Journal of High Speed Networks 21 (2015) 83–94
DOI 10.3233/JHS-150517
IOS Press

A data extraction and layout generation framework for web service

Wu Gui-Cheng * and Yang Jie
School of Software Engineering, South China University of Technology, Guangzhou, China

Abstract. SOA web services are widely used as the new platform to build interoperable distributed applications which can eliminate the differences between the enterprise systems. Web services deployed over the network are accessible for a wider user base. However, the user interfaces are still implemented manually, and concepts that aim to automate this process are still in their infancy. In this paper, we introduce a web service customization method with a UI display and layout model as well as a UI layout algorithm. By using web service, we allow generation of form controls such as selection lists, text boxes, text areas, checkboxes and radio buttons on the client side. The UI layout algorithm is described and the time consumed by building service is compared in table. Performance tests are given in different testing environments. The experimental results demonstrate that our generated web services are efficient and effective in interoperability, flexibility and visualization.

Keywords: Data extraction, layout generation, web service

1. Introduction

Due to the convenience and flexibility, web services have gained a lot of interest in industry. Web services allow the creation of interactive service-based applications by combining the functionality offered by different web services. Using the idea of social network and technology of semantic web, a service network is developed to bridge the gap between service consumers and service providers [19]. In order to effectively process web services, an interaction model of context-aware web services based on context-aware process network is presented [24]. Based on behavior network, Jung and Cho [10] develop an application using Amazon web services.

Web services facilitate enterprise-class applications as well as Internet information resources integration. To cater the need of different kinds of users, multiple customizations of the base functionality are required. However, it turns out that many existing web services are heavily relied on data sources, which are not part of the web service specifications, disabling direct clients access. Moreover, the user interfaces are still implemented manually, and concepts that aim to automate this process are still in their infancy.

Based on studies conducted by Lawrence and Giles [11], a tremendous amount of content on web is dynamic. However, as Vaculin et al. [18] point out, majority of dynamic web pages and dynamic forms rely on "hidden" data sources are not well described as part of the web pages. Therefore, data residing in such data sources are very hard to access in an automated way.

A lot of research has been studied in the data mining field for dynamic web data extraction. Raghavan and Garcia-Molina [14] address the problem of extracting content from the hidden web by providing a framework. Hammer et al. [6] propose the usual form of extraction as wrapper. The wrapper requires users to customize extraction rules, and a graphical interface can assist users to define these rules. Xu et al. [22] introduce a method that generates wrappers automatically for HTML files. The wrapper is generated from label training examples. Shaohua et al. [15] propose a browser-based data extraction operation on the target HTML document directly in

*Corresponding author. Tel.: +86 13828401506; E-mail: 878100317@qq.com.

the browser. The markup language and software tools that support web information resource services are presented in document [2].

Horovcak et al. [8] propose a way of creating elements <select> and <input type = "radio"> based on defined database table columns. This web service is designed for quick and easy creation of HTML form elements. Principle of solution is based on the idea of creation a string that corresponds to HTML form input element so that the corresponding values of particular options are derived from a specified database table.

He et al. [7] propose a framework of service customization. Firstly, display preferences and constraints are generated based on input specifications and device characteristics. Secondly, a decision algorithm is invoked to generate optimal GUI layout. The framework demands that a WS specifications document must be designated before the web service is generated. So we think it is not compatible with the existing web services effectively.

Users of service tools are always the persons with some programming experience. The service customization work is also depending on other integrated development environment, such as the support of the eclipse. With the increasing users hung on mobile phones and other mobile devices, Farley and Capp [4] describe the application of web services on mobile devices. The web service is a HTTP-based service which is composed of service requests and data responses. For better user experience, graphical user interface should be designed.

The capability matching methods of web service are illustrated in [5] and [9]. Banati et al. [1] have extended WSDL and WSDL-T to WSDL-TC (extension of WSDL-T over a period of time) that aims at reducing the cost by maintaining the different collaborative customized versions of operations of the web service in a single deployment. The paper gives an example of hotel reservation web service to present WSDL-TC approach.

Liu and Chen [13] use agent to interpret the static OWL-S description, and the Belief-Desire-Intention (BDI) model is applied to develop BDI agent. This paper proposes an ontology-based BDI agent architecture, in which a BDI agent dynamically generates customized workflow, and binds semantic web services. Wang et al. [20] introduce a novel QoS measure approach to efficiently measure QoS of web service. The core of this approach is to take the three factors, service providers, the context of customers and historical statistics into QoS measure.

Li and Liu [12] propose an automatic generation system for web page code on the basis of XSLT technology and java programming language. It also describes a practical implementation of the system. Troncoso and Navon Cohen [17] describe the proposed architecture and a prototype implementation to support client side customization.

Dannecker et al. [3] aim at providing UI-related service annotations to improve the quality, and decrease the effort of UI generation for web services. The categories of Service annotations include visual annotations, behavioral annotations and relational annotations. Song and Lee [16] propose a method of generating XForms-based GUIs from WSDL files. The proposed method consists of three phases: parsing a WSDL file with an XML schema, generating XForm codes, and embedding it into a host language document.

According to [21], web services Customization approach follows three steps. Firstly, conduct a dialogue with individual customers to help them articulate their needs. Secondly, identify the precise offering that fulfills those needs. Thirdly, make customized web services for them.

Based on the research above, we propose a platform, which combines dynamic data extraction and layout generation by using web service. Our platform is achieved by the following steps. Firstly, provide a service customization website. Secondly, define HTML data extraction rules. Thirdly, generate web service code and WSDL documents by extraction rules. Finally, generate the interface layout automatically.

In the document [21], the authors concentrate on automatic layout generation method by providing a UI layout markup language and a UI layout algorithm. During the generation process, this UI layout language and algorithm only consider the size and position attributes, which limit the usability and user satisfaction level in some content and it does not involve the method of extracting the web data from the form controls. However, in this paper, we describe the solution to extracting data from form's controls such as selection lists, checkboxes and radio buttons in Section 2. In Section 4, we propose a new UI layout and display model that provides the means to describe UI elements, UI element layout hierarchy, layout constraints and layout preference. In our layout algorithm, we first process the controls within a vertical or horizontal group, and then process other controls to determine their position.

```
<td>
 <select id="Select1" name="ChinaCity">
   <option value="city1">Beijing</option>
   <option value="city2">Shanghai</option>
   <option value="city3">Guangzhou</option>
   <option value="city4">Shenzhen</option>
   <option value="city5">Chongqing</option>
   <option value="city6">Wuhan</option>
   <option value="city7">Xian</option>
 </select>
</td>
```

Fig. 1. The HTML of a selection list. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JHS-150517.)

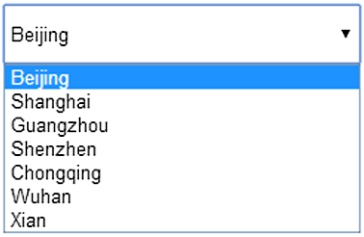

Fig. 2. The layout of a selection list. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JHS-150517.)

2. Extract data from the form controls

Traditionally, most approaches try to locate the data items in a "certain" way. That is, after a certain tag or a certain delimiter, there is the data we need until another delimiter should appear, and we know this delimiter means the end of data. The advantage of this extraction method is that they could find data items precisely when the web pages are not changed, but at the same time, most of them could not handle more flexible web pages, especially there are missing tag or out of order tags in the HTML files.

In order to extract data from forms controls efficiently, the form F is introduced which is modeled as a set of (element, data) pairs; $F = \{(E_1, D_1), (E_2, D_2), \ldots, (E_n, D_n)\}$ where the E_i means the unique ID of a specific element and D_i indicates corresponding data values in E_i. The data D_i of an element E_i is the set of values which is associated with corresponding form element. For example, Fig. 1 is a piece of HTML markup that was used to generate a selection list. In our form, we assume E_1 to represent the unique ID of the selection list, that is E_1 = "Select1", then D_1 is the data set corresponding to E_1, D_1 = {"Beijing", "Shanghai", "Guangzhou", "Shenzhen", "Chong qing", "Wuhan", "Xian"}. Similarly, data value in other controls such as RadioBox, checkbox, TextArea could be stored.

During the service customization process, the form F is generated by the web server and recorded as a part of customization documents. When the layout generation service is processed on the client side, the form f is retrieved to fill in the forms controls. Figure 2 shows the layout of a selection list on the client side.

3. Service customization

3.1. Service customization model

In order to package information source into web service, the service customization model is derived to record the customization process information generated by user in web service. These customization process information such as the parameters and attributes of web services would be needed when generating web services automatically.

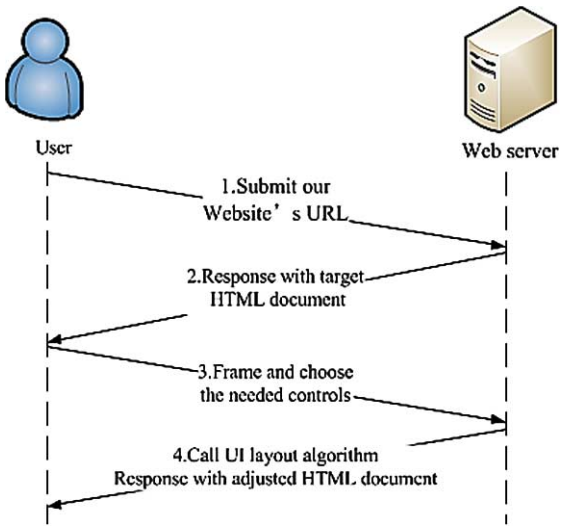

Fig. 3. The process of service customization. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JHS-150517.)

The service model is defined as the 6-tuple: $CM = (INP, OUTP, XT, OP, CONF, UI)$, in which the *INP* states the detailed information of service input operation pages, the *OUTP* states the detailed information of output pages. *INP* and *OUTP* are identical in structure. They include the URL of the page, the request method and other parameters. *XT* is the variable set. Each element in the set is corresponding to an information node's path of the DOM tree in the output page. *OP* represents the operation of the service, including the type of input parameters in the input operation, and the type of output service. *CONF* contains configuration information for the service.

3.2. The process of service customization

We would rather choose to obey the existing web standards, instead of creating a new protocol or specification. In this way, web browsers could remain unchanged. The process is based on existing web standards (HTTP, HTML, JavaScript, AJAX). It can be implemented in existed web infrastructure with a minimal impact on the client side.

The process of customization service (shown in Fig. 3) is as follows: firstly, users submit the URL of the page clarifying the web information resources in which they are interested. The URL contains the target link or dynamic form-based web page such as online dictionaries, trips inquiries. Secondly, server-side web agent obtains the original target HTML document via an HTTP request, and returns response page embedded with the AJAX dynamic interactive code to the users. During the process, the related information of input page is stored in *INP*. The user can frame and choose the marquee target form or link with the mouse easily. Thirdly, the user enter the appropriate parameters in the form and then submit it. Finally, the web proxy again request for echo to the HTML document with dynamic interaction code embedded. The user can click on whatever information points of interest after that. The output page information is stored in the *OUTP*. During the whole process, the daemon program will record all operations of the user, and define the data extraction rules.

3.3. Dynamic interaction and data-extraction rule definition

The interaction of the input page: the user can frame and choose his marquee target form or link controls with the mouse in the input page (the system uses JavaScript marquee model to identify the user selected controls).

After the coordinate matching, the system will judge whether the form is targeted. If the form is targeted by the user, the relevant parameters and operating information of the current form are extracted and stored into the *INP* and *OP* of services customized model.

The interaction of the output page: dynamical interaction features are provided based on mouse clicks. When the user chooses the interested data items with a mouse click, the *XPATH* expression of the chosen DOM node gets stored in the *XT* service customization model using AJAX.

When the operation is finished, the extraction rules definition process is finished at the same time.

3.4. Service automatically generation

Similar to the compiler generator, the input of the service builder is all the data in the service customization model and the output is the code about web service written in Java, and WSDL (web service description document). The service builder generates complete web services and unit test cases according to service customization model, data extraction rules, and server configuration information. The generated web service can be run after the deployment in server-side is finished.

4. The UI display and layout model

In this section, a UI display and layout model provides a way to describe UI elements, layout hierarchy, layout constraints and layout preferences. In the UI display and layout model, each UI object represents a single web service user interface element such as Label, Textbox and Button. The layout hierarchy describes the relationship among UI elements as a hierarchy, which provides group layout preference information. The constraints are used to represent layout features that must meet in the UI layout. The preferences describe layout features that are preferred.

4.1. UI element's attributes

For each UI element, we identify three types of attributes to represent its display and layout features: style attributes, size attributes and position attributes.

The style attributes describe the UI element's features such as color and font. The HTML could be used as the markup language to describe the style attributes. For example, *<input id = "Text1" style = "font-family: 'Times New Roman', color: red"/>* describes a textbox with red color and 'Times New Roman' style font. A membership function is used to specify the preference in display styles for UI element. Each possible display style of the UI element is assigned a weight. If a style is more preferred than other styles, then a higher weight is assigned. The final layout is partially based on the style weights.

As in the common web-based user interfaces, each UI element occupies a rectangle block in the GUI. The size attributes describe the UI element's width and height. The process of deciding the size attributes is as follows. The server-side program extracts and records the width and height of all the UI elements on the page. We then define the width set as W, the height set as H, and the reference units as S. The reference unit is defined as

$$S = \min\big(\min(W), \min(H)\big). \tag{1}$$

The web page pops up the recommended layout operating area in the form of grid. A reference unit is corresponding to two grids. The following example will illustrate how a specific UI element is placed in the grid. Assuming the width and height of an UI element are x and y respectively, we set x as the minimum value of the widths and heights in all UI elements, in another words, x is the reference unit. We assume this UI element will occupy n grids. The number of grids n is defined as

$$n = 2 * \left(\frac{2 * y}{x}\right). \tag{2}$$

By using the reference unit S and the width W_i of each UI element, the grid width $Nw(i)$ for each UI element can be calculated. In a similar way, the grid height $Nh(i)$ for each UI element can be calculated using the reference unit S and the height H_i of each UI element. Each UI element's grid width $Nw(i)$ and grid height $Nh(i)$ is defined as

$$Nw(i) = \left(\frac{2 * W_i}{S}\right), \tag{3}$$

$$Nh(i) = \left(\frac{2 * H_i}{S}\right). \tag{4}$$

The system extracts the original web page layout information and shows the corresponding built-in UI elements in the form of grid layout. The user can modify the recommended layout. If the user is satisfied with the default layout, it will be recorded; otherwise, the user can drag the UI elements according to his own preference. Such layout strategy using the preference of user can obtain a high degree of satisfaction. The system does record every modified UI element's attributes, including grid horizontal coordinate $Rx(i)$, grid vertical coordinate $Ry(i)$, grid width $Nw(i)$, grid height $Nh(i)$ and its type, and then stores them in the *UI* layout markup document.

We provide the position layout guidelines by using the gravity method: the UI elements of larger size such as TextArea are given high weights, while the small ones such as label are given low weight. After the UI elements are given weights, the layout's center of gravity will be calculated, which should coincide with the center of the screen during the generation of the client interface. The purpose is to make the layout more balanced as the larger elements are closer to the center of the screen while the small ones are closer to the screen margin. For the weight is the direct proportion relationship with the area of the element, so we can obtain the weight $W(i)$ of each element by multiplying the grid width $Nw(i)$ with the grid height $Nh(i)$. We define the weight $W(i)$ as

$$W(i) = Nw(i) * Nh(i). \tag{5}$$

Then the horizontal coordinate of the layout's center of gravity Cx and the vertical coordinate of the layout's center of gravity Cy is derived. The layout's center of gravity (Cx, Cy) is given as

$$Cx = \frac{\sum W(i)Rx(i)}{\sum W(i)}, \tag{6}$$

$$Cy = \frac{\sum W(i)Ry(i)}{\sum W(i)}. \tag{7}$$

The server-side program then writes the coordinates of each UI element to the UI layout markup document, relative to the layout's center of gravity.

We also provide some position guidelines for some special UI elements, such as the title element. These guidelines use abstract position concepts. Nine values are defined for position attribute specifications as: "topcenter", "topleft", "topright", "middlecenter", "middleleft", "middleright", "bottomcenter", "bottomleft", "bottomright". They are self-explained from their name.

4.2. Layout hierarchy

We consider the layout hierarchy as nested groups. We define two hierarchy types: horizontal group, vertical group. Different group types represent different layout preferences for the UI elements in the group. The UI elements belonging to a vertical group will be placed in the same column. The UI elements belonging to a horizontal group will be placed in the same row.

4.3. Constraints

Some constraints are general to the entire layout generation, and should be included in the UI display and layout model. There are three general constraints below:

- Screen height constraint: All UI objects should be placed within the height of the screen.
- Screen width constraint: All UI objects should be placed within the width of the screen.
- Non-overlapping constraint: All UI objects should not be overlapped.

The problem is how to identify the general layout constraints that are applicable to a specific layout scenario. If a horizontal scrolling is allowed, then the screen width constraint is not applicable. Similarly, if a vertical scrolling is allowed, then the screen height constraint is not applicable. In this paper, we choose to allow vertical scrolling as most user interface does.

4.4. Preferences

Three types of preference have been identified. The first one is the individual display preference. It specifies the style attribute of the individual UI element. The second one is the group layout preference. It represents the style attribute of the layout hierarchy. The third one is the global layout preference. It makes the layout decision as a whole. Considering that the preferences could be conflicting with each other, we assign a preference level to each preference in our framework. If a preference is more important than other preferences, then a higher preference level is assigned. We think that the global layout preference is the most important one, so level value 3 is assigned to it. Similarly, level value 2 is assigned to group layout preference. Level value 1 is assigned to individual display preference. In this way, the conflicting preferences could be solved.

4.5. Define model

Based on analysis above, the UI display and layout model M is defined as $M = (O, H, C, P, OB)$, where O stands for the set of UI elements, we use $O = \{o_1, o_2, \ldots, o_n\}$ to denote the set of UI elements. As discussed above, each UI element o_i has three types of attributes: style attribute, size attribute and position attribute. These attributes together express the UI element's display and layout features. H stands for the UI element layout hierarchy, which describes the relationship among UI elements as a hierarchy and provides group layout preference information, C represents the set of constraints, P represents the set of preference and OB stands for the layout objective.

The UI display model is decided through maximizing the overall preference satisfaction level as well as meeting all constraints. This is achieved by defining the layout objective as a function of the overall preference satisfaction level. The layout objective function OB is defined as

$$OB = \sum_{i=1}^{n} t_i * k_i, \tag{8}$$

where n is the number of preferences, t_i is the weight of the ith preference and k_i represents the preference level of the ith preference.

5. The UI layout algorithm

From the service list on the mobile client, the user can "download" the web-based service applications running on the server. According to the user's choice of services, the client downloads the service's *UI* layout markup documents, extracts relevant information to generate the layout. Meanwhile, the client keeps the processed information

```
UI-layout algorithm
 1:  currentItem <- readFirstItem()
 2:  while i < numberOfItem do
 3:     processItem(currentItem)
 4:     if currentItem is newVerticalGroup.start
 5:        processVerticalGroup()
 6:     end if
 7:     else if currentItem is newHorizontalGroup.start
 8:        processHorizontalGroup()
 9:     end if
10:        location[i] <- readNextItem()
11:           i <- i + 1
12:  classifyByVerticalCoordinate(location,rowsOfItems)
13:     f <- 0
14:     row <- 0
15:     for j <- 0 to length[rowsOfItems] − 1
16:     left <- |rowsOfItems[j].leftMostRx() − rowsOfItems[j].leftMostNw()|
17:     right <- |rowsOfItems[j].rightMostRx() − rowsOfItems[j].rightMostNw()|
18:     if f < Max(left,right)
19:        f <- Max(left,right)
20:        row <- j
21:     j <- j + 1
22:  sum <- 0
23:  for j <- 1 to sumOfItems(rowsOfItems[row]) − 1
24:     sum <- rowsOfItems[row].item(j).Nw() + rowsOfItems[row].item(j + 1).X() −
25:        rowsOfItems[row].item(j).X() + sum
26:     sum <- sum + rowsOfItems[row].item(sumOfItems(rowsOfItem[row])).Nw()
27:     Sw <- getScreenWidth()
28:  For i <- 0 to numberOfItem
29:     Item[i].setWidth(item[i].Nw()*(Sw/(sum + 0.5)))
30:     Item[i].setHeight(item[j].getwidth()*item[i].Nh()/item[i].Nw())
31:     Item[i].setX(item[i].Rx()*(Sw/(sum + 0.5)))
32:     Item[i].setY(item[i].Ry()*(Sw/(sum + 0.5)))
```

Fig. 4. The UI layout algorithm.

records in the local documents. Figure 4 is the client's UI layout algorithm for the auto-generation of client UI, in which some parameters are taken from the UI layout markup documents.

(1) Extract the UI elements from the UI layout markup document. If the current UI element is the start of a vertical group, then the method "processVerticalGroup" will be called. After the method "processVerticalGroup" is called, the UI elements belonging to a vertical group will be placed in the same column. Similarly, if the current element is the start of horizontal group, the method "processHorizontalGroup" will be called. After the method "processHorizontalGroup" is called, the UI elements belonging to a horizontal group will be placed in the same row.

(2) Extract the relative grid coordinates from the document and make statistics in groups in accordance with the value of the grid vertical coordinate $Ry(i)$. $Rx(i)$ and $Nw(i)$ are the grid horizontal coordinate and the grid width of the UI element respectively, then we calculate f in each group:

$$f = \mathrm{Max}\big(Left = \big|Rx(i) - 0.5 * Nw(i)\big|, Right = \big|Rx(i) + 0.5 * Nw(i)\big|\big). \tag{9}$$

In this way, we can find the largest group f. The largest group means it has the largest layout width, then we can calculate the value *sum* as

$$sum = \sum_{1}^{n} Nw(i) + \sum_{1}^{n-1} Rx(i+1) - Rx(i), \tag{10}$$

where n is the number of UI elements in the largest group f, $Nw(i)$ is the width of the ith element in the largest group f sorting from left to right, $Rx(i+1)$ and $Rx(i)$ are separately the grid horizontal coordinates of the ith and the $(i+1)$th elements in the largest group f sorting from left to right.

(3) Get the phone screen resolution ratio of the phone screen *Sw*, and then write *Sw* to the *config* attribute in the layout document.

(4) The actual width $Rw(i)$ and actual height $Rh(i)$ of each UI element are

$$Rw(i) = Nw(i) * \left(\frac{Sw}{sum + 0.5}\right), \tag{11}$$

$$Rh(i) = Rw(i) * \frac{Nh(i)}{Nw(i)}. \tag{12}$$

(5) Building the coordinate system in the center of the screen, and $Rx(i)$, $Ry(i)$ are respectively the relative horizontal and vertical coordinates extracting from the document. The actual coordinates $X(i)$ and $Y(i)$ of each UI element are

$$X(i) = Rx(i) * \left(\frac{Sw}{sum + 0.5}\right), \tag{13}$$

$$Y(i) = Ry(i) * \left(\frac{Sw}{sum + 0.5}\right). \tag{14}$$

(6) Finally, *Rw*, *Rh*, *X* and *Y* of each UI element as mentioned above are written to the layout document. The client can directly read the parameter-generating layout from the document every time the service is executed.

Using our platform, users can easily encapsulate their own interested web page into web service, and generate the interface on the phone client automatically. The generated interface can meet the basic functional requirements. Mobile clients do not need to develop applications specifically for the web service. It is both practical and entertaining. Quantitatively analyzed, this platform can reduce the time in transforming web page into web service and deploying web service on the phone. The people who customizing the web service on our platform are the ordinary users (not developers). The time table is shown in Table 1. From the table, the time is spent on defining the input and output rules and setting UI layout. We select four common web services to test the platform when building the web services on the Internet. We test the web service when it executes on the mobile client. Moreover, we also analyze the performance by measuring the time consumed in each logical unit.

From the mobile client's view, web service running includes the following stages:

Table 1
Time of build service

Service name	Input (minute)	Output (minute)	UI layout (minute)	Total time (minute)
E-dictionary	1	2	2	5
Flights	2	3	2	7
IP query	1	1	1	3
Forecast	2	3	2	7

Table 2
The testing environment

Tool	System	Memory	CPU
Cellphone	Android 2.3	512 MB	1 GHz
PC	Window Server 2008	2 GB	2.4 GHz

Table 3
Performance test

Service name	Number of UI controls	Size of WSDL document (byte)	Time of parsing WSDL (ms)	Time of data extraction (ms)	Time of interface generation (ms)	Total time (ms)
Dictionary	5	60,416	129	202	673	1004
Flight	9	37,417	76	172	762	1011
IP query	4	16,232	34	65	521	620
Forecast	8	72,129	165	232	714	1111

(1) Generate service interface automatically. In detail, the user finds the target service on the client's service list. Then the client generates interface automatically according to the service's UI layout markup document and WSDL document. The time consumed depends on the complexity of the interface.
(2) Obtain and parse HTML document. It is processed on the server. System uses web proxy to get target HTML document. This process is related to the current network conditions.
(3) Extract and encapsulate data into web services. This process is also processed on the server. The time depends on the number of data extraction rules. We take the time consumed in various stages as the indicators of the performance evaluation.

For the whole service customization process on the client side, the time tests are carried out in laboratory (testing environments are shown in Table 2). During the test process, only one service on the server is run. The five different sets of input parameters are used on the mobile client to test the performance. From Table 3, when the client uses a web service for the first time, the service interface generation process spends more than 50% of the total web service operating time. However, due to the generated web service interfaces storing in the local client side, the time efficiency of the web service will be largely improved when the web service is used again. This property is attractive to the service providers who pay high attention to cultivating users. The evolutionary technology may be used as a competitive strategy to secure customer loyalty.

6. Conclusion

Web services deployed over the web are accessible to a wilder user base. Web services are designed to meet the need of large number of users. Most of time, different customizations of basic functionality are required to cater the requirements of multiple sets of users. It will force service providers to develop many web services for each collection of users, leading increasing cost of infrastructure and maintenance [1].

This paper introduces a web service customization platform which is consisted of service customization, a UI display and layout model and a UI layout algorithm. The customization of web service encapsulates the Internet information resources into web services for ordinary users. Compared with the traditional web service wrappers, our web services generator is used on the mobile platform, and can automatically generate the service interface. We select four common web services to test the platform's performance on the Internet. We test the web service when it executes on the mobile client. Moreover, we also analyze the performance by measuring the time consumed in each logical unit.

The experimental results demonstrate that the web services generated by the platform have high scores in the interoperability, flexibility and visualization.

We also give the time test for the whole service customization and application process on the client. During the test, we allow only one service on the server. For each service, five different sets of input parameters are used on the mobile client to test the performance. The services we tested are as follows: Electronic dictionary, Flights, The IP query, Weather forecast. From the test, when the client uses a web service process for the first time, the service interface generating process spends more than half of the total service operating time. Moreover, the customization method shows reusability in generating web services.

Web Service (WS) is used to support implementation of the distributed system and has already achieved interoperability with a variety of online platforms [23]. In the future, we will explore better layout strategy to improve customer satisfaction. We will also attempt to apply web services into different mobile platforms. Try hard to improve their adaptability and achieve platform-independence truly. Moreover, the objective of this work is far more than create web services according to the users' interests or business requirements. They can run on mobile clients now. And in the near future, they may run in other environment, such as clouds. That may help in achieving high degree of customer satisfaction while reducing the cost for service providers.

Acknowledgements

This work is supported by the SNSF (the National Natural Science Foundation of China) X2jsB55101680 and FRFFCU (the Fundamental Research Funds for the Central Universities) x2rjD2116860.

References

[1] H. Banati, P. Bedi and P. Marwaha, WSDL-TC: Collaborative customization of web services, in: *International Conference on Intelligent Systems Design and Applications*, 2012, pp. 692–697.
[2] R. Baumgartner, S. Flesca and G. Gottlob, Visual web information extraction with Lixto, *The VLDB Journal* **1** (2001), 119–128.
[3] L. Dannecker, A. Preußner and T. Nestler, Service annotations for improving the generation of web service user interfaces, in: *Web Services*, 2010, pp. 197–204.
[4] P. Farley and M. Capp, Mobile web services, *BT Technology Journal* **23** (2005), 202–213.
[5] X. Gao, Y. Jian and M.P. Papazoglou, The capability matching of web service, in: *Proceedings of the IEEE Four International Symposium on Multimedia Software Engineering*, 2002, pp. 56–63.
[6] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha and A. Crespo, Extracting semistructured information from the web, in: *Proceedings of the Workshop on Management of Semistructured Data*, 1997, pp. 18–25.
[7] J. He, I. Yen, L. Peng, T. Dong and F. Bastani, An adaptive user interface generation framework for web services, in: *Congress on Services Part II*, 2008, pp. 175–182.
[8] P. Horovcak, J. Terpak and B. Stehlikova, Generation of form's input elements select and radio type using web service, in: *Carpathian Control Conference (ICCC)*, 2013, pp. 123–127.
[9] J.Q. Hu, P. Zou, H.M. Wang and B. Zhou, Research on web service description language QWSDL and service matching model, *Chinese Journal of Computer* **28** (2005), 39–47.
[10] M.-C. Jung and S.-B. Cho, A novel method based on behavior network for web service composition, in: *Next Generation Web Services Practices*, 2005, pp. 6–11.
[11] S. Lawrence and C.L. Giles, Accessibility of information on the web, *Nature* **400** (1999), 107–109.
[12] L. Li and Z. Liu, The research and application of web page code automatic generation technology, in: *Artificial Intelligence Management Science and Electronic Commerce*, 2011, pp. 5246–5249.
[13] C.-H. Liu and J.-Y. Chen, Using ontology-based BDI agent to dynamically customize workflow and bind semantic web service, *Journal of Software* **7** (2012), 884–894.
[14] S. Raghavan and H. Garcia-Molina, Crawling the hidden web, in: *VLDB*, 2001, pp. 129–138.
[15] Y. Shaohua, Z. Liyong and H. Yanbo, A markup language for generating web services out of web-based information, *Journal of Computer Research and Development* **43** (2006), 5–9.
[16] K. Song and K.H. Lee, An automated generation of XForms interfaces for web service, in: *Web Services*, 2007, pp. 856–863.
[17] R.M. Troncoso and J. Navon Cohen, Ubiquitous client side customization of web applications, in: *Web Systems Evolution*, 2011, pp. 87–92.
[18] R. Vaculin, T. Heath and R. Hull, Data-centric web services based on business artifacts, in: *Web Services*, 2012, pp. 42–49.
[19] H. Wang, Z. Feng, Y. Sui and S. Chen, Service network: An infrastructure of web services, in: *Intelligent Computing and Intelligent Systems*, 2009, pp. 303–308.

[20] S. Wang, Q. Sun and F. Yang, Quality of service measure approach of web service for service selection, *IET Software* **6** (2012), 148–154.
[21] G.-C. Wu, J. Yang and D.-Y. Tang, Web service automatically layout generation method, in: *Emerging Intelligent Data and Web Technologies*, 2013, pp. 6–10.
[22] Y. Xu and T.W. Ling, Using weight-controlled token matching to extract data from HTML files, in: *Web Information Systems Engineering*, 2001, pp. 341–349.
[23] T.H. Yang, C.Y. Ku, D.C. Yen and Y.C. Lin, Utilizing the interactive techniques to achieve automated service composition for web services, *Journal of High Speed Networks* **17** (2010), 219–236.
[24] X. Zhang, H. Liu and A. Abraham, A novel process network model for interacting context-aware web services, in: *Services Computing*, 2013, pp. 344–357.