JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 25, 1427-1445 (2009)

BPEL Extensions to User-Interactive Service Delivery*

JONATHAN LEE, YING-YAN LIN, SHANG-PIN MA AND SHIN-JIE LEE
Department of Computer Science and Information Engineering
National Central University
Chungli, 320 Taiwan

Web service technologies are best exploited by composing services, and BPEL (Web Services Business Process Execution Language) is adopted industrial-wide as the *de facto* service composition standard. However, a BPEL composite service is typically treated as a fully automated service flow that orchestrates multiple web services and involves no user interactions – a desirable feature for service delivery, and is presently not included in the BPEL standard. In this work, we propose an extension to BPEL to infuse user interactions into composite services along three dimensions: (1) to develop two BPEL extension activities to describe the inner workings of user interactions in BPEL service and the rendering of service user interfaces; (2) to provide a wizard-style mechanism to guide the user to interact with the service flow in accordance with the sequence of service execution; and (3) to devise a UI service communication protocol to facilitate secure cross-domain communication among UI services from various domains. An enhanced BPEL engine with a service UI rendering engine has been accordingly developed.

Keywords: service composition, user interaction, BPEL, service delivery, user interface

1. INTRODUCTION

Service-orientated architecture (SOA) has become a main trend in software engineering, motivating the construction of software applications based on the notion of services. As noted by Maamar and his colleagues [18] that composing multiple web services is more beneficial to users than a single service. Several languages for web service composition have emerged, for example, WSFL [8], WS-BPEL [1], WSCI [2] and *etc*.. WS-BPEL (Web Services Business Process Execution Language, also known as BPEL) combines the best of other standards for web service composition and allows for a mixture of block-structured and graph-structured process models [31]; and consequently, BPEL has been adopted industrial-wide as the *de facto* service composition standard [5, 14, 15].

A BPEL composite service is typically treated as a fully automated service flow that orchestrates activities of multiple software components exposed as web services and involves no user interactions – a desirable feature for service delivery advocated by [12, 28], but is presently not included in the BPEL standard.

Several industrial research groups, as a result, have developed human work-flow services that leverage the BPEL support for user interaction by means of asynchronous services [11]. BPEL4People [6] was developed atop the BPEL specifications for the orchestration of role-based human activities. In BPEL4People, a people activity is defined as a BPEL activity to specify user interactions and to associate with human tasks that

Received November 17, 2008; accepted December 4, 2008.
Communicated by Jonathan Lee, Wei-Tek Tsai and Yau-Hwang Kuo.
* This work was supported by the National Science Council of Taiwan, R.O.C. under grant No. NSC 97-2631-H-008-001.

users perform. The Oracle BPEL Process Manager (OPRM) [24] integrates standard BPEL functionality with workflow services that are deployed on a supported application server. The workflow service requires a powerful set of programming and WSDL interfaces in order to build UI workflow interfaces. Neither BEPL4Peole nor OPRM does specify explicitly how to render or integrate user interfaces; and therefore, the feature of user interfaces can only be supported through external client applications.

Missing supports with respect to user interactions in BPEL can be summarized as follows:

- BPEL developers have to program specific service user interfaces for each BPEL service. BPEL does not state how to deliver services with user interfaces to enable end users to consume services.
- BPEL can neither guide process participants on when and how to join the service flow, nor allow a participant to provide data during the execution of composite services.
- SOA-based applications cannot render multiple user interfaces, including newly developed service user interfaces and existing legacy web user interfaces, in a uniform work-space.

As a continuation of our previous work on a discovery-based service composition framework to enhance availability of composite services [16, 17], and further inspired by the concept of human involvement advocated in BPEL4People [6], in this work, we propose an extension to BPEL to infuse user interactions into composite services along the following three dimensions:

- to develop two BPEL extension activities: interactPeople and renderUI, to describe the inner workings of user interactions in BPEL service and the rendering of service user interfaces that are described by an extension to XUL (called EXUL);
- to provide a wizard-style mechanism to guide the user to interact with the service flow in accordance with the sequence of service execution; and
- to devise a UI service communication protocol to facilitate secure cross-domain communication among service user interfaces from various domains.

By means of these extensions, a deployed composite service can be consumed directly by end users via an integrated service workspace.

This paper is organized as follows. Section 2 outlines several extant research work related to the proposed approach. Section 3 analyzes the requirements of integrating services and user interactions. Section 4 describes in detail the proposed extensions of XUL and BPEL. Section 5 fully discusses a secure cross-domain UI service communication protocol. Section 6 explains the implementation of our enhanced BPEL engine. The final section draws conclusions and suggests future work.

2. RELATED WORK

In what follows we outline some related work with respect to service composition and user interactions in service-oriented architecture.

2.1 XML-based User Interface Description Languages

Several currently available XML-based user interface description languages are standardized, including XAML [19], UIML [23, 30], XForms [32] and XUL [20].

XAML [19] stands for Extensible Application Markup Language, which is a user interface description language that was created by Microsoft, and is mostly used in the WPF (Windows Presentation Foundation) platform. XAML provides developers with a set of UI elements, support of flow control, event handling, data bindings and other facilities to describe relationships among the XAML elements and their corresponding runtime objects. Developers can use XAML elements to build a tree structure of .NET objects (Common Language Runtime objects) with specific properties and logic. The drawback of XAML is that it is not a platform-portable user interface description language like HTML since it is required to be processed under the .NET framework.

UIML [23, 30] stands for OASIS User Interface Markup Language, which is a meta-language and standardized by OASIS, and is to provide a standard representation of any user interface which can be mapped to a variety of existing languages, such as HTML, XHTML, Java, C++ and others. UIML can be implemented by anyone and it works on multiple platforms and multiple devices. UIML is not suitable for this work because UIML is provided as an abstract meta-language, and therefore not friendly to UI developers.

XForms [32], which was standardized by W3C, is an XML application markup language which can be integrated with numerous XML-based markup languages, such as XHTML and SVG. XForms provides many form-based elements to support interactive web applications, including the form model, submission control, instance data manipulation, strong data typing, expressions evaluation, event handling and action control. However, XForms does not support other widely used widgets that developers could use to enrich the representation of their service UIs.

XUL [4, 20] stands for XML User Interface Language, which is an XML-based user interface markup language created by Mozilla, and is designed to be a widget-based user interface description language to reduce effort of developing web applications. XUL relies on multiple existing web standards and technologies, such as XHTML, DOM, JavaScript, CSS, and AJAX. With XUL, the user interfaces of web applications can be implemented and modified more easily by UI developers.

A summary of the comparisons of aforementioned XML-based user interface description languages [20, 25, 27, 34] is depicted in Table 1. XUL provides rich set of widget elements and is friendly to HTML developers. XUL not only has platform portability, like HTML, but also separates presentation from application logic. However, it is limited in the sense that it works only with the Mozilla-compatible browser, such as Firefox. To solve this problem, the ZK Framework [26] was adopted herein the design of the service UI rendering engine due to the most of overlapping between XUL and ZUML (ZK User Interface Markup Language) [26] that is mainly supported by ZK, to make it possible to use XUL with our extensions in multiple browsers. Since XUL has aforementioned features and benefits, it was used herein as the service user interface markup language.

Table 1. Comparison of XML-based user interface markup languages.

Languages	Rich set of widgets	Friendly to HTML developers	Platform portability	Browser compatibility	Visual authoring support
XUL Mozilla, 1998	Yes	Yes	Yes	Mozilla Firefox	Text-based editor
XAML Microsoft, 2006	Yes	Yes	Windows platform, .Net	WinFX/Sliveligh t plugin is required	Visual Studio
UIML OASIS, 1997	UIML provides abstract creation of UI	No	Yes	Depends on the runtime implementations	Text-based editor
XForms W3C, 2003	Form-based elements only	Yes	Yes	Depends on the runtime implementations	Visual XForms Designer, Text-based editor

2.2 BPEL4People and Oracle BPEL Process Manager

Several research groups have developed human workflow services that leverage the BPEL support for user interaction by means of asynchronous services [11]. People and tasks become another asynchronous service in the BPEL process. BPEL4People [6] was published as an OASIS standard in June 2007. BPEL4People provides the BPEL extensions to address the human involvement in BPEL standard. The major BPEL extensions in BPEL4People are people activities and role-based human tasks. A people activity is defined as a BPEL activity to specify user interactions and to associate with human tasks that users perform.

The Oracle BPEL Process Manager (OPRM) [24], integrates standard BPEL functionality with workflow services that are deployed on a supported application server. The workflow service includes a set of programming APIs and WSDL interfaces for constructing UI workflow interfaces. These interfaces enable the use of current UI approaches, including JSF, AJAX, .NET, Adobe Flex and others.

Neither BEPL4Peole nor OPRM does specify explicitly how to render or integrate user interfaces; and therefore, the feature of user interfaces can only be supported through external client applications. In contrast to BEPL4Peole and OPRM, our proposed approach explicitly defines how to integrate and render user interfaces by extending the BPEL specification.

2.3 WSRP (Web Services for Remote Portals)

The WSRP [22] defines a web service interface that allows for interacting with presentation-oriented web services. The two main usages of WSRP are (1) to provide portlets as presentation-oriented web services that can be used by portal servers; and (2) to consume presentation-oriented web services provided by portal or non-portal content providers and to integrate them to produce a portlet-based application.

The current version of WSRP is WSRP v2.0, which has been approved by OASIS in April, 2008. WSRP concentrates on the adaptation of user interfaces into services, but does not address integration of user interfaces and service components, on which our work will focus.

3. ANALYSIS OF REQUIREMENTS TO INTEGRATE SERVICES AND USER INTERACTIONS

To address the aforementioned missing supports with respect to user interactions in BPEL, we analyze and identify requirements as an attempt to infuse user interaction into composite services. Furthermore, BPEL extensions to describe the inner workings of user interactions in BPEL service as well as a service UI communication protocol are proposed to fulfill these requirements, and an enhanced BPEL engine with a UI rendering engine is then developed to support these BPEL extensions.

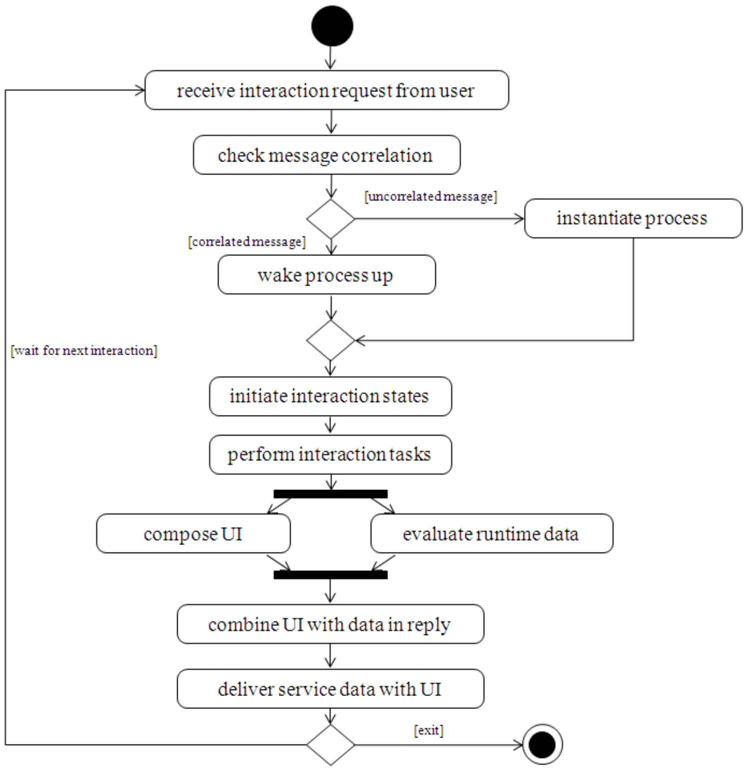
3.1 System-Level Requirements

Based on the characteristics of BPEL, the following six requirements are analyzed and defined to support the missing components that are described in section 1, including:

(1) Standard-compatibility. The solution should be compatible with industrial standards – BPEL, WSDL or other specifications that are adopted industrial-wide.
(2) Newly-defined BPEL activity. New BPEL activities should be defined to infuse user interactions into BPEL composite services.
(3) Dynamic UI generation. The solution should support the dynamic generation of a user interfaces based on variable data that are generated during the execution of service flow.
(4) Separation of UI and business logic. Based on the specific needs of users, the template of the user interfaces can be changed, for example, a simple form-based UI can be replaced by a complex one.
(5) Wizard-style mechanism. The solution should guide the user to interact with the service flow in accordance with the sequence of service execution.
(6) Cross-domain UI communication. The solution should provide a communication protocol to facilitate secure cross-domain communication among service user interfaces from various domains, and prevent scripting attack problem which is deemed as key security issue in the composition of service user interfaces.

3.2 User Interaction Model in SOA

To bridge the above requirements and the design of BPEL extensions and further implementation, a user interaction model is devised as the basis for the proposed approach (see Fig. 1).


receive interaction request from user
check message correlation
[uncorrelated message]
instantiate process
[correlated message]
wake process up
[wait for next interaction]
initiate interaction states
perform interaction tasks
compose UI
evaluate runtime data
combine UI with data in reply
deliver service data with UI
[exit]
Fig. 1. User interaction model.

Firstly, a service client, which is a human user, communicates a designated service flow by operating a service user interface rendered in a browser with the ability to send an interaction request to begin an interactive service flow. Secondly, BPEL engine receives the interaction request from the user, and then, the message correlation sets [1] are checked to enable *message-to-process-instance* routing. If the interaction request is an uncorrelated message, which means there has no one process instance holds conversations with this service client, the BPEL engine will instantiate a new process instance to take over this interaction request. Otherwise, the correlated interaction request will be dispatched to the process instance located. After a process instance obtains the interaction request, it initiates interaction states for the execution of further tasks. Finally, the BPEL engine combines the user interface, *e.g.* the service user interface, with runtime service data in reply and delivers the mixed outcome to the user. In the meanwhile, the legacy web user interfaces could also be delivered to the user in accompany with the service user interfaces. In addition, the user interaction flow could be repeated many times to perform a complex task that requires multiple user interactions.

4. BPEL EXTENSION TO INTEGRATE USER INTERACTIONS

Our extensions to XUL and BPEL support user interactions in SOA-based applications with the following features: (1) combining XUL user interface markups with run-time service data, called service user interfaces, by which the users can interact with composite services in run time, called EXUL; and (2) facilitating the rendering of the service user interfaces and the handling of user interactions. The following sub-sections will describe in detail the proposed extensions of XUL and BPEL, respectively.

4.1 XUL Extensions

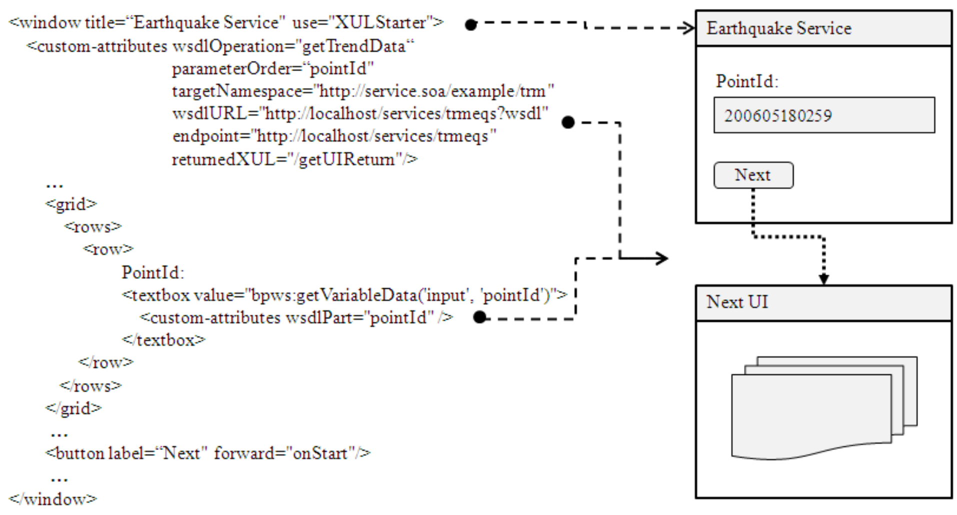
XUL [20] provides a rich set of widget components, container elements, layout models, event handling and scripting support for describing user interfaces of applications. To combine user interface markups with runtime service data during the execution of composite service, XUL is extended in this work to assign XPath query expressions and BPEL-defined functions as values to XUL attributes. These expressions and functions, assigned in the design time, will be automatically evaluated by runtime BPEL variable data.

A wizard-style mechanism is provided to guide the users to interact with the service flow in accordance with the sequence of service execution. This work aims to elucidate an approach to enabling the users to navigate the service flow by clicking a "Next" button, similar to an application wizard, with which service developers do not need to consider the flow control in the client-side applications.

To enable users to automatically navigate to the next step of a service flow, a service invocation to the next operation of the service flow is needed. Therefore, the binding information associated with the invocation to a service operation is a desired feature to be included in XUL in each delivery of service user interface. Another extension to XUL to specify the binding data in XUL is proposed in this work by defining specific extension markups.

To provide these specific extension markups, we adopt the custom-attributes declaration, advocated by ZK [26], for users to get involved in a service flow. The custom-attributes declaration is a convenient way to assign attributes to associated XUL elements in a direct manner without programming. Two kinds of custom-attributes are thus defined: (1) binding-related custom attributes that are mapped to the binding information in WSDL, including the URL and the target namespace of WSDL, the service location, the WSDL operation to be invoked, and the expression of message part targeted as XUL output; and (2) a set of wsdlPart custom-attributes that are associated with the <part> elements of the input message in WSDL. Clearly, the binding-related custom attributes are used to invoke a (composite) service. A wsdlPart custom attribute is used to designate which XUL element is associated with the value of the input parameter while a (composite) service is invoked.

Fig. 2 depicts an example of EXUL. The left side of this figure shows the code snippets of the EXUL, while the right side shows the possible result rendered from this EXUL: the "Earthquake Service" window component is delivered to the client with the runtime service data of "PointId" that is evaluated by BPEL function bpws:getVariable Data ('input', 'pointId') and treated as the default value. After the client obtains the rendered service user interface, a "Next" button enables him/her to navigate to the next step of the service flow by processing the specific custom-attributes.



```
<window title="Earthquake Service" use="XULStarter">
  <custom-attributes wsdlOperation="getTrendData"
                     parameterOrder="pointId"
                     targetNamespace="http://service.soa/example/trm"
                     wsdlURL="http://localhost/services/trmeqs?wsdl"
                     endpoint="http://localhost/services/trmeqs"
                     returnedXUL="/getUIReturn"/>
    ...
    <grid>
      <rows>
        <row>
            PointId:
            <textbox value="bpws:getVariableData('input', 'pointId')">
              <custom-attributes wsdlPart="pointId" />
            </textbox>
        </row>
      </rows>
    </grid>
    ...
    <button label="Next" forward="onStart"/>
    ...
</window>
```

Fig. 2. Example of EXUL.

4.2 BPEL Extensions

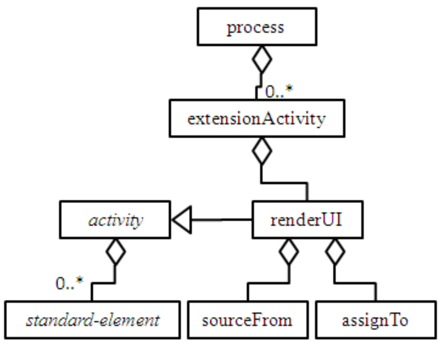
Our solution to providing service composition flow with user interactions is supported by two key elements: (1) the rendering of the service user interfaces; and (2) an interaction mechanism between a long-running service flow and its participants. Both elements are built based on the definition of extension activity in the BPEL specifications [1].

4.2.1 RenderUI activity

The renderUI activity is a BPEL extension activity that facilitates rendering and composition of service user interfaces described by EXUL. The renderUI activity allows the service flow to be associated with an EXUL document, and triggers BPEL engine to update the content of EXUL with the runtime service data during the execution of composite service. To establish service delivery with UI, a service UI rendering engine is required to translate the content of EXUL to pure HTML with scripts and style sheets and to deliver these user interfaces to the client.

The structure of renderUI activity is described in Fig. 3: a renderUI activity is composed of a single sourceFrom and the corresponding assignTo elements. The sourceFrom element specifies the source of EXUL: a standalone EXUL document (standalone mode association) can be referred to or the content of EXUL can be written into the content of sourceFrom element (inline mode association). The assignTo element allows BPEL engine to assign the processed content of XUL to a part attribute within a variable or selection of variable part for the upcoming delivery of service user interfaces. A target destination of assignTo is defined by using XML Schema string type, *i.e.* xsd:string.

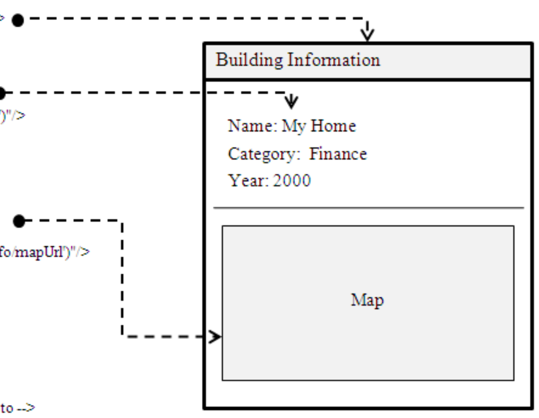
As shown in Fig. 4, the left side of the figure shows the code snippets of an example of renderUI activity, and the right side shows the possible result: a building information table, which is described by EXUL, is written in the renderUI activity. EXUL consists of the original XUL elements and the BPEL extension functions.



```
<process sdml:xmlns="urn:sdml4bpel">
  ...
  <extensionActivity>
    <sdml:renderUI standard-attributes>
      standard-elements
      (
        <sdml:sourceFrom location="anyURI"? refId="SDML-element-Id"?/>
        |
        <sdml:sourceFrom>
          EXUL-window-elements
        </sdml:sourceFrom>
      )
      (
        <sdml:assignTo variable="BPELVariableName" part="NCName">
          <sdml:query queryLanguage="anyURI"?>?
            queryContent
          </sdml:query>
        </sdml:assignTo>
      )
    </sdml:renderUI>
  </extensionActivity>
  ...
</process>
```

Fig. 3. Structure of renderUI activity.

```
<extensionActivity>
  <sdml:renderUI name="render-building-info" >
    <sdml:sourceFrom>
      <!-- EXUL content -->
      <window id="building_window"  title="Building Information">
        <grid width="820px">
        ...
        <label value="bpws:getVariableData('input',
                                           'buildingInfo',
                                           '/BuildingInfo/name')"/>
        ...
        </grid>
        <groupbox id="tryView">
        ...
        <image src="bpws:getVariableData('input',
                                         'buildingInfo',
                                         '/BuildingInfo/mapInfo/mapUrl')"/>
         ...
        </groupbox>
      </window>
      <!-- EXUL content -->
    </sdml:sourceFrom>

    <sdml:assignTo variable="output" part="return">
      <!-- specify where the processed XUL content will be assigned to -->
      <sdml:query>/ReturnInfo/serviceUI</sdml:query>
    </sdml:assignTo>
  </sdml:renderUI>
</extensionActivity>
```


Building Information

Name: My Home

Category: Finance

Year: 2000

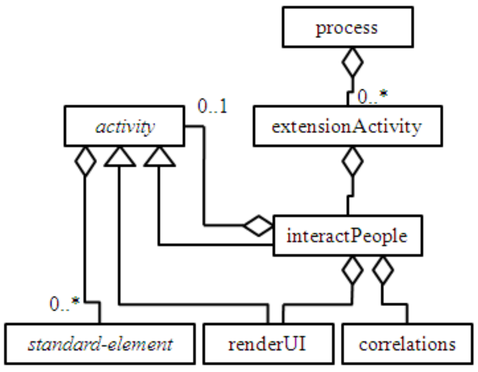
Fig. 4. Example of renderUI activity.

These extension functions, assigned in the design time, will be automatically evaluated by runtime BPEL variable data, for example, the expression function "bpws:get VariableData ('input', 'buildingInfo', '/BuildingInfo/name')" is used to extract the name of building that is stored in the part "buildingInfo" of the variable "input".

4.2.2 InteractPeople activity

The interactPeople activity is a BPEL extension activity that handles the interaction behavior between a service flow and its human participants. The interactPeople activity is designed as a receive-reply tuple. To facilitate user interaction, not only the interact-People activity is used to handle the interaction behavior, but also a predefined renderUI construct is contained in the interactPeople activity to describe the rendering and composition of service user interface.

The structure of interactPeople activity is described in Fig. 5: an interactPeople activity specifies the partner link from which a message is expected to receive, and the port type and operation with which a human client partner is expected to interact. Moreover, an input variable to cache the received message data can also be specified, along with an output variable that contains the service data with UI to be sent in reply. Additionally, interactPeople activity, similar to the receive activity plays a role in the lifecycle of a service flow: the way to instantiate a service flow in BPEL is to annotate an interactPeople activity with the createInstance attribute set to "yes" (refer to receive and pick activities in the BPEL specification for the variant [1].) The default value of this attribute is set to "no".

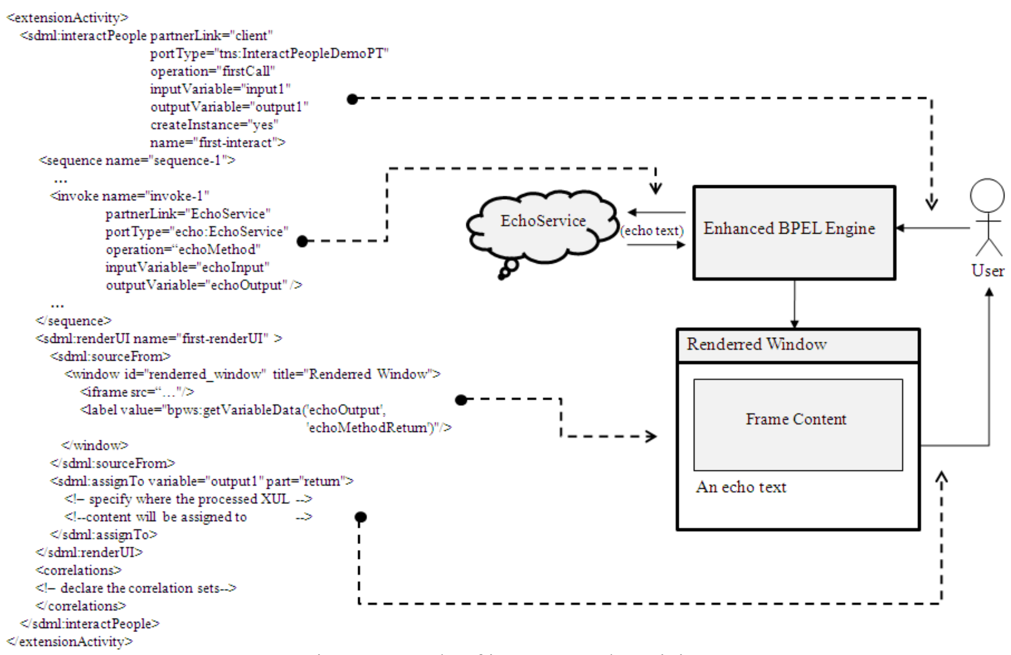


```
<process sdml:xmlns="urn:sdml4bpel">

  <extensionActivity>
    <sdml:interactPeople partnerLink="NCName"
                         portType="QName"
                         operation="NCName"
                         inputVariable="BPELVariableName"?
                         outputVariable="BPELVariableName"
                         createInstance="yes|no"?
                         standard-attributes>
      standard-elements
      BPEL-activity
      renderUI-spec
      <correlations>?
        <correlation set="NCName"  initiate="yes|join|no"? />+
      </correlations>
    </sdml:interactPeople>
  </extensionActivity>

  </process>
```

Fig. 5. Structure of interactPeople activity.

As mentioned above, an interactPeople activity must contain a renderUI construct to markup the service user interface delivered in an interaction. Before replying the output message with UI, the interactPeople activity allows tasks specified by the BPEL activities to be performed. If a series of interactPeople activities are correlated during the execution of service flow, then a correlations element is required to establish the correlated conversations.

Fig. 6 explains an example of the interactPeople activity. The left side of the figure shows the code snippets of the interactPeople activity, and the right side shows a possible scenario during the execution of the inteactPeople activity: a human client requires an echo interaction. The human client invokes the operation "firstCall", and then the service flow obtains the input message and dispatches the message to the interactPeople activity. Before a reply is initiated, the interactPeople activity triggers a service invocation to send a text to the "EchoService"; the same text is sent in reply. Finally, the contained renderUI activity combines the output service user interface with an embedded frame that is targeted to a remote URL and the echo text label for reply.



```
<extensionActivity>
  <sdml:interactPeople partnerLink="client"
                       portType="tns:InteractPeopleDemoPT"
                       operation="firstCall"
                       inputVariable="input1"
                       outputVariable="output1"
                       createInstance="yes"
                       name="first-interact">
    <sequence name="sequence-1">
      ...
      <invoke name="invoke-1"
              partnerLink="EchoService"
              portType="echo:EchoService"
              operation="echoMethod"
              inputVariable="echoInput"
              outputVariable="echoOutput" />
      ...
    </sequence>
    <sdml:renderUI name="first-renderUI" >
      <sdml:sourceFrom>
        <window id="renderred_window"  title="Renderred Window">
          <iframe src="..."/>
          <label value="bpws:getVariableData('echoOutput',
                                            'echoMethodReturn')"/>
        </window>
      </sdml:sourceFrom>
      <sdml:assignTo variable="output1" part="return">
        <!– specify where the processed XUL -->
        <!--content will be assigned to      -->
      </sdml:assignTo>
    </sdml:renderUI>
    <correlations>
    <!– declare the correlation sets-->
    </correlations>
  </sdml:interactPeople>
</extensionActivity>
```

Fig. 6. Example of interactPeople activity.

5. SECURE CROSS-DOMAIN UI SERVICE COMMUNICATION

Recently, due to the emergence of web 2.0, there is a tremendous demand for integrated service-oriented applications, in which services and their user interface components that may come from multiple web sites across various domains, are combined and delivered to a single workspace displayed on the browser; and therefore, it is vital that these service user interface components can communicate with each others to share resources and exchange messages in client-side workspace.

However, a strict restriction on the security policy has been imposed by most of the browsers to prevent the user interface components from cross-domain communication.

XUL, usually treated as the service user interface mash-ups, can be used to combine contents of service user interfaces from multiple web sites across various domains by adopting an inter-frame communication mechanism, namely its <iframe> elements. The main problem in adopting XUL for cross-domain communication is that there lacks a secure mechanism in XUL to enable the service user interface components to share resources and exchange messages.

In this section, we first describe the inter-frame communication and the problem inherited in XUL <iframe> elements to combine the contents of service user interface components. Secondly, we present the details of our service UI communication model. Finally, the secure cross-domain UI service communication protocol is accordingly proposed.

5.1 Inter-frame Communication

In web 2.0, a well-known technique – mash-up, which is a web application that combines content (data and codes) from multiple sites, is widely used to develop integrated web applications. The content, combined in a mash-up web application, is called a gadget. The component that is used to combine and coordinate the gadgets into one integrated web application is called the gadget integrator [3].

For example, consider a mash-up application of a pre-trip planning portal where the information from multiple web sites, including the user's airline booking, hotel reservations and bank service, can be combined and displayed on the browser. In this mash-up application, the user interface components that are employed to locate the content of airline booking service is a gadget. Usually, such a gadget is combined into a mash-up application using the HTML <iframe> element. The top-level frame in the browser, rendered from the entire HTML document, combines the gadget frames and acts as the gadget integrator.

In XUL service user interface mash-ups, the integrated service user interface, rendered from the XUL <window> element, is treated as the integrator. A service user interface component, combined using an XUL <iframe> element, is a gadget.

To develop service user interface mash-ups using XUL can be as simple as combining isolated gadgets, or as complex as combining gadgets to communicate with each other. In such a complex situation, an integrator communicates with its gadgets from various web sites with a highly likelihood of encountering un-trusted contents from malicious service providers; and vice verse.

Another problem that impedes the cross-domain communication is the restrictions imposed by the browser's same-origin policy [21] to protect client-side resources from being attacked. The same-origin policy allows the browser to prohibit a document or script loaded from one domain to access the client-side resources from another domain [21]. Without such policy, the browser would not be able to prevent attacks from a malicious web site.

5.2 Service User Interface Communication Model

A secure cross-domain user interface service communication model is an inter-frame communication model devised to secure service user interface mash-ups. It is meant to enable gadgets to communicate with the integrator and with other gadgets by means of a well-defined communication protocol. It is also intended to support a cross-domain communication through secure messaging.

Figs. 7 and 8 present the proposed secure cross-domain communication model. Fig. 7 depicts the gadget-to-integrator communication, and meanwhile Fig. 8 shows the integrator-to-gadget communication, respectively.

In Fig. 7, if a gadget attempts to send a message to the integrator, it transmits at once a formatted message and a key that are sent from the integrator upon a page is loaded. The key is a random number, generated automatically by the integrator, to be used in a single run of communication. With the key, the integrator can verify whether the formatted message has been transmitted from a trusted gadget sender or not. If the key is verified, then the integrator accepts the formatted message and processes the message. Otherwise, the integrator ignores the message.

In Fig. 8, if the integrator intends to send a formatted message to a gadget, then it sends the message with its URL or the "origin" defined in the aforementioned same-origin

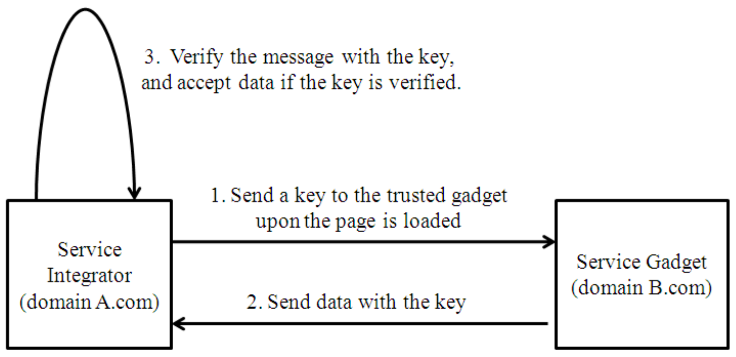

3. Verify the message with the key, and accept data if the key is verified.
1. Send a key to the trusted gadget upon the page is loaded
2. Send data with the key
Service Integrator (domain A.com)
Service Gadget (domain B.com)

Fig. 7. Gadget-to-integrator communication.

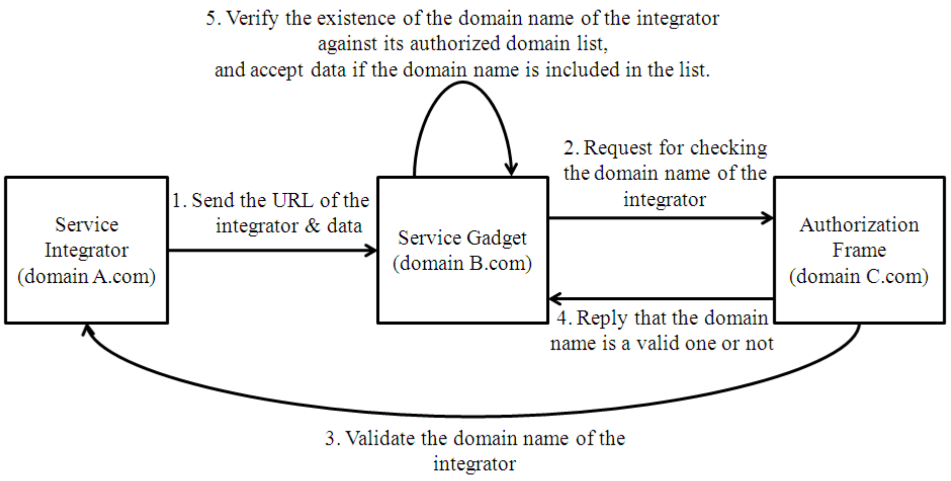

5. Verify the existence of the domain name of the integrator against its authorized domain list, and accept data if the domain name is included in the list.
Service Integrator (domain A.com)
1. Send the URL of the integrator & data
Service Gadget (domain B.com)
2. Request for checking the domain name of the integrator
Authorization Frame (domain C.com)
4. Reply that the domain name is a valid one or not
3. Validate the domain name of the integrator

Fig. 8. Integrator-to-gadget communication.

policy. This URL is then used to check the validity of the integrator. Once the gadget receives the URL, it will trigger its authorization frame to send requests to "domain-authorization.htm" web page to validate the domain name of the integrator. If the URL of the integrator is a valid one, then the authorization frame returns the gadget a "passed" signal in reply. Finally, the gadget verifies the existence of the domain name of the integrator against its authorized domain list. If the domain name is included in the authorized list, then the message sent from the integrator will be processed. Otherwise, the integrator is treated as being not permitted to send any data to the gadget.

The gadget-to-gadget communication can be achieved by integrating both communication models: gadget-to-integrator and integrator-to-gadget models, that is, to treat the integrator as a coordinator to dispatch messages among gadgets.

5.3 Secure Cross-Domain User Interface Service Communication Protocol

To implement the two proposed secure cross-domain user interface service communication models, a technique called fragment identifier messaging (FIM) is adopted to facilitate secure inter-frame communication [7, 9, 13, 29]. FIM motivates the need for secure inter-frame communication by exploiting the principle of browser's frame navigation to send messages among frames.

Based on the concept of FIM, the formats of request and response messages are accordingly defined to embody a secure cross-domain user interface service communication protocol to enable gadgets to communicate with the integrator and other gadgets through secure messaging. JSON [10] is used to encode request and response messages in executable Javascript format.

- The request message is defined as:

  {*t*: 1, *d*: {*m*: *METHOD*, *a*: [*ARGUMENTS*]}}

  t denotes the type of message. In the case that the value of *t* equals to 1, the message type is a request message; meanwhile, in the case the value of *t* equals to 2, it is a response message type. *d* denotes the message data that consists of a Javascript function *m* and a set of arguments *a* separated by comma, by which the sender can execute the function *m* with the set of arguments *a* in the receiver frame.
- The response message is defined as:

  {*t*: 2, *r*: {*c*: *NUMBER*, *m*: *METHOD*, *v*: *VALUE*}}

  t denotes the type of message. In the case that the value of *t* equals to 1, the message type is a request message; meanwhile, in the case the value of *t* equals to 2, it is a response message type. *r* denotes the returned message data, in which the receiver returns the status code *c* that is referred to the definition of the HTML status code [33], the executed Javascript function *m*, and the returned value *v*.

6. IMPLEMENTATION

In summary, there are three main features in our proposed approach to infusing user interactions into service composition:

- extensions to BPEL activity to specify the inner workings of user interactions in BPEL service as well as the rendering of service user interfaces that are described by EXUL;
- a wizard-style mechanism to guide the users to interact with the service flow in accordance with the sequence of service execution; and
- a secure UI service communication protocol to facilitate secure cross-domain communication among service user interfaces from various domains.

To implement these features, we developed an enhanced BPEL engine in which the handling of user interactions and the delivery of service with user interfaces could be infused into BPEL composite services.

Fig. 9 illustrates the high-level architecture of the enhanced BPEL engine, in which the BPEL engine is enhanced to equip with the capability of parsing and processing

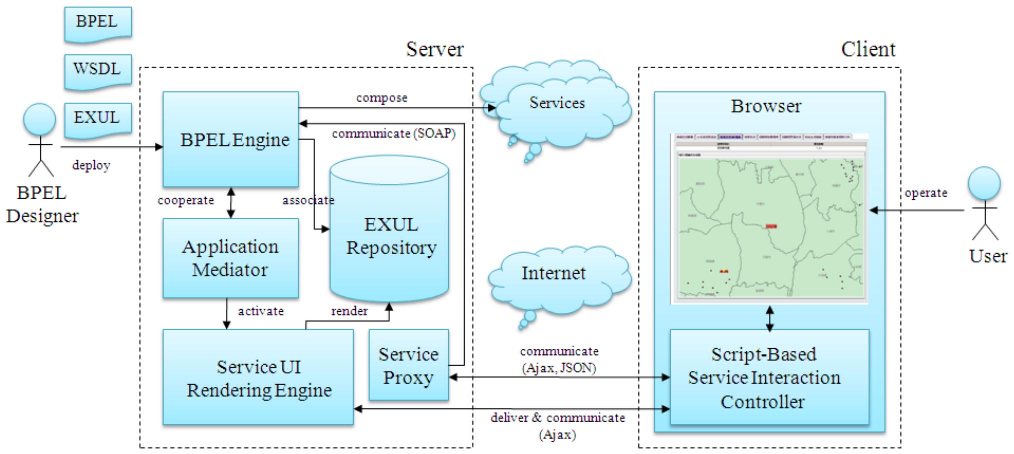


Fig. 9. Enhanced BPEL engine.

BPEL documents with extension markups in order to facilitate the rendering of the service user interfaces and the handling of user interactions during the execution of service composition flow.

To render the service user interfaces and to cooperate with BPEL engine to handle the user interaction control between a service flow and its clients, the service user interface rendering engine is accordingly developed.

In the service user interface rendering engine, we adopt the ZK framework to facilitate the rendering of service user interfaces, and to develop XULStarter that is deployed within ZK to handle user interaction control and service flow navigation. It translates the XUL description into pure HTML with Javascripts and style sheets, as well as communicates with the script-based service interaction controller delivered in client side browser to process events in an interaction.

As Fig. 2 shows, to implement the wizard-style mechanism in service flow, a navigation control is activated, after the "Next" button is clicked, by XULStarter with a set of custom-attributes declarations. Accordingly, XULStarter invokes the service operation "getTrendData" with an input parameter "pointId" whose value is retrieved from the <textbox> element, and obtains the returned XUL content of next step from the "getUIReturn" part of the output message.

As for the interaction behavior on client side, the Ajax framework with SOAP is favored to handle the event-based interaction behavior exhibited by human user in client side browser. A script-based service interaction controller is developed to perform the interaction tasks on client side, such as message exchange, event handling, and service invocation. The proposed secure cross-domain service user interface communication protocol is designed and implemented in the script-based service interaction controller. It exploits the current browser security policy to enable secure inter-frame communication by sending or receiving the formatted messages among the named frames from multiple web sites across various domains; and therefore, no modification to the browser is required.

The application mediator communicates with the BPEL engine is to govern the dispatching of user interaction events from the service user interface rendering engine to the BPEL engine, and vice versa. Moreover, it provides an interface for the service user inter-face rendering engine to invoke operations that are provided by BPEL composite services.

7. CONCLUSION

This work presents a framework to integrate service composition flow with user interactions, with the following key features:

- EXUL to describe service user interfaces in BPEL;
- extensions to BPEL activity: interactPeople and renderUI, to specify the inner workings of user interactions in BPEL composite service as well as the rendering of service user interfaces;
- a wizard-style mechanism to guide the user to interact with the service flow in accordance with the sequence of service execution; and
- a secure cross-domain UI service communication protocol to facilitate secure cross-domain communication among the service user interface components from multiple web sites across various domains.

By means of these extensions, an enhanced BPEL engine with a service UI rendering engine is accordingly developed.

The main benefit of the proposed approach is to infuse user interactions into service composition, by which a deployed BPEL composite service can be consumed directly by human users via an integrated service workspace.

Additionally, an SOA-based application with user interfaces can be developed and maintained more flexibly by controlling the set of XML-based documents, including WSDL, extended BPEL, and extended XUL (EXUL), without a need to manage language-specific programs. In other words, a developer can save effort by using various visual tools to orchestrate XML-based documents rather than to develop the SOA applications by coding. The operational SOA applications can be built simply by deploying the aforementioned documents on the enhanced BPEL engine.

Our future research plan will further extend BPEL to deal with a variety of service composition issues, such as integration of intelligent services, support of client-side service computation, and monitoring and management of composite services.

REFERENCES

1. A. Alves, A. Arkin, and *et al.*, "Web services business process execution language (WS-BPEL) 2.0," Organization for the Advancement of Structured Information Standards (OASIS), April 2007.
2. A. Arkin, S. Askary, and *et al.*, "Web service choreography interface (WSCI) 1.0," BEA Systems, Intalio, SAP, and Sun Microsystems, 2002.
3. A. Barth, C. Jackson, and J. C. Mitchell, "Securing frame communication in browsers," in *Proceedings of the 17th USENIX Security Symposium*, 2008, pp. 17-30.

4. P. Bojanic, “The joy of XUL,” http://www.mozilla.org/projects/xul/joy-ofxul.html.
5. Oracle BPEL Process Manager Provides SOA and Integration Platform Support, http://xml.coverpages.org/ni2004-06-30-a.html.
6. WS-BPEL Extension for People (BPEL4People), http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/.
7. J. Burke, “Cross domain frame communication with fragment identifiers?” http://tagneto.blogspot.com/2006/06/cross-domain-frame-communication-with.html.
8. F. Curbera, F. Leymann, D. Roller, and S. Weerawarana, “Web services flow language (WSFL) 1.0,” IBM Corporation, May 2001.
9. C. Jackson and H. J. Wang, “Subspace: Secure cross-domain communication for web mashups,” in *Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 611-620.
10. JavaScript Object Notation (JSON), http://www.json.org.
11. M. Juric and D. H. Todd, “BPEL processes and human workflow,” http://webservices.sys-con.com/read/204417.htm.
12. M. Kassoff, D. Kato, and W. Mohsin, “Creating GUIs for web services,” *IEEE Internet Computing*, Vol. 7, 2003, pp. 63-73.
13. F. D. Keukelaere, S. Bhola, M. Steiner, S. Chari, and S. Yoshihama, “SMash: Secure cross-domain mashups on unmodified browsers,” in *Proceedings of the 17th International Conference on World Wide Web*, 2008, pp. 535-544.
14. M. Kloppmann, D. Konig, F. Leymann, G. Pfau, and D. Roller, “Business process choreography in websphere: Combining the power of BPEL and J2EE,” *IBM Systems Journal*, Vol. 43, 2004, pp. 270-296.
15. P. Krill, “BEA upgrading business process integration package,” http://www.inforworld.com/article/05/08/22/HNbeaintegration_1.html.
16. J. Lee, S. P. Ma, S. J. Lee, Y. C. Wang, and Y. Y. Lin, “Dynamic service composition: A discovery-based approach,” *International Journal of Software Engineering and Knowledge Engineering*, Vol. 18, 2008, pp. 199-222.
17. J. Lee, Y. C. Wang, C. L. Wu, S. J. Lee, S. P. Ma, and W. Y. Deng, “A possibilistic petri-nets-based service matchmaker for multi-agent system,” *International Journal of Fuzzy Systems*, Vol. 7, 2005, pp. 199-213.
18. Z. Maamar, S. Mostefaoui, and H. Yahyaoui, “Toward an agent-based and context-oriented approach for web services composition,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, 2005, pp. 686-697.
19. XAML Overview, http://msdn.microsoft.com/en-us/library/ms752059.aspx.
20. XML User Interface Language (XUL) Project, http://www.mozilla.org/projects/xul/.
21. The same origin policy, http://www.mozilla.org/projects/security/components/same-origin.html.
22. OASIS Web Services for Remote Portlets Specification (WSRP), http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html.
23. OASIS User Interface Markup Language (UIML), http://www.oasis-open.org/committees/uiml/.
24. Oracle BPEL Process Manager, http://www.oracle.com/technology/products/ias/bpel/index.html.
25. Cover Pages XML Markup Languages for User Interface Definition, http://xml.coverpages.org/userInterfaceXML.html.

26. ZK, http://www.zkoss.org/.
27. Scriptol.com, "Which interface for a web application?" http://www.scriptol.com/ajax-xul-xaml.php.
28. J. Soriano, D. Lizcano, J. J. Hierro, M. Reyes, C. Schroth, and T. Janner, "Enhancing user-service interaction through a global user-centric approach to SOA," in *Proceedings of the 4th International Conference on Networking and Services*, 2008, pp. 194-203.
29. D. Thorpe, "Secure cross-domain communication in the browser," http://msdn.microsoft.com/en-us/library/bb735305.aspx.
30. UIML.org, User Interface Markup Language (UIML), http://www.uiml.org/.
31. W. van der Aalst, M. Dumas, and A. ter Hofstede, "Web service composition languages: Old wine in new bottles?" in *Proceedings of the 29th Euromicro Conference*, 2003, pp. 298-305.
32. The Forms Working Group, http://www.w3.org/MarkUp/Forms/.
33. HTML 4.01 Specification, http://www.w3.org/TR/REC-html40/.
34. User Interface Markup Languages, http://www.xul.fr/comparison-userinterface-markup-languages.html.

Jonathan Lee (李允中) is a professor in the Computer Science and Information Engineering at National Central University (NCU) in Taiwan, and was the department chairman from 1999 to 2002. He is currently the director of Computer Center at NCU. His research interests include agent-based software engineering, service-oriented computing, and software engineering with computational intelligence. He has authored more than 100 journal and refereed conference papers, and was the editor-in-chief of International Journal of Fuzzy Systems and in the editorial boards of Fuzzy Sets and Systems, International Journal of Artificial Intelligence Tools, Fuzzy Optimization and Decision Making, International Journal of Artificial Life Research, Open Software Engineering Journal, International Journal of Applied Computational Intelligence and Soft Computing and International Journal of Soft and Intelligent Computing and Mathematics. He received his Ph.D. in Computer Science from Texas A&M University in 1993. He is the president of Taiwan Software Engineering Association, a senior member of the IEEE Computer Society and a member of the ACM.

Ying-Yan Lin (林英彥) received his B.S. degree in Computer Science and Information Engineering from Tunghai University, Taiwan, in 2000, and he received his M.S. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2002. He is currently a Ph.D. student in the Department of Computer Science and Information Engineering of National Central University. His current research interests include service-oriented computing and software engineering.

Shang-Pin Ma (馬尚彬) received his Ph.D. and B.S. degrees in Computer Science and Information Engineering from National Central University, Chungli, Taiwan, in 2007 and 1999, respectively. He has been an assistant professor of Computer Science and Engineering Department, National Taiwan Ocean University, Keelung, Taiwan, since 2008. His research interests include software engineering, service-oriented computing and software process improvement.

Shin-Jie Lee (李信杰) received his Ph.D. degree in Computer Science and Information Engineering from National Central University, Taiwan, in 2007. He is currently a postdoctoral researcher in Software Research Center at National Central University. His current research interests include agent-based software engineering and service-oriented computing.