Mashing-up Rich User Interfaces for Human-Interaction in WS-BPEL

Qi Zhao[1,2], Xuanzhe Liu[1,2]*, Dawei Sun, Tiancheng Liu[3], Ying Li[3], Gang Huang[1,2]
[1]Key Laboratory of High Confidence Software Technologies, Ministry of Education
[2] School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China;
[3] IBM Research - China, Beijing, 100193, China;
{zhaoqi06, liuxzh, sundw}@sei.pku.edu.cn, {liutc, lying}@cn.ibm.com, huanggang@sei.pku.edu.cn

***Abstract*—Services computing paradigm together with Web services have significantly promoted the automation of business process in enterprise. Prevalent service composition technologies, such as WS-BPEL and WSCI, provide promising means to deal with machine-to-machine communication. Traditionally, in the phase of business process modeling, there usually require some human-involved tasks. Recent new technologies such as BPEL4People and Human Task begin to consider involving human interaction in business process. However, such approaches still have some limitations. On one hand, they exactly require some extensions of current BPEL standards. As a result, the existing business processes have to be rewritten and redeployed. On the other hand, they yet lack of the development and deployment supports of flexible and reusable user interfaces in business process. In this paper, we address these issues by enabling human interaction in business process with rich web applications. Our approach models human tasks as services, and can be seamlessly integrated to current BPEL without any modifications to existing engine and processes. We further support building human task presentations from service-oriented rich user interfaces. During the process execution, the corresponding task stakeholders can select, configure and compose these reusable and rich UI components according to their own application context.**

Keywords-web service; service composition; BPEL; human task;

I. INTRODUCTION

In the recent years, the Services Computing paradigm has been widely adopted. It allows the both enterprises and end-users to participate and collaborate for their own interests and benefits by means of service composition. Particularly, supported by the Web services technologies, business process automation has significantly evolved in a service-oriented fashion. Web services providing specific functionalities can be assembled according to specific business logics, represented in form of business process model (e.g, WS-BPEL and WSCI), and finally deployed and executed on a process engine. The most dominant service-oriented business process specification, WS-BPEL, has been very popular in enterprises together with useful tools support, such as IBM WPS [1] and Active BPEL Engine [2].

One of the fundamental assumptions of current service composition technologies is the automated execution of business process in a machine-to-machine communication manner. In other words, Web services providing specific functionalities are fully orchestrated without involving any human tasks. Human-involved workflows are very important topics in traditional business process research. Common human activities in processes involve data input and validation as well as decision making. Obviously, most of current service-oriented business process standards and engines do not well reflect the undisputed importance of human interactions [3]. Recently, several vendors like IBM and Oracle provide proprietary BPEL extensions in their engines to support such "human tasks". Promising recent proposals like BPEL4People[4] and WS-HumanTask [5] allow for a standardized integration of human-based activities in BPEL processes.

However, we argue that there yet remain some problems when adopting human tasks in service-oriented business process. On one hand, neither BPEL4People nor WS-HumanTask can be seamlessly integrated to current WS-BPEL editors and engines without any modifications. It is the fact that they both require extensions beyond current WS-BPEL standards, while current process editors and engines have to be enhanced to support the new features. Therefore, current running processes may require re-developed and re-deployed. On the other hand, user interfaces usually play an important role in human-centered business process, for example, data visualization (interactive graphs and tables), human action presentation (textboxes and submit/cancel buttons) or even multimedia integration (image slide shows). Current business process techniques mainly focus on the process modeling, specifications and execution, while not address the presentation rendering issues for human tasks. Moreover, as the human interaction logics should well adapt the service functionalities and business logics, it exactly requires the development of flexible and reusable rich user interfaces without too time-consuming and costly efforts.

To adequately respond to the challenges in human tasks in service-oriented business process, this paper proposes a novel approach to mashing up rich user interfaces in service-oriented business process. First of all, without disruption for current WS-BPEL standards, we model human tasks as services and introduce the concept

* Corresponding author: liuxzh@sei.pku.edu.cn

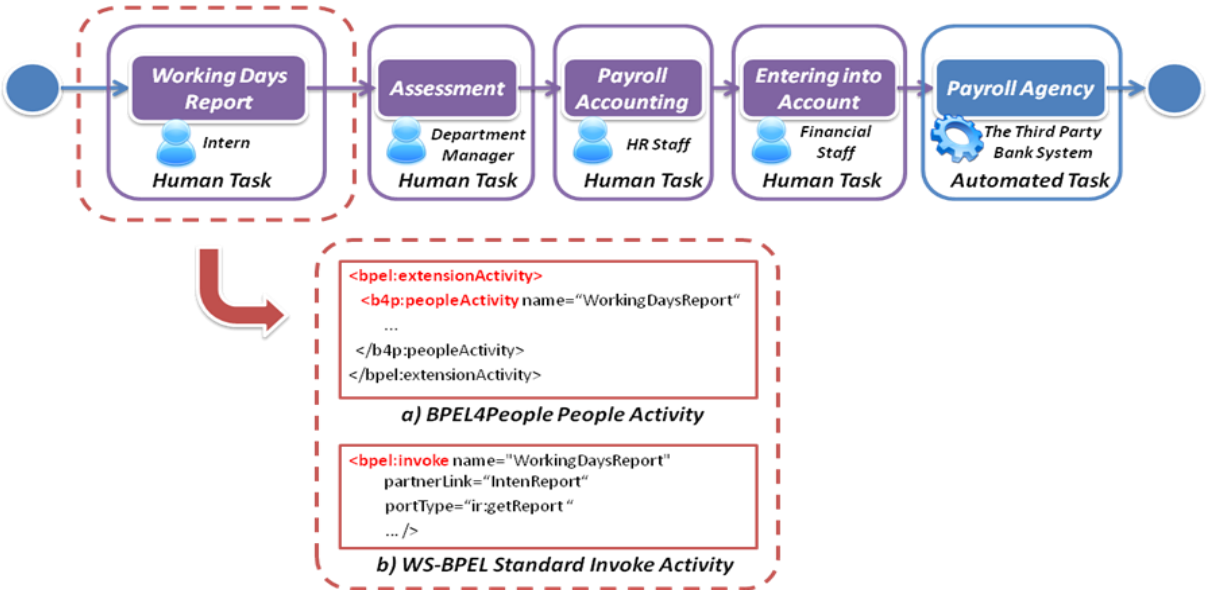

Figure 1 Business Process for Intern Payroll

of Human-Interaction Services (HTS), a special type of Web service taking charge of presentation logics and coordinating with the Web services with business functionalities. Therefore, the HTS can be seamlessly integrated into current business process. Secondly, we show how to automatically generate rich user interfaces for HTS leveraging the iMashup project toolkit, which is a web browser-based mashup toolkit. iMashup project implements a component-based, on-the-fly mashup composition builder and runtime [6][7] so that non-professional mashup developers, such as business process modelers and designers, are easy to utilize the auto-generated UI and create their own rich, visual and friendly HTS UI.

The remainder of this paper is organized as follows: Section 2 illustrates a motivating scenario and discusses the major related work, BPEL4People. Section 3 illustrates an overview of our approach. Section 4 provides the details of our HTS engine, and rich UI building and running environment. Section 5 describes our prototypical toolkit and how it deals with the sample scenario. Finally, we discuss some improvements in Section 6 and conclude this paper in Section 7.

II. BACKGROUND

A. Motivating Scenario

In this section, we present an intern payroll process as a scenario which explains our motivation. In many companies, interns should report their working days every month, since their working time is very flexible. In the salary payment process, the first step is that interns report their working days. Then their department managers will check whether these reports are appropriate or not. If the managers confirm the reports, HR staffs will compute the intern payroll and financial staffs will submit this into the account system. Finally, the process will invoke a payroll agency service provided by the third party bank system to put the salaries into the interns' account. Figure 1 shows the sample process.

This scenario obviously illustrates that human tasks are necessary in business process, since the first four steps of the process all require human-involved tasks. We call them Human-Interaction Services (HTS) in this paper. It also presents that the HTS exactly requires rich, flexible user interfaces (UI). For example, when the interns submit working days or the department mangers check the reports, a visual calendar might be required, since a series of dates are hard to be input, read and understood by people without help from the calendar. Similarly, both HR staffs and financial staffs need some rich UI to assist their work.

B. BPEL4People and WS-HumanTask

WS-BPEL was originally proposed to enable automated Web service orchestration according to specific business logics. Therefore WS-BPEL mainly supports machine-to-machine communication without involving human tasks. As WS-BPEL has been the most popular standard in practice, there have been some complementary works for human-tasks beyond WS-BPEL. The most typical works are BPEL4People and WS-HumanTask released in 2007. BPEL4People introduces the notion of "***PeopleActivity***" as a new type of basic activity which enables the human-interaction in BPEL processes. The details of PeopleActivity are described by WS-HumanTask. A sample PeopleActivity for the "Working Days Report" human task is shown in Figure 1 a).

The main problem of BPEL4People is that the PeopleActivity cannot be directly and seamlessly integrated into WS-BPEL. As we can see from Figure 1 a),

BPEL4People uses BPEL extension activity and imports a new element "***b4p:peopleActivity***". Obviously, such extension requires adaption of current WS-BPEL editor and engine. Otherwise BPEL4People processes cannot be edited and executed in these tools, because PeopleActivity is far different from standard WS-BPEL invoke activity (Figure 1 b)).

On the other hand, there is a lack of concept for the definition and deployment of flexible and reusable rich UIs in BPEL4People and WS-HumanTask based processes [8]. The development and deployment of rich UIs for human tasks is usually time-consuming and costly in these processes. Although CRUISE[8] provides an approach to building rich UI, the solution exactly relies on WS-HumanTask and cannot be seamlessly applied into standard WS-BPEL. Furthermore, this approach does not support UI automatic generation to facilitate rich, flexible and reusable UI.

III. Approach Overview

Based on the analysis above, we provide an approach to seamlessly integrating human tasks in WS-BPEL as well as automatically generating flexible user interfaces for human tasks. The most important principle of our HTS and WS-BPEL integration approach is that the solution should adhere to standard WS-BPEL specifications without extensions. Therefore, in our approach, all interactions between the HTS and standard WS-BPEL tools strictly follow the standard WS-BPEL and Web service specifications, and exchange standard BPEL or Web service artifacts, such as WSDL files and SOAP messages. Figure 2 describes the architectural overview of our approach.

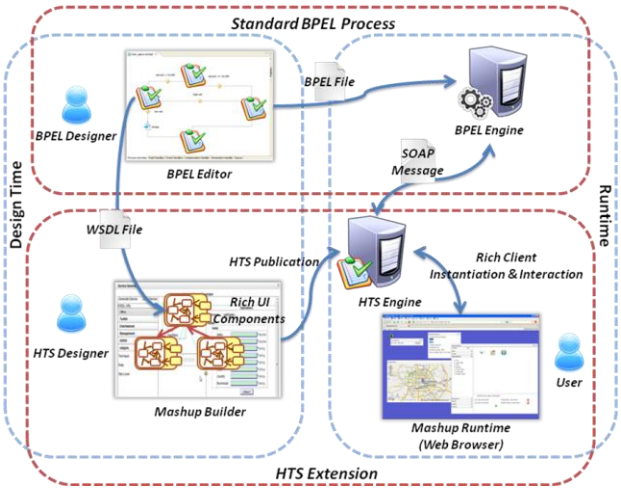

Figure 2 Approach Overview

The upper part of Figure 2 presents the standard WS-BPEL design and execution process. At design time, designers build WS-BPEL processes in editors as usual, and their outputs are standard WS-BPEL files. These files can be deployed to any standard WS-BPEL compliant engine. Neither editors nor engines need offer additional supports for HTS. The HTS support is shown in the lower part of Figure 2. This part also follows the separation of design time and runtime. As mentioned above, our solution integrates a mashup composition approach to implementing rich UI of HTS. Therefore we provide a mashup builder at design time to facilitate rich UI of HTS. The mashup builder communicates with the WS-BPEL editor through WSDL files. When a Web service in WS-BPEL process is specified as a HTS, designer can submit its WSDL file to mashup builder, and the builder will parse the inputs and outputs of operation and generate a basic rich UI. In iMashup toolkit, there are a series of built-in visual mashup components which are used to enhance the user experience beyond basic UI. Finally, the HTS and its rich UI are published to the HTS engine (whose design details will be described in Section 4). The engine will automatically assign a new binding for this service and WS-BPEL designers only need to set the "invoke" service binding to the new one. Then this HTS is online and able to be composed in any WS-BPEL process.

In our HTS design time mechanism, both HTS and common Web services with business functionalities are described as standard WSDL files. Therefore, HTS can be imported into standard WS-BPEL editors and integrated into BPEL processes seamlessly. Another benefit of this seamless integration is that the process need not be modified when automated business services and human tasks replace with each other. For example, in our motivating scenario, the company might deploy a management system to standardize intern management one day. In that case, the first HTS in the payroll process should be replaced by an automated report service of the management system. Since both HTS and automated services are described by same "invoke" activity in BPEL process, when the new intern management system is deployed, the BPEL designer just need modify the binding in WSDL file instead of re-developing the process specifications.

After the new binding is completed, all requests to HTS will be redirected to HTS engine. The details of human interaction are shielded behind this HTS engine instead of being exposed to standard WS-BPEL engines. To achieve this goal, we provide a mashup runtime [7] which is embedded in web browser. The mashup runtime monitors a pending services list in the HTS engine. When a SOAP request arrived, the HTS engine will create a new pending service and the mashup runtime will instantiate a corresponding rich UI. After task is completed, the submission data will be sent back to the HTS engine, packaged as a SOAP response and returned to BPEL engine. From perspective of BPEL engine, it sends a SOAP request to HTS engine and receives a SOAP response. All messages exchanged are standard. By this means, the whole procedure has no difference from a standard Web service invocation.

IV. HUMAN-INTERACTION SERVICES IN WS-BPEL PROCESSES

This section describes the details of some important components of our HTS supporting mechanism, including HTS engine and mashup builder. And we also present some key technical challenges of our solution. Figure 3 shows the internal structure of HTS engine and mashup runtime. It also demonstrates the communication actions among them.

A. Human-Interaction Services Engine

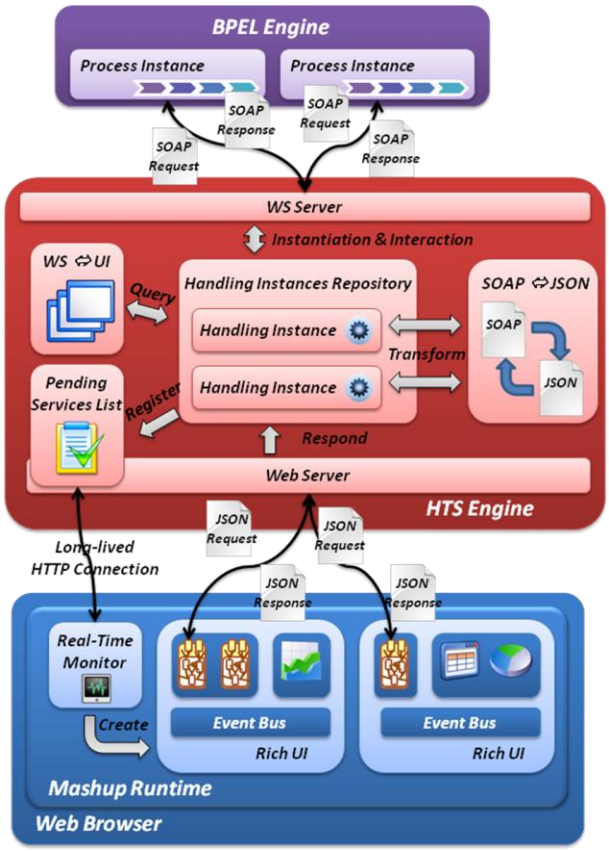

Figure 3 Interactions among BPEL Engine, HTS Engine and Mashup Runtime

The upper part of Figure 3 displays the details of HTS engine and how it communicates with WS-BPEL engine. When a SOAP request arrives at HTS engine, a Web services server (in our implementation, we integrate Apache Axis [14]) in the engine will handle the details of the underlying protocols. Then a handling instance will be created and transform SOAP request message into JSON (JavaScript Notation Object) [15] format, query corresponding rich UI for this HTS, and then register a new service request(?) into the pending service request? list. The mashup runtime will help users to finish the task and return data in JSON format to the web server in the HTS engine. When the HTS handling instance receives the response data, it will transform the JSON data into SOAP response and send it back to the WS-BPEL engine through the Web Service server.

Handling Instances Passivation

A critical problem of HTS engine is how to deal with the pending HTS requests/instances/handling instances?. Since people might not deal with tasks in time, HTS requests are often waiting for a long time before completed. If there are too many pending service handling instances, the HTS engine will probably become slow or even overload.

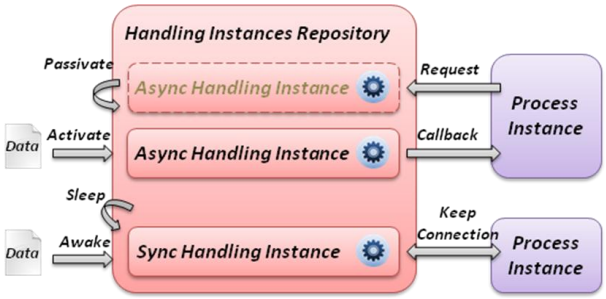

Figure 4 Handling Instances Passivation Mechanism

In our solution, we provide a handling instances passivation mechanism, which passivates pending services until human tasks are completed. This mechanism, as shown in Figure 4, deals with both asynchronous and synchronous requests. To passivate an asynchronous request handling instance, the mechanism serializes the instance into storage with its callback address as context, and free the instance. When user's submission arrives, the passivation mechanism will reconstruct that instance. Synchronous requests passivation is quite different. Synchronous requests should keep a live connection with the BPEL engine. Otherwise the service invocation will fail since the connection is closed. Therefore, when passivating a synchronous request, the mechanism just sleep the thread of that instance and awake it when data returns.

Real-Time Pending Services Monitoring

Although the HTS engine supports handling instances passivation, too many passivated instances (especially those synchronous ones keeping open connections) may still consume a lot of resources in the HTS engine. Therefore another technical challenge is how to notify users of the arrival of new tasks as soon as possible, since the delay of notifications will make pending service accumulate in the engine.

Unfortunately, the web browser, the platform for our mashup runtime, only supports pull data from server natively. Such "pull data" pattern means that users get new pending services only when they retrieve data on their own initiative. It does not well meet our requirement obviously. We need that the HTS engine "pushes" new pending services to the runtime. Therefore we implement a real-time pending services monitor in the mashup

runtime. Unlike the common "pull data" pattern between rich UI and web server, the monitor communicates with the pending services list in HTS engine through a streaming long-lived HTTP connection, which makes the server be capable of pushing data to the runtime hosted in web browser. The monitor sends a HTTP request to the pending services list at first. After the initial request, unlike common HTTP connections, the list does not close the connection, nor does it give a full response. It just keeps the connection open. Meanwhile, when a new HTS request is received, a new pending service will be added to the pending service list. The pending service list returns the newly added pending service to the monitor in HTTP chunked mode, using the same request and the connection. Then the monitor can notify users the new tasks. The open connection will be kept available until its timeout or some other reasons, e.g., browser shutdown. Once the connection is closed, the monitor will request a new streaming tunnel. This pending services push mechanism makes users to be able to get new list in real time.

B. Rich User Interface for Human-Interaction Services

Rich Components Mashup Approach.

In section 3 and 4.1, we have explained how the HTS engine acts as the mediator between the BPEL engine and UI of HTS. However, a sound human-interaction solution also needs to cover HTS UI development and deployment. Accordingly, we propose a rich component mashup approach to building rich, flexible and reusable user interface for HTS. Rich components encapsulate both UI and some application logics [6]. These components consist of the interfaces and the implementation. The component model is shown in Figure 5 a).

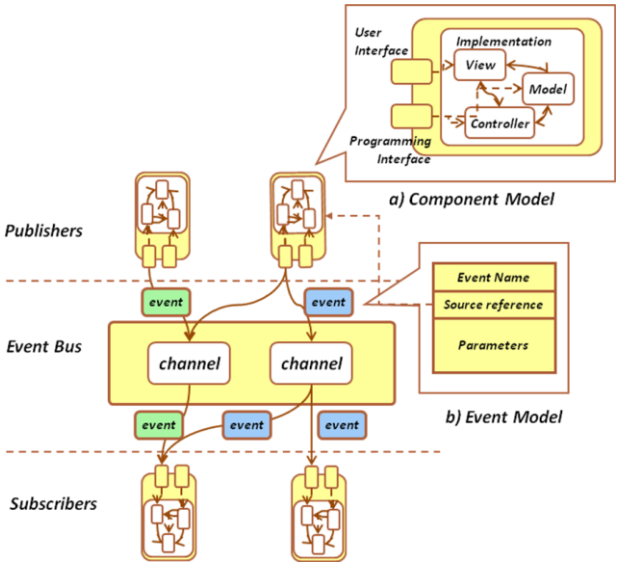

Figure 5 Component Model, Event Model and Event Bus for Rich UI

The implementation of rich components adopts the Model-View-Controller pattern. The model implements application logic for input parsing, data rendering, type checking, output sending and so on. The view is a fragment of HTML which is rendered for displaying data and responding user actions. The controller manages the interaction logic between model and view. The programming interface exposes the application logic of components. It consists of methods and events. Methods query and modify the component state. Events notify the changes of the component state and can be published into the event bus. The UI is implemented by the view part of component. When developers assemble the components, they can determine whether the UI of components should be shown or not (hidden).

Besides a component model, another technical problem is to provide a composition mechanism to mashup components together. Our approach provides an event-based publish-subscribe composition model thus, as show in Figure 5. The event-based composition model is well suited to rich UI for HTS, since the rich UI is running in web browser whose nature is strongly event-based [9].

To support event-based composition model, the composition model provides a unified event model for all events of components. The event model comprises the name of event, the reference of source instance triggering this event and a hash map containing event parameters, as shown in Figure 5 b). An event bus supports Publish-and-Subscribe event binding and provides some channels which are predefined by developers. The events of components can be published to these channels, while the methods of components can subscribe channels as well. The component publishes an event to the specific channel in the event bus when this event is triggered. Once an event arrives, the event bus will look up all subscribed methods of the given channel, traverse and invoke the corresponding methods with the event as input parameter.

We develop a mashup builder to support automatic rich component generation through WSDL files of HTS. It parses the input parameters of HTS operation, generates the model maintaining the input and the view displaying them in a name-value table. The output parameters are parsed for creating a web form which receives user submission data. A common controller takes charge of rendering display table with data model, checking values based on their types and sending submission data. HTS designers can enrich the automatic generated rich component by assembling built-in visual components, since the user experience of basic component may not be enough friendly as they wish. Each basic rich component has a default "onLoaded" event. This event is triggered when the input data is parsed, and processes the parsed data as parameters of the event. Visual components can subscribe this event and use the data rendering their rich UI, such as calendar or pie charts. Finally, the rich client and its source HTS can be published into the HTS engine as mentioned in section 3.

Mashup Runtime

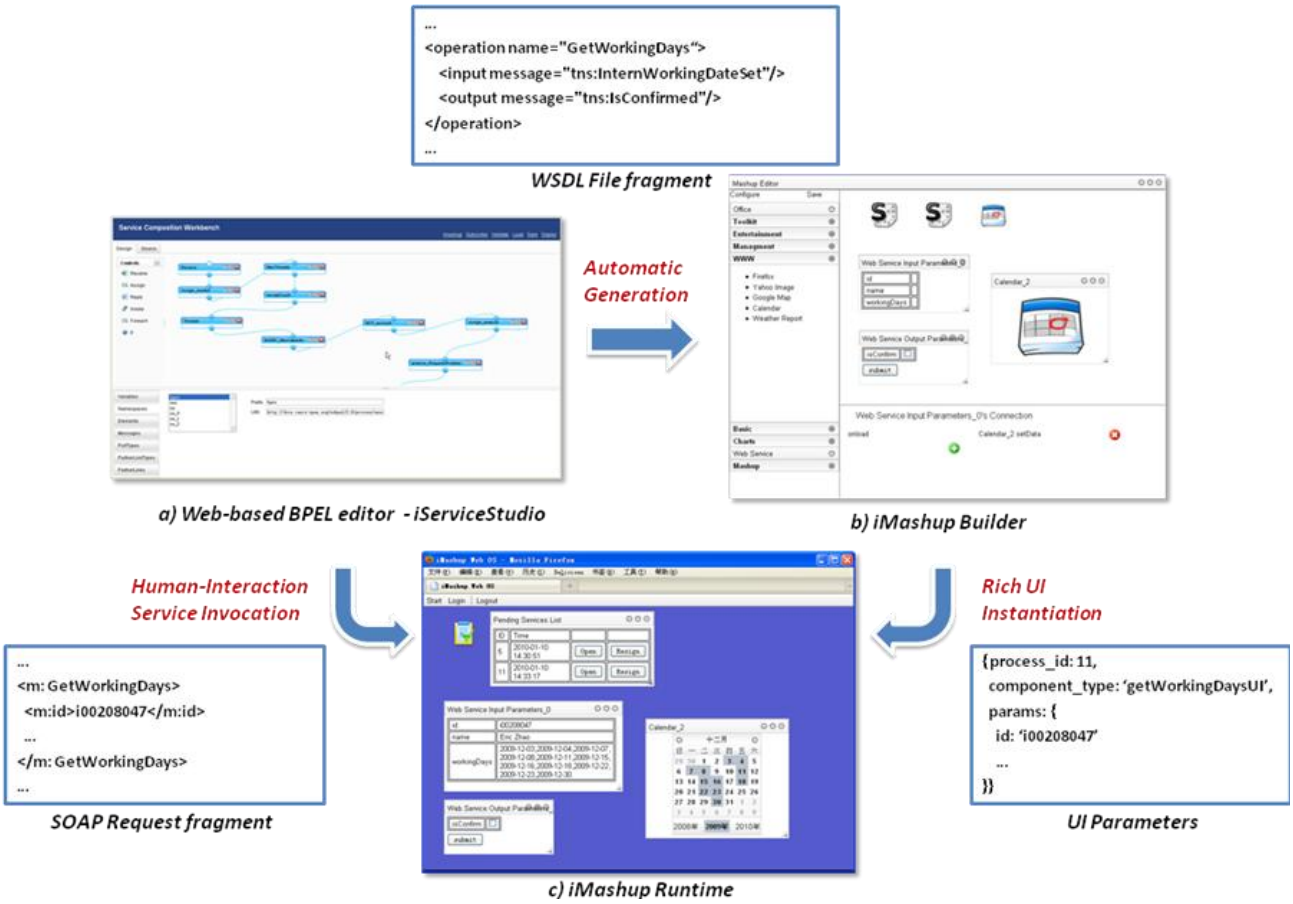

Figure 6 Human-Interaction Services Toolkit

The lower part of Figure 3 presents the details of interactions between the HTS engine and rich UI at runtime. As we mentioned in section 4.1.2, the mashup runtime embedded in web browser contains a real-time pending service list monitor. This monitor can acquire new pending services from the list once they published. After a new pending service arrives, runtime will create a new rich UI instance to help a user deal with the task. The mapping between rich UI and specific HTS are saved in the HTS engine when they are published.

V. TOOLKIT IMPLEMENTATION

The previous sections propose an approach that both integrate HTS into BPEL processes and adheres to WS-BPEL specifications. We evaluate this approach by building a prototype toolkit and implementing the scenario in Section 2 based on the toolkit. Some user interfaces are shown in Figure 6.

To prove the generality of our approach, we choose widely used WS-BPEL 2.0 compliant tools, Netbeans BPEL editor [16] and GlassFish BPEL engine [17], and build the intern payroll process. However, in practice, although Netbeans editor is able to integrate into our solution since both our solution and the tool adheres to BPEL specification, there are still some limitations. The other tools in our toolkit, iMashup builder and runtime, are both delivered through web browser, but NetBeans is a Java-based standalone editor. Designers should switch between browser and native window, when they use these tools. Therefore although every BPEL specifications compatible editor can be used in our approach, we implement a web-based WS-BPEL editor, iServiceStudio [10]. iServiceStudio runs in web browser, as shown in Figure 6 a). It is similar with Yahoo! Pipes and supports designers to visually edit WS-BPEL processes in drag-and-drop manner. iServiceStudio is able to generate standard WS-BPEL deployment packages and deploy them on GlassFish BPEL engine. With this web-based editor, the whole work, including process design and deploy, rich UI design and execution, can be completed in web browser. This improves the usability of our toolkit.

The second tool including in the toolkit is iMashup builder. iMashup builder supports WYSIWYG (What-You-See-Is-What-You-Get) rich component composition, as shown in Figure 6 b). When implementing the motivating scenario, the builder firstly generates input and output components, based on "GetWorkingDays" operation described in WSDL file. The components

display request data and gather submission. We load another visual calendar component, and connect it with input component as data source for displaying working days information friendly.

When intern payroll processes running, the HTS engine receives SOAP requests, transforms them into JSON format and publishes new pending service request. The real-time pending service monitor in web browser checks updates and refreshes a pending task table constantly. When a new item displays, the department manager can open rich UI, which is the composition result from the previous step. After the rich UI is instantiated, the instance gets JSON request, displays it in the input component and passes the data to that visual calendar. Figure 6 c) gives the screencast of that rich UI in iMashup runtime. The department manager checks the information and decides whether to confirm the report or not.

VI. Discussion

Though we have observed that our HTS extension does benefit human-involved BPEL processes, there are still several issues to address.

Firstly, although the handling instances passivation mechanism can deal with the task processing speed mismatch between human and automated machine to a certain degree, it is still hard to handle long-time running human tasks, as they might keep executing for several days or even several weeks before completed. Especially, if designers use synchronous "invoke" activities to model these long-running tasks, the connections for service invocations will always be closed before the tasks are completed. Therefore, the long-running tasks should be modeled as asynchronous activities. In iServiceStudio, designers can declare a synchronous HTS as a long-time running task. If they do so, iServiceStudio will remove the synchronous "invoke" activity, replace with the asynchronous "invoke-and-receive" activities and give a long timeout value. Although the series of actions will modify the initial process and result in re-deployment, it seems that there is no better solution to deal with the long-running human tasks.

In practice, some scenarios may require a sub-flow including several WS-BPEL activities to be mapped to one human-involved task. For example, in a system management WS-BPEL process, the administrator may set up parameters of different servers (web server, application server and database) in each flow step. However, these activities may relate to others, e.g. application server needs IP address of database. Therefore if one human-interaction service supports multiple WS-BPEL activities, the task can be proceeded more quickly and much richer UI can be offered since the related information from different activities can be handled together. As we known, mashup is not only UI composition but also a lightweight approach for service composition [11][12]. Consequently, our rich UI composition approach can be easily improved to support multi-activities human-interaction services. This enhanced solution allows designers to create one mashup application from a sub-flow of BPEL and replace the sub-flow with a HTS binding with that mashup application.

Another important problem in human-involved processes is how to integrate real people into WS-BPEL processes, which lead to human roles, people identifying and grouping, people linking with activities, people assigning and so on. BPEL4people specification has several modeling concepts to deal with people integration. This topic is out of scope of this paper. However, we are considering people integration in future work, employing the human workflow model proposed in [13].

VII. Conclusion and Future Work

SOA and WS-BPEL provide a rapid, flexible and loosely coupled manner to seamlessly integrate the enterprise resources into business processes. Many business processes require human-involved tasks in practice. However, human-interaction services are not covered by WS-BPEL, which is primarily designed to support automated machine-to-machine communication. Although BPEL4People and WS-HumanTask introduce an extension to address HTS into WS-BPEL, they require modification of the WS-BPEL specification and cannot be applied into standard BPEL editors and engines.

In this paper, we propose a human-interaction services extension mechanism which adheres to standard WS-BPEL specifications. The main contributions of this paper include:

- Enabling Human-Interaction Service in standard WS-BPEL Processes without specifications modification or extension;
- Automatically generating rich UI for Human-Interaction Services and providing a rich component mashup approach allowing users to enhance the rich UI.

As we mentioned in section 6, there are some open issues for our HTS extension approach. In the future, we plan to investigate into people integration in standard WS-BPEL. We are also further interested in converting several HTS into one rich UI if these HTS are related and may be assigned to the same user. The combining UI may speed the task processing and offer friendlier user experience. Finally we will perfect the prototype toolkit to improve its usability, performance and so on.

Acknowledgements

This work is partly sponsored by the National Key Basic Research and Development Program of China under Grant No. 2009CB320703; the National Natural Science Foundation of China under Grant No. 60821003, 60873060, 60933003; the High-Tech Research and Development Program of China under Grant No. 2009AA01Z16; and the IBM-University Joint Study Program.

REFERENCES

[1] IBM WebSphere Process Server , http://www.ibm.com/software/integration/wps/

[2] ActiveBPEL Engine, http://www.activevos.com/community-open-source.php

[3] Matthias Kloppmann, et al. WS-BPEL Extension for People-BPEL4People. International Business Machines Corporation and SAP AG, 2007.

[4] Ashish Agrawal, et al. WS-BPEL Extension for People (BPEL4People), Version 1.0. Active Endpoints Inc., Adobe Systems Inc., BEA Systems Inc., International Business Machines Corporation, Oracle Inc., and SAP AG, 2007.

[5] Ashish Agrawal, et al. Web Services Human Task(WS-HumanTask), Version 1.0. Active Endpoints Inc., Adobe Systems Inc., BEA Systems Inc., International Business Machines Corporation, Oracle Inc., and SAP AG, 2007.

[6] Qi Zhao, Gang Huang, Jiyu Huang, Xuanzhe Liu, Hong Mei, Ying Li, Ying Chen. An On-the-fly Approach to Web-based Service Composition. Proceedings of IEEE Service Congress and International Conference on Web Services, 2008 (SCC 2008).

[7] Gang Huang, Qi Zhao, Jiyu Huang, Xuanzhe Liu. Towards Service Composition Middleware Embedded in Web Browser. International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009 (CyberC 2009).

[8] Stefan Pietschmann, Martin Voigt, and Klaus Meißner. Adaptive Rich User Interfaces for Human Interaction in Business Processes, International Conference on Web Information Systems Engineering, 2009 (WISE 2009), pp351-364.

[9] Jin Yu, Boualem Benatallah, Fabio Casati, Florian Daniel, Understanding Mashup Development. IEEE Internet Computing, No.5, 2008. pp 44-52.

[10] Xuanzhe Liu, Gang Huang, Pei Wen, Hong Mei. Discovering Homogeneous Web Service Community in the User-Centric Web Environment. IEEE Transactions on Services Computing, Vol 2, No.2, April-June, 2009, 167-181.

[11] F. Curbera, M. Duftler, R. Khalaf, and D. Lovell, Bite: Workflow composition for the web. in Proceeding of International Conference on Service-Oriented Computing, K.-J. Lin and P. Narasimhan, Eds. Berilin-Heidelberg: Springer-Verlag, 2007, pp. 94-106.

[12] Xuanzhe Liu, Wei Sun, Yi Hui, Haiqi Liang. Investigating Service Composition based on Mashup. Service Congress 2007. pp 332-339.

[13] Xiangpeng Zhao, Zongyan Qiu, Chao Cai, Hongli Yang. A Formal Model of HumanWorkflow. Proceedings of 2008 IEEE International Conference on Web Services (ICWS 2008).pp 195-202.

[14] Apache Axis, http://ws.apache.org/axis/

[15] JSON. http://www.json.org

[16] NetBeans with the Enterprises Pack, http://soa.netbeans.org/soa

[17] GlassFish, https://glassfish.dev.java.net/

[18] Yahoo! Pipes, http://pipes.yahoo.com