T0804059

Towards a Component Model for Web-based Service Composition*

ZHAO qi[1], HUANG gang[1+], LIU xuanzhe[1], HUANG jiyu[1]

1. School of Electronics Engineering and Computer Science, Peking University; Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing, 100871, China

+ Corresponding author: E-mail: huanggang@sei.pku.edu.cn

一种基于 Web 的服务组装构件模型

赵祺[1], 黄罡[1+], 刘譞哲[1], 黄冀渝[1]

1.北京大学信息科学技术学院软件研究所 高可信软件技术教育部重点实验室 北京 100871

摘要：基于 web 的服务组装正成为一种流行的组装风格。许多已有的工作提出将服务业务逻辑和用户界面封装为一个基于 web 的服务构件，并在基于 web 的环境中组装这些构件。这些构件模型在复用，尤其是构件验证、适配、以及复合构件的支持方面仍存在许多局限。本文提出一种新型构件模型以支持基于 web 的服务组装。首先，该构件模型将用户界面和服务业务逻辑分离以获得更好的适配性。其次，开发人员不仅可以在业务逻辑的层次组装这些构件，还可以在用户界面的层次进行组装。同时，该构件模型支持实时组装以实现及时、有效的验证和适配。该构件模型还支持将组装结果发布为一个新的可复用的服务构件。
关键词：复用；服务组装；混搭

Abstract. The web-based service composition is becoming a popular composition style in Service Oriented Computing. Many existing work proposed encapsulating service business logic and User Interface (UI) into a single web-based service component and assembling these components in web-based environment. However, these component models are yet limited in terms of reusability, especially for component qualification, adaptation and composite structure support. This paper proposes a component model for web-based service composition. Firstly, we present a well-structured component model that decouples the UI and service business logic for better adaptation. Secondly, developers are able to assemble components not only at business logic level but also at UI level. And our component model also supports on-the-fly composition, which provides a quick and effective feedback way for qualification. Thirdly, the component model supports the composition result to be published as a new component for further reuse.
Key words: reuse; service composition; mashup

* the National Key Basic Research and Development Program of China (973) under Grant No. 2002CB312000（国家重点基础研究发展规划，973）; the High-Tech Research and Development Program of China (863) under Grant No. 2007AA010301 (国家高技术研究发展计划); the National Natural Science Foundation of China under Grant No. 90612011（国家自然科学基金）; and the IBM University Joint Study Program（IBM 大学合作项目）.

1 Introduction

Being one of the key ideas for Internetware, software reuse offers a great deal of potential in terms of software productivity and software quality [2]. Generally, there are two main reuse styles: composition-based reuse and generation-based reuse. Composition-based reuse has been more widely used in practice, such as Component Based Software Development (CBSD) and Service-Oriented Architecture (SOA). As a target application domain or computing paradigm of Internetware, SOA allows developing software systems by assembling loosely-coupled, gained services.

Currently, there are many services published via Internet with open APIs, such as Amazon S3 and Google Map. Therefore, more and more users are able to access these services through web as well as assemble them for their own applications, e.g., those so-called mashups [7].

Many existing work [3][4][5][6] proposed encapsulating service business logic and User Interface (UI) into a single component (called web-based service components, WBS component for short, in this paper). And then developers can create applications in web-based environment, such as web browser, in a more interactive manner. However, there still have many limitations in current WBS component model, especially for component qualification, adaptation and composite structure support.

Given a set of components, and a schema for assembling them, qualification is to check the proposed composition satisfies a given set of requirements [2]. Unlike traditional off-the-shelf components that need compile and deployment, services are actively running entities [11]. Such a significant difference makes service composition "on-the-fly" with immediate reaction. And that gives a quick and effective feedback way for qualification. First, a service can be invoked without deploying it and the result will be returned immediately. Therefore developers can qualify the service on the fly. Second, since services can be assembled and invoked just-in-time, developers can retrieve the real-time "execution result" of service composition once they assemble some services. Accordingly, the service composition can be qualified on-the-fly. In briefly, qualification can be carried out simultaneously with on-the-fly composition. However, the on-the-fly ability is not well supported in current WBS component models.

If retrieved components are sufficiently "close" to the requirements and are of sufficient quality, developers should adapt them [2]. WBS components encapsulate UI, while UI is the most variable element considering the fast evolving composition context. For example, a Scenic-Spots List WBS component may invoke a remote service to get scenic-spots data and display it in a list. When this component is used individually, the list is necessary for displaying data. When it is composed with a map WBS component, the scenic-spots can be displayed on the map and the scenic-spots list becomes useless and redundant. However, in most current WBS component models, such as current mashups, UI is described by fixed markup files and cannot be modified. On the other hand, WBS components also include some interaction logic which manipulates the relationship between UI and services. However, the interaction logic and UI in current WBS component models are tightly coupled. Hence, once UI is modified, the interaction logic will break down. In other words, these component models are not able to adapt flexibly in different scenarios.

Moreover, composite component is important in reuse, which allows developers to publish the composition result as a new component for further reused. In SOA, composite component is also one of the core principles and supported by the most process-based service composition, such as

Business Process Execution Language for Web Services (BPEL4WS). However, current WBS models do not support composite component yet.

In this paper, we present a novel component model to solve the above reusability limitations in web-based service composition. This component model is easy to adapt within different scenarios, supports on-the-fly service composition and composite component1.

The rest of this paper is organized as follows. Section 2 illustrates a sample scenario to explain the problems in web-based service composition we mentioned above. Section 3 gives an overview of our approach. Section 4 provides the details about component model. It also presents how it supports composite component and on-the-fly composition. Finally, we discuss related work in Section 5 and conclude in Section 6.

2 Motivation Scenario

We begin with a typical scenario to explain the problems in WBS composition mentioned above. This scenario is the development of a City Tourist Guide, as shown in Fig.1. In this scenario, there are three WBS components: A City Scenic-Spots List, which lists scenic spots of a selected city; an Image Searcher which searches and displays images by keywords; a Google Map which displays locations of given addresses. The City Tourist Guide requirement is that scenic spots of this city should be displayed on the map with images when a city is selected. It can be implemented by assembling the three components.

Fig.1 Motivation Scenario

图 1 动机场景

In this scenario, we can find the problems mentioned above. First, the geo-data of scenic-spots

1 A prototype is open source at http://sourceforge.net/projects/imashup/

and map may be mismatch, which makes location markers display at wrong places. In traditional composition environment, this mismatch cannot be discovered until the application is deployed. However, the on-the-fly capability makes developers be able to find the mismatch once the two components are assembled. Second, the UI should be able to be customized and assembled together, which means the images should be nested in the maps location markers and redundant UI elements (i.e. spots list and keyword input of Image Searcher) should be removed. Finally, composition result, the city tourist guide, may be further integrated with other components, such as an airplane ticket booking component. Current WBS component models do not support composite component. Therefore developers cannot use the city tourist guide as a WBS component but should assemble the three components again with the airplane ticket booking component.

3 Approach Overview

In this section, we give a brief overview of our approach, which is briefly illustrated in Fig.2. The Fig.3 is a snapshot of assembling the city tourist guide in our composition environment. The left part displays the composition result, while the right part is the management panels which allow developers control the components.

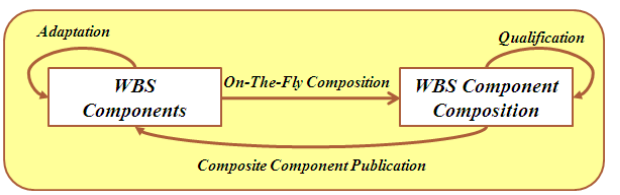


Fig.2 Approach Overview

图 2 方法概要

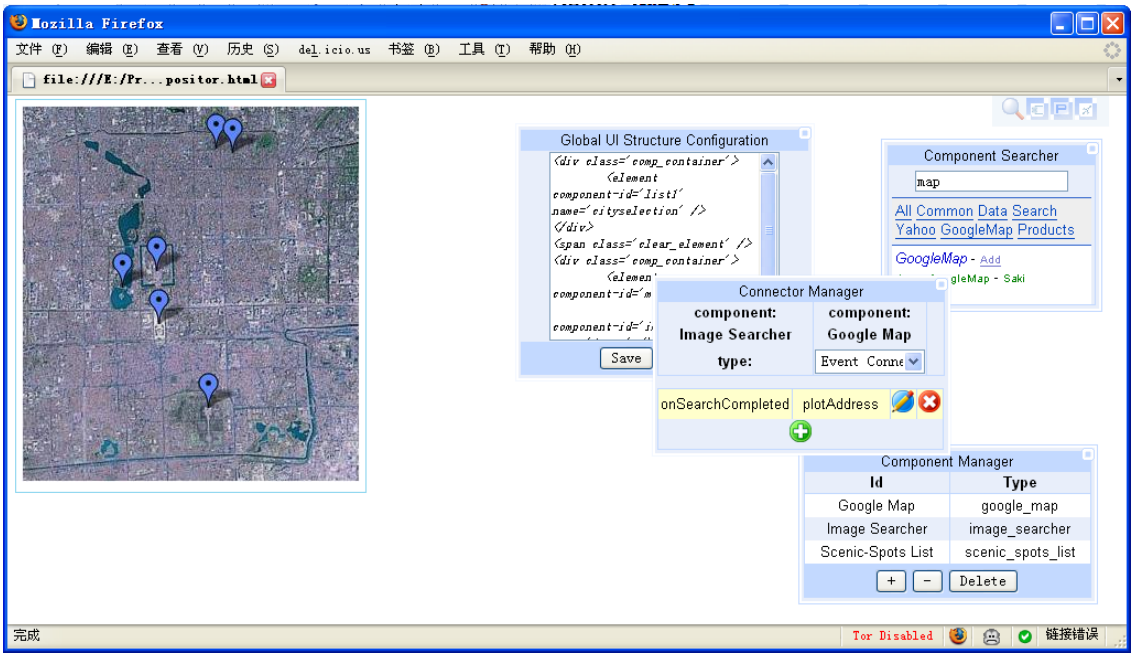


Fig.3 Composition Environment

图 3 组装环境

Considering our city tourist guide scenario, developers should firstly use the searcher panel to search and retrieve three components into environment. The components will be initialized once they are retrieved. And they will be displayed as A), B) and C) in Fig.1. Next, developer can interact with these components (i.e. searching some example keywords for images) for qualification. If the components do not satisfy the requirements completely, they can adapt with the configuration panel. In our scenario, developers will nest the image in the map and remove spots list and keyword input of Image Searcher. The adaptation result should be like the left part of Fig.3. And then developers can assemble the components. Since our component model and composition environment support on-the-fly capability, qualification is able to be carried out simultaneously with composition. For example, the scenic-spots list will connect with the Google map. Since on-the-fly composition support, the geo-data of scenic-spots list will be sent to Google map and location markers will display immediately once the connection sets up. If the geo-data of two components is mismatch, the location markers will display at wrong places. Developers can discover it just in time and modify the connector to resolve it. Finally, developers can publish the composition result as a composite component for further reuse. In addition, the management panels in our composition environment are also WBS components. And these components can be removed from the environment after composition completed. In that sense, our composition environment can be considered as an incubator, while this composition can be considered as evolving the incubator to the city tourist guide application.

4 Component Model

Like traditional component models, our WBS component model consists of two parts, interface and implementation. In contrast with common components, the interface of component model consists of User Interface and Programming Interface since WBS component model encapsulates UI. The Programming Interface exposes business logic of WBS component. WBS components interact with others by their programming interface. The UI responds to users' actions and invokes the corresponding functions in the implementation. The component model is shown in Fig.4.

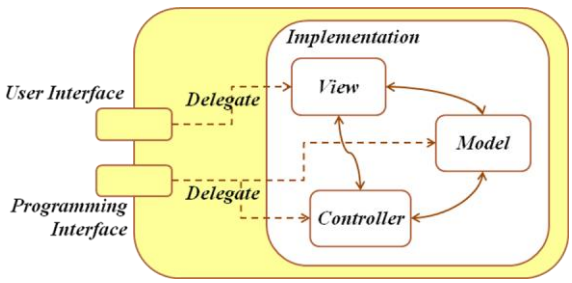


Fig.4 WBS component Model

图 4 WBS 构件模型

4.1 Implementation

The implementation of our WBS component model employs Model-View-Controller pattern so that presentation and functionality can be coupled in a loose way.

The model implements business logic by invoking a remove service. In the city tourist guide example, the Image Searcher component's model part is responsible for calling a remote Image search service (i.e. Flickr) to find images.

The view constructs and manages UI elements. An element of UI can be an atomic HTML element such as the button of the City Scenic-Spots List component, or a group of HTML atomic elements such as the scenic-spots list of the City Scenic-Spots List component.

The controller manages the interaction logic between model and view. The controller part consists of several element controllers. Each element controller encapsulates the interaction logic of one specific UI element. The Image Searcher component has two element controllers: one is a button controller which responds to button clicked event and calls image search functionality, the other is an image controller which gets return image link and displays the corresponding image.

Our WBS component model's UI can flexibly adapt with different scenarios, because the controller is divided into several element controllers and the view leaves the capability of configuring their structure and presentation to developers. We will discuss how to adapt UI in the following section.

4.2 Interface

The programming interface exposes the business logic of WBS component. It consists of

properties, methods and events. Properties describe the state of a component and can be queried and modified. Methods can query and modify the component state. Events notify changes of the component state. The Image Searcher Component's programming interface includes "keyword" property, "clickSearchButton" and "searchImages" methods, "onImageChanged" and "onSearchCompoleted" events.

An Interface of a reusable component consists of two parts [1]: the fixed part, which consists of the part of the component that must be used, and the variable part, which depends on the particular use. Our WBS component UI makes elements in UI as the fixed part which is implemented by the view, while presentation and structure of the elements as the variable part. The structure defines the elements' place and their relationship (i.e. parent-child), while the presentation defines the elements' presentation information such as size and color. A WBS component has default structure and presentation configurations. When developers reuse the WBS component, they can adjust configurations to adapt UI with particular scenario.

The Fig.5 give the City Scenic-Spots List component UI elements, default structure and presentation configurations. The UI elements are described in JSON (JavaScript Object Notation) format, which is widely used in web-based environment. The structure and presentation configurations are described by related web standards, XHTML and CSS. This component has two UI elements, a city selection and a list. And the default structure configuration puts the selection and list into two "div" markups. The presentation configuration describes how to display these elements.

```
/* City Scenic-Spots List Uesr Interface
{
  elements: [
    {name : 'cityselection'},
    {name : 'spotslist'}
  ]
}
```

```
<!-- City Scenic-Spots List Structure Configuration-->
<div class='comp_container'>
  <element component-id='list1' name='cityselection' />
</div>
<span class='clear_element' />
<div class='comp_container'>
  <element component-id='list1' name='spotlist' />
</div>
```

```
/* City Scenic-Spots List Presentation Configuration*/
div.comp_container {
  background:#fff;
  float:left;
  border:1px solid rgb(137, 213, 239);
  padding:5px;
  margin:5px
}
span.clear_element {
  clear:both;
}
```

Fig.5 User Interface and Configurations

图 5 界面与配置文件

It should be noted that, another problem is how to adapt the interaction logic while keeping the component correct when some elements are changed or removed. According to our WBS component model, UI elements can invoke proper business logic and respond to return results through a specific controller no matter where it is placed and how it is presented since the whole controller is divided into element-combined controllers and the interaction logic is divided to element granularity. Moreover, if a UI element has been removed, the specific controller will be removed automatically while not affect the other UI elements and controllers as well.

4.3 Component Composition

We define two types of connectors to enable components to interact with others' functionalities through programming interfaces, as shown in Fig.6.

A) Simple Connector *B) Data Connector*

Fig.6 Component connectors

图 6 构件连接子

Simple connector enables two components to interact directly. This connector supports a component to connect its method with another component's event. Then this component can catch the event triggered by the other component and invoke the corresponding method.

However, data of multiple components may be in different formats, so that it prevents components from being connected directly. Data connector can handle data format mismatch. Data connectors work with the built-in data wrappers which handle specific kinds of data transformation, e.g. content-filter. Connectors also allow developers to program a function for enhancing their functionalities, in case that the logic of data transformation is too complex to be handled by built-in data transformers

On the other hand, UI composition is relatively easy since our WBS component model gives flexible UI adaptation capability. The UI elements of WBS components will be merged into a new UI when developers assemble them. Then developers need to adjust the structure and presentation configuration of this new UI. The new UI's configuration is the same as the atomic component UI's.

The structure configuration of "city tourist guide" is shown in Fig.7. In this scenario, the input and button of the Image Searcher, along with the display list of the City Scenic-spots List, are removed from the configuration. Moreover, since the image is displayed in the location marker on the map, the xml element corresponding to the Image Searcher is nested in the xml element of the Map in the configuration.

```
<!-- City Tourist Guide Structure Configuration-->
<div class='comp_container'>
  <element component-id='list1' name='cityselection' />
</div>
<span class='clear_element' />
<div class='comp_container'>
  <element component-id='map1' name='map'>
    <element component-id='imagesearcher1' name='image' />
  </element>
</div>
```

Fig.7 UI Composition Structure Configuration

图 7 复合界面结构配置文件

4.4 Composite Component

When developers assemble the WBS components, they just work against the component interfaces and do not need to care about the implementations. So making a composition result as a composite

component means developers should define interface of the composition result. The composite component is shown in Fig.8.

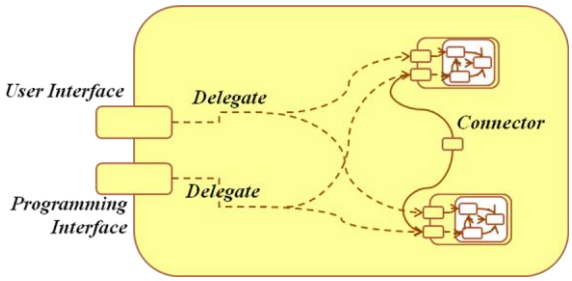


Fig.8 Composite Component

图 8 复合构件

Just like the atomic component, the composite component interface consists of programming interface and UI. Developers need to define properties, methods and events in programming interface and how they are delegated to subcomponents. The composition environment is responsible for binding the delegations. On the other hand, UI elements of a composite component consist of the UI of subcomponents. Moreover, developers can give default structure and presentation configuration to composite component.

4.5 On-the-fly Support

In most web-based service composition environments, the instantiation of a component is done either design-time or run-time. A design-time component is a fake component without any real functionality. When developers want to see the execution result of the application, they have to deploy the application in the specific environment and debug for the desired result.

Our WBS component model does not distinguish between design-time and run-time, while our composition environment does not follow the design-run-debug cycle. The composition environment is hosted in a web browser and implemented by JavaScript, which is a dynamic script language running in web browser. Hence, the environment can load WBS component definitions without restarting. Also because a WBS component is implemented by JavaScript, a component can be instantiated without compilation and deployment once its definition is retrieved into composition environment. After being instantiated, a component is at runtime. It connects with a running service and can respond to user actions with full functionalities. Since every component is always at runtime, components can be connected just-in-time when developers assemble them. Service invocations may be caused by this connection and these invocations may modify component internal states and UI. That makes be developers able to validate the composition result on-the-fly.

5 Related Work

The web-based service composition has involved a lot related work. In [3][4][5], some fundamental work about web-based service component were discussed, including the basic WBS

component model and event-based composition model, providing a proof of concept implementation. However, the component model of the work is not suitable for the UI reusability. Its UI is fixed and hard to be customized. It does not support composite component yet. Furthermore, this component model cannot be used in on-the-fly composition since it supports integration of heterogeneous components, such as ActiveX and Java Applet, which separate the design-time and runtime strictly.

There are many form-based web component models and related frameworks [9][10] that facilitate building composite web applications. However, these components are form-based. That means any interaction with these components will cause request and respond to backward server. Therefore it does not support on-the-fly service composition.

Thousands of mashup applications [7] already exist on the web, which allow users to create web applications by using widgets. However, in these mashups, UI is described by fixed markup files, which are encapsulated in widgets and cannot adapt to particular scenario. Furthermore, current mashup tools do not support on-the-fly composition yet.

Java Portal [8] lets users customize composite pages with full-fledged, pluggable components called portlets. However UI is always considered as a fixed part in portlets and hard to be reused. Moreover, Portal explicitly distinguishes between UI components (portlets) and composite results (portals), it's hard to support composite component, which means turning a portal to a portlet.

In addition, some work [12][13] and tools (i.e. Microsoft Visual Basic) in CBSD field allow developers to create applications with visual components. These components encapsulate UI and functionalities. However, the functionalities of these components do not connect with services but are implemented standalone. Therefore, these components should be compiled and deployed before execution, which makes on-the-fly composition impossible.

6 Conclusion and Future Work

The web-based service composition is becoming a popular composition style in Service Oriented Computing. This paper makes the following contributions. Firstly, we present a well-structured component model that decouples the UI and service business logic for better adaptation. Secondly, developers are able to assemble components not only at business logic level but also at UI level. And our component model supports on-the-fly composition, which provides a quick and effective feedback way for qualification. Thirdly, the component model supports the composition result to be published as a new component for further reuse.

There are some open issues for our component model. First, our WBS components encapsulate service and UI. Although there are many services on the web, it is not an easy task to create WBS components from these services. Hence, a WBS component builder, which helps developers to create WBS components, is under development and will be open source soon. Second, to adapt WBS components, developers should write XML configuration by hand now. It makes component adaptation difficultly. However, the next version of our composition environment will have a more powerful configuration panel, which allows developers to adapt components in a visual way and creates XML configuration automatically. Finally, our current model can only support simple composition logic by providing simple connector and data connector. We are now progressing on much more complex service composition style, such as business process driven. A new connector

type will be provided in our next version.

References

[1] H Mili, A Mili, S Yacoub, E Addy. Reuse-based software engineering: techniques, organization, and controls. Wiley-Interscience New York, NY, USA, 2001.

[2] Hafedh Mili, Fatma Mili, and Ali Mili. Reusing Software: Issues and Research Directions. IEEE Transactions on Software Engineering, Vol 21, no. 6, June 1995.

[3] J. Yu et al. A Framework for Rapid Integration of Presentation Components. In the Proceedings of WWW'07, Banff, Canada, May 2007.

[4] J. Yu et al. Mixup: a Development and Runtime Environment for Integration at the Presentation Layer. In the Proceedings of International Conference on Web Engineering 2007, Como, Italy, July 2007.

[5] F Daniel, J Yu, B Benatallah, F Casati, M Matera. Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. IEEE Internet Computing, 2007.

[6] Oscar Diaz, Salvador Trujillo, Sandy Perez. Turning Portlets into Services: The Consumer Profile. Presentation Components. In the Proceedings of WWW'07, Banff, Canada, May 2007.

[7] Merrill, D. Mashups: The new breed of Web app. <ibm.com/developerworks/library/x-mashups.html>

[8] Java Portlet Specification. <http://jcp.org/aboutJava/communityprocess/final/jsr168>.

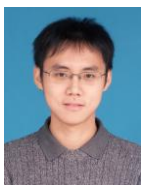
[9] Richard Cardone, Danny Soroker, Alpana Tiwari. Using XForms to Simplify Web Programming. In the Proceedings of WWW 2005, May 2005, Chiba, Japan.

[10] Ito, K. and Tanaka, Y. A visual environment for dynamicweb application composition. HT'03

[11] Girish Chafle, Gautam Das, Koustuv Dasgupta, Arun Kumar, Sumit Mittal, Sougata Mukherjea, Biplav Srivastava. An Integrated Development Environment for Web Service Composition. In the Proceedings of IEEE International Conference on Web Services 2007.

[12] Padmal Vitharana. Risks and challenges of component-based software development. Aug 2003, Communications of the ACM

[13] AI Morch, G Stevens, M Won, M Klann, Y Dittrich, Volker Wulf. Component-based technologies for end-user development. Sep. 2004, Communications of the ACM.



Zhao Qi was born in 1984. He is now a PhD student in the School of Electronics Engineering and Computer Science of Peking University. His research directions include service-oriented computing and software engineering.

赵祺(1984－),男,北京人,目前于北京大学信息科学技术学院攻读博士学位 ,主要研究领域为面向服务计算, 软件工程。

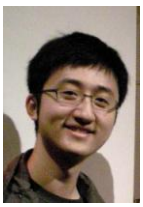
Gang Huang was born in 1975. He received the PhD/MSc degree in computer science from Peking University in 2003 and Northwestern Polytechnical University in 2000 respectively. He is an Associate Professor in the School of Electronics Engineering and

Computer Science of Peking University. His research interests are in the area of distributed computing and software engineering.

黄罡（1975 年-）,男,湖南常德人,博士,北京大学信息科学技术学院副教授,主要研究领域为分布计算与软件工程。

Xuanzhe Liu is now a Ph.D candidate in the School of Electronics Engineering and Computer Science,Peking University. His research interests include Software Engineering and Service Oriented Computing.

刘譞哲（1980-），博士生，甘肃兰州人，主要研究领域为软件工程、面向服务计算等。



HUANG Jiyu was born in 1985. He is now a master student in the School of Electronics Engineering and Computer Science of Peking University. His research directions include service-oriented computing and software engineering.

黄冀渝(1985-),男,成都人,目前于北京大学信息科学技术学院软件工程研究所攻读硕士学位,主要研究领域为面向服务计算,软件工程。