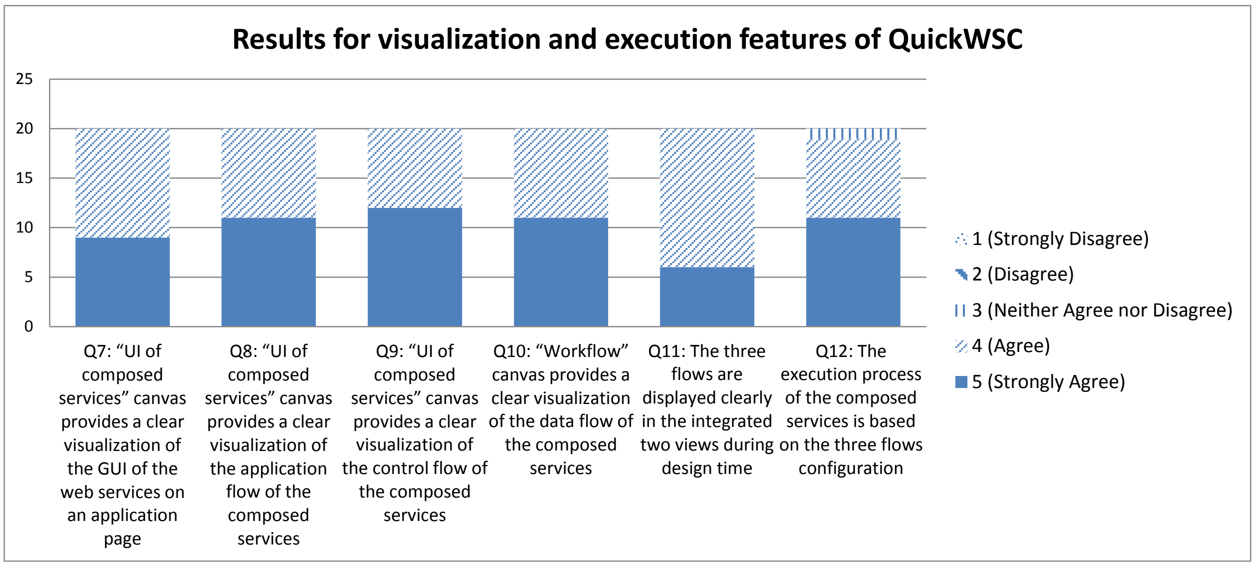

Fig 6: Results for visualization and execution features of QuickWSC

Question 12 states “The execution process of the composed services is based on the three flows configuration (application flow, control flow and data flow). If your response is 3 and below, please state why you say that the execution process of the composed services is not based on the three flows configuration.”. Nineteen participants (95%) agreed or strongly agreed that the execution process of the composed services was based on the three flows configuration and they managed to get the execution result. The only undecided participant stated that he was not sure whether he was composing in the right way due to confusion between application flow and control flow.

Fig. 7 shows the results for Questions 13 and 14. Question 13 states “Does having the three flows configurations integrated in the two views help you in composing the application? Please state your reason.”. All of the participants indicated a “Yes”. The reasons they gave were the system showed a clear visualization of the three flows and the three flows assisted them in understanding how the composed application processes and works.

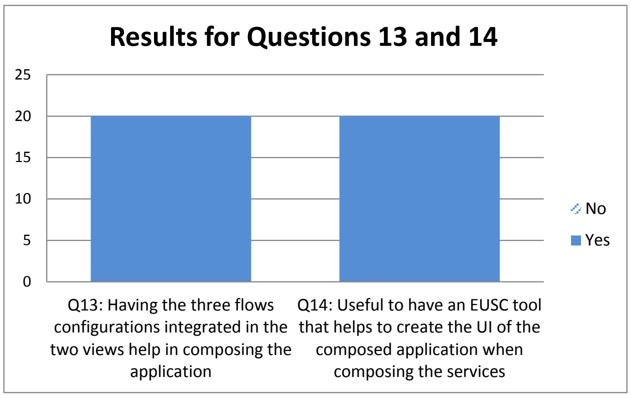

Fig 7: Results for Questions 13 and 14

Question 14 states "Is it useful to have an end-user service composition tool that helps you to create the user interface of the composed application when you are composing the services? Please state your reason.". All participated stated "Yes". The reasons they gave were the tool is easy to operate and save time in composing the web services; that they could use the tool to solve some simple problems and during emergency. However, some participants stated that it is slightly difficult for the first time users as they are not familiar with the tool.

7.0 DISCUSSION OF RESULTS

Table 4 summarises the comparison of our work with the related works on FESC. In terms of composition model, CRUISe requires composition knowledge to define the composition logic and description in the context module to find a suitable UI for the web service. The components are configured with their description files and they are connected with each other through the definition of the events and operations in the user requirement context; MashArt requires the understanding of communication protocol between components and event attachment to the UI components. Users need to attach the events and operations to UI components and connect to service components to compose the mashup service; ServFace Builder employs the hybrid of control flow and data flow in its composition model; SCE and LSCE use the data flow as the composition model. Our work employed the hybrid of application, control and data flows as the composition model.

Table 4: Comparison with related work

Related work	**Composition model**	**Type of flow**			**Technical knowledge required**
		Application flow	**Control flow**	**Data flow**	
ServFace Builder [6], [27], [29]	Hybrid of control and data flows	Explicit	Implicit	Explicit	None
MashArt [9], [30]	Event-based	-	Implicit	Explicit	Communication protocol
CRUISe [10], [30], [33]	Abstract model	-	Implicit	Implicit	Component-based software and event-based communication
SCE [31]	Data flow	-	-	Explicit	None
LSCE [32]	Data flow	-	Implicit	Explicit	None
Our Work	Hybrid of application flow, control and data flows	Explicit	Explicit	Explicit	None

All the work in Table 4 includes data flow. This could be due to the semantics of data flow is easy to understand [30]. All work except SCE includes control flow and only ServFace Builder and our work include application flow. We regard a provided flow as explicit if it is visually visible to the end-user programmers, and as implicit if the flow is implied and not directly visible. MashArt allows end-user programmers to connect the data flow between components explicitly in its editor. In MashArt, control flow is implicitly configured as events attached to the components. CRUISe allows control flow and data flow to be configured in the user requirement context and this is implicit. SCE allows end-user programmers to compose services by configuring data flow through creation of explicit links between GUI widgets. In LSCE, data flow configuration is done through creation of explicit links between services, the control flow is implied by the rules in the services. In terms of technical knowledge required, studies have reported that CRUISe [10], [30], [33] and MashArt [9], [30] are not for end-users programmers [30].

ServFace Builder is the closest to our work. It also provides the three flows features but the control flow is implicitly implied by the application or page flow. Control flow could be implied by the application flow in simple applications, but in more complex applications, web services might need to be executed concurrently, and this requires control flow to be separated from the application flow. ServFace Builder allows the definition of two of the five basic control flow patterns (sequence, merge, split, condition and loop) [32]. They are sequential control flow (sequence) and alternative control flow (condition). Our work supports the definition of sequence, merge and split. In ServFace Builder, the standard view enforces a sequential application/page flow. To alter it, the end-user programmers need to switch to the "page flow" view. The three-flow approach in our QuickWSC shows the visualization of application, control and data flows explicitly by using a multiple-view design and the end-user programmers do not have to switch to a different view.

The results of the user evaluation study shows promising potential of QuickWSC and the underlying approach. It shows that it is easy to compose web services by explicitly specifying the three flows in an integrated two views (Part B Question 5) and the configurations of the three flows integrated in the two views helps in composing application from web services (Part B Question 13). The high service composition success rate of the participants with no technical knowledge recruited in the user evaluation study shows that no technical knowledge is required to use QuickWSC. Besides, the positive results of the ease of finding the relevant web services and of the ease of specifying each of the three flows (Part B Questions 1 – 4) show that QuickWSC is of high usability or ease of use. Results also show that QuickWSC provides clear visualization of the GUI of the web services and the three flows (Part B Questions 7 – 11) and the clear visualization of three flows and the three flows assist the participants in understanding how the composed application processes and works (Part B Question 13). Results are also positive in terms of synchronization of the web service information between the two views/canvases (Part B Question 6) and execution of composed services following the three flows configuration (Part B Question 12).

8.0 CONCLUSION AND FUTURE WORK

We developed an integrated three-flow approach to deal with the current limitations of FESC. The approach generates the GUI of web services automatically, thus allowing the UI of the application to be developed at the same time the required web services are assembled. The approach allows end-user programmers to explicitly configure the three different types of flows (application flow, control flow and data flow) involved in service composition. A proof-of-concept prototype, QuickWSC, that incorporates the three-flow approach was developed. It adopts a side-by-side multiple-view design to support visual configuration of the three flows in an uncluttered yet synchronized manner that adhered to established design guidelines. A user evaluation study was conducted on QuickWSC. The results show that it is easy to compose web services by explicitly specifying the three flows, that the three flows configurations integrated in the two views helps in composing application from web services, and that no technical knowledge is required to use QuickWSC. In comparison to existing work in FESC, ours is the only approach that provides explict configuration of application, control and data flows, and without requiring technical knowledge. Possible future work would be to automate the discovery of urls of WSDL files to simplify the addition of more web services into QuickWSC and to provide semantic searching for relevant web services.

ACKNOWLEDGEMENT

The authors thank University of Malaya's BKP Special Grant (BKS002-2018) for supporting this work.

REFERENCES

[1] R. Latih, A. Patel, and A. M. Zin, “A Systematic Literature Review of end-user programming for the web mashup”, *Journal of Theoretical and Applied Information Technology*, Vol. 60 No. 1, 2014, pp. 119–132.

[2] M. M. Burnett and B. A. Myers, “Future of End-user Software Engineering: Beyond the Silos” in *Proceedings of the Future of Software Engineering*, ACM, New York, NY, USA, 2014, pp. 201–211.

[3] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, “End-User Development: An Emerging Paradigm”, In: Lieberman H., Paternò F., Wulf V. (eds), *End User Development. Human-Computer Interaction Series*, Springer, Dordrecht, Vol. 9, 2006, pp. 1–8.

[4] F. Hang and L. Zhao, “Supporting end-user service composition: A systematic review of current activities and tools” in *Proceedings of the 22nd IEEE International Conference on Web Services (ICWS),* 2015, pp. 479–486.

[5] A. Ridge and E. O’neill, “Establishing Requirements for End-user Service Composition Tools”, *Requir. Eng.*, Vol. 20 No. 4, Nov. 2015, pp. 435–463.

[6] A. Namoun, T. Nestler, and A. D. Angeli, “Conceptual and Usability Issues in the Composable Web of Software Services”, In: Daniel F., Facca F.M. (eds), *Current Trends in Web Engineering. ICWE 2010. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 6385, 2010, pp.396-407.

[7] N. Laga, E. Bertin, and N. Crespi, “Composition at the frontend: The user centric approach” in *Proceedings of 2010 14th International Conference on Intelligence in Next Generation Networks*, 2010, pp. 1–6.

[8] C. Cappiello, M. Matera, and M. Picozzi, “A UI-Centric Approach for the End-User Development of Multidevice Mashups”, *ACM Trans. Web*, Vol. 9 No. 3, Jun. 2015, pp. 11:1–11:40.

[9] F. Daniel, F. Casati, B. Benatallah, and M.-C. Shan, “Hosted Universal Composition: Models, Languages and Infrastructure in mashArt”, In: Laender A.H.F., Castano S., Dayal U., Casati F., de Oliveira J.P.M. (eds), *Conceptual Modeling - ER 2009. ER 2009. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 5829, 2009, pp. 428–443.

[10] S. Pietschmann, M. Voigt, A. Rümpel, and K. Meißner, “Cruise: Composition of rich user interface services”, In: Gaedke M., Grossniklaus M., Díaz O. (eds) Web Engineering. ICWE 2009. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, Vol. 5648, 2009, pp. 473–476.

[11] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, “Web services composition: A decade’s overview”, *Information Sciences*, Vol. 280, Oct. 2014, pp. 218–238.

[12] A. L. Lemos, F. Daniel, and B. Benatallah, “Web Service Composition: A Survey of Techniques and Tools”, *ACM Comput. Surv.*, Vol. 48 No. 3, Dec. 2015, pp. 33:1–33:41.

[13] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, “Baya: Assisted Mashup Development As a Service” in *Proceedings of the 21st International Conference on World Wide Web*, ACM, New York, NY, USA, 2012, pp. 409–412.

[14] B. Pi, G. Zou, C. Zhong, J. Zhang, H. Yu, and A. Matsuo, “Flow editor: Semantic web service composition tool” in *2012 IEEE Ninth International Conference on Services Computing (SCC),* Honolulu, HI, 2012, pp. 666–667.

[15] F. Hang and L. Zhao, “HyperMash: A Heterogeneous Service Composition Approach for Better Support of the End Users” in *2013 IEEE 20th International Conference on Web Services*, Santa Clara, CA, 2013, pp. 435–442.

[16] J. Zhang, “Co-Taverna: A tool supporting collaborative scientific workflows” in *2010 IEEE International Conference on Services Computing (SCC),* Miami, FL, 2010, pp. 41–48.

[17] C. Lin *et al.*, “Service-oriented architecture for VIEW: a visual scientific workflow management system” in *2008 IEEE International Conference on Services Computing (SCC’08),* Honolulu, HI, 2008, pp. 335–342.

[18] Ž. Obrenović and D. Gašević, “End-User Service Computing: Spreadsheets as a Service Composition Tool”, *IEEE Transactions on Services Computing*, Vol. 1 No. 4, Oct. 2008, pp. 229–242.

[19] J. Wong and J. Hong, “Making mashups with marmite: towards end-user programming for the Web,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2007, pp. 1435–1444.

[20] G. Wang, S. Yang, and Y. Han, “Mashroom: End-user Mashup Programming Using Nested Tables” in *Proceedings of the 18th International Conference on World Wide Web*, 2009, pp. 861–870.

[21] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau, “End-user Programming of Mashups with Vegemite” in *Proceedings of the 14th International Conference on Intelligent User Interfaces*, ACM, New York, NY, USA, 2009, pp. 97–106.

[22] T. Yamaizumi, T. Sakairi, M. Wakao, H. Shinomi, and S. Adams, “Easy SOA: Rapid Prototyping environment with Web Services for End Users” in *2006 IEEE International Conference on Web Services (ICWS’06)*, Chicago, IL, 2006, pp. 931–932.

[23] C. Bogart, M. Burnett, A. Cypher, and C. Scaffidi, “End-user programming in the wild: A field study of CoScripter scripts” in *2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008*, Herrsching am Ammersee, 2008, pp. 39–46.

[24] G. Vulcu, S. Bhiri, M. Hauswirth, and Z. Zhou, “A User-Centric Service Composition Approach”, In: Meersman R., Tari Z., Herrero P. (eds) *On the Move to Meaningful Internet Systems: OTM 2008 Workshops. OTM 2008. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 5333, pp. 160–169.

[25] N. Mehandjiev, F. Lecue, U. Wajid, and A. Namoun, “Assisted Service Composition for End Users” in *2010 Eighth IEEE European Conference on Web Services*, Ayia Napa, 2010, pp. 131–138.

[26] C. Marin and P. Lalanda, "DoCoSOC- Domain Configurable Service-Oriented Computing," in *IEEE International Conference on Services Computing (SCC 2007)*, Salt Lake City, UT, 2007, pp. 52-59.

[27] T. Nestler, M. Feldmann, G. Hübsch, A. Preußner, and U. Jugel, “The ServFace builder-A WYSIWYG approach for building service-based applications”, In: Benatallah B., Casati F., Kappel G., Rossi G. (eds) *Web Engineering. International Conference on Web Engineering 2010. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 6189, 2010, pp. 498–501.

[28] F. Daniel, J. Yu, B. Benatallah, F. Casati, M. Matera, and R. Saint-Paul, “Understanding UI Integration: A survey of problems, technologies, and opportunities,” University of Trento, Departmental Technical Report, Oct. 2006.

[29] T. Nestler, L. Dannecker, and A. Pursche, “User-centric composition of service front-ends at the presentation layer”, In: Dan A., Gittler F., Toumani F. (eds) *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. ServiceWave 2009, ICSOC 2009. Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, Vol. 6275, pp. 520–529.

[30] S. Pietschmann, T. Nestler, F. Daniel, “Application composition at the presentation layer: alternatives and open issues", in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services*, ACM, New York, NY, USA, pp. 461-468.

[31] N. Laga, E. Bertin, R. Glitho, and N. Crespi, “Widgets and composition mechanism for service creation by ordinary users,” *IEEE Communications Magazine*, Vol. 50 No. 3, 2012, pp.52-60.

[32] Z. Zhai, B. Cheng, Y. Tian, J. Chen, L. Zhao, and M. Niu, “A Data-Driven Service Creation Approach for End-Users”, *IEEE Access*, Vol. 4, 2016, pp. 9923–9940.

[33] S. Pietschmann, M. Voigt, and K. Meißner, “Dynamic Composition of Service-Oriented Web User Interfaces”, in *2009 Fourth International Conference on Internet and Web Applications and Services*, Venice, 2009, pp. 217–222.

[34] Sun Microsystem, “A Simple Example: HelloWorld.” [Online]. Available: http://www.inf.fu-berlin.de/lehre/SS03/19560-P/Docs/JWSDP/tutorial/doc/JAXRPC3.html. [Accessed: 30-Jul-2019].

[35] M. Q. Wang Baldonado, A. Woodruff, and A. Kuchinsky, “Guidelines for Using Multiple Views in Information Visualization” in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ACM, New York, NY, USA, 2000, pp. 110–119.

Appendix A: Design of Data Collection Instrument of Pilot Study

Table A.1: Design of Data Collection Instrument of Pilot Study

Overview Section
This section presents a brief introduction to the study. It comprises mainly of the purpose of the study, participant recruitment criteria, terms and condition of participation, estimated participation time (30 to 45 minutes), and researchers' contact details.

Section 1: Introduction to QuickWSC
A user guide that describes the main functionalities of QuickWSC.

Section 2: Using the tool to build an application by aggregating web services
This section contains the tasks that a participant needs to perform: Use QuickWSC to build or compose an application by aggregating relevant web services based on a given scenario; to execute the composed services to check on the composition result; to state the start time and end time of buiding the application; and to engage in think-aloud protocol during the composition process.

Section 3: Questionnaire (Part A)
Part A of the questionnaire comprises of the three questions below. The purpose of these questions is to collect the information about a participant' educational background, programming experience, and level of computer skill.

1. Educational background
   i. What is the name of your degree and major? (for e.g. *Bachelor of Science in Physics)* _________
   ii. What is your current level of study?
      a. Undergraduate year 1
      b. Undergraduate year 2
      c. Undergraduate year 3
      d. Undergraduate year 4
2. How much experience do you have in programming?
   a. I have not learned programming before.
   b. I learned programming on my own and can write simple programs.
   c. I learned programming through formal education/training and can develop complete software applications.
   If you choose b. or c, please state the respective programming language(s):___________
3. What is your level of computer skills?
   a. Fundamental (typing and using the mouse)
   b. Basic (Internet, Email, Hardware, Software Concepts, Word Processing, Formatting, Presentations, Graphics, Multimedia, Spreadsheets)
   c. Intermediate (Internet, Email, Hardware, Software Concepts, Terminology, Word Processing, Formatting, Tables, Presentations, Graphics, Multimedia, Spreadsheets, Databases)
   d. Advanced: (Internet, Email, Hardware, Operating Systems, Maintenance, Word Processing, Formatting, Tables, Presentations, Graphics, Multimedia, Spreadsheets, Databases)
   e. Proficient: (Internet, Email, Hardware, Operating Systems, Maintenance, Word Processing, Formatting, Tables, Presentations, Graphics, Multimedia, Spreadsheets, Databases, Programming)

Section 3: Questionnaire (Part B)
Part B of the questionnaire comprises of the fourteen questions below. Questions 1 – 12 require a participant to rate the features of QuickWSC on a 5-point Likert scale (1 - "Strongly Disagree", 2 – "Disagree", 3 – "Neither Agree nor Disagree", 4 – "Agree" and 5 - "Strongly Agree") and to state the reason if he or she choose the option of 3 or below.

Questions 1 – 5 are related to the ease of use of the respective feature of QuickWSC. These questions are used to evaluate the usability or ease of use of QuickWSC. Questions 7 –11 are related to the visualization aspect of QuickWSC. These questions are used to evaluate whether QuickWSC provides a clear visualization of the items configured by a user (such as GUI of web services and the three types of flows), which would affect the ease of use of QuickWSC. Question 6 (whether the two views/canvases are synchronization) and Question 12 (whether the execution process of the composed services is based on the three flows configuration) are used to evaluate the reliability of QuickWSC in terms of the synchronization between the two views, and, the execution of the composed services. Question 5 and Question 13 are used to evaluate whether the three flows configurations in the integrated two views eases and helps in composing the application. Question 14 is used to check whether it is useful to have an EUSC tool that helps in creating the UI of the composed application when the services are being aggregated.

1. It is easy to find the relevant web services for the scenario. If your response is 3 and below, please state the obstacles of finding the relevant web services.
2. It is easy to specify the application flow in the “UI of composed services” canvas. If your response is 3 and below, please state why it is not easy to specify the application flow in the respective canvas.
3. It is easy to specify the control flow in the “UI of composed services” canvas. If your response is 3 and below, please state why it is not easy to specify the control flow in the respective canvas.
4. It is easy to specify the data flow in the “workflow” canvas. If your response is 3 and below, please state why it is not easy to specify the data flow in the respective canvas.
5. It is easy to compose the application by specifying the three flows (application flow, control flow and data flow) in the integrated two views (workflow canvas and UI of composed services canvas). If your response is 3 and below, please state why it is not easy to compose the application by specifying the three flows in the integrated two views.
6. The web service information is synchronized between the “workflow” canvas and “UI of composed services” canvas. If your response is 3 and below, please state what web service information is not synchronized between the two canvases.
7. “UI of composed services” canvas provides a clear visualization of the GUI of the web services on an application page. If your response is 3 and below, please state why the “UI of composed services” canvas does not provide a clear visualization of the GUI of an application page.
8. “UI of composed services” canvas provides a clear visualization of the application flow of the composed services. If your response is 3 and below, please state why the “UI of composed services” canvas does not provide a clear visualization of the application flow of the composed services.
9. “UI of composed services” canvas provides a clear visualization of the control flow of the composed services. If your response is 3 and below, please state why the “UI of composed services” canvas does not provide a clear visualization of the control flow of the composed services.
10. “Workflow” canvas provides a clear visualization of the data flow of the composed services. If your response is 3 and below, please state why the “Workflow” canvas does not provide a clear visualization of the data flow of the composed services.”.
11. The three flows (application flow, control flow and data flow) are displayed clearly in the integrated two views (workflow canvas and UI of composed services canvas) during design time. If your response is 3 and below, please state which part(s) is/are not displayed clearly in the integrated two views during design time.
12. The execution process of the composed services is based on the three flows configuration (application flow, control flow and data flow). If your response is 3 and below, please state why you say that the execution process of the composed services is not based on the three flows configuration.
13. Does having the three flows configurations integrated in the two views help you in composing the application? Yes/No. Please state your reason.
14. Is it useful to have an end-user service composition tool that helps you to create the user interface of the composed application when you are composing the services? Yes/No. Please state your reason.