On Supporting the Design of Human-Provided Services in SOA*

Daniel Schall, Christoph Dorn, Hong-Linh Truong, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
Argentinierstr 8/184-1, 1040 Vienna, Austria
{schall,dorn,truong,dustdar}@infosys.tuwien.ac.at

Abstract. Collaboration platforms evolve into service-oriented systems, promoting composite and user-enriched services. The problem we address in this paper is the support of human interactions in SOA. Current collaboration tools do not support humans to specify different interaction interfaces (services), which can be reused in various collaborations. We focus on the design of Human-provided Services (HPS). Our contributions center around two main aspects of human interactions in SOA: (i) an approach for designing service interfaces embodying human activities as actions offered by Web (HPS) users; (ii) a tagging model for activities and services to recommend resources in the design process. We discuss techniques for mapping human activities onto Web services. We present a recommendation algorithm that is based on collaborative tagging of resources in SOA. Our algorithm helps to determine suitable resources drawn from properties of user preferences and measured similarity of human activities and actions.

1 Introduction

The global nature of the Web promotes access to the knowledge of an immense large number of people. The Web enables the participation of users in collaborations by various means. Currently, users interact with the Web and share content, their knowledge, and opinions, etc. through Web sites, forums, Wikis, or blogs. We have taken user participation on the Web a step further by introducing Human-provided Services (HPS) utilizing SOA technologies and Web services [1]. People can publish activities and their capabilities as Web services, creating a Web of HPSs interwoven with current Wikis, blogs, social networks and enterprise services. We believe that HPS will be the future trend of human participation on the Web and enterprise collaboration. Therefore, supporting users in designing HPSs in an easy manner is an important issue. HPSs are exposed as real Web service interfaces acting as interaction interfaces toward humans. From the user's point of view, services are represented as *activities* and *actions* a user can perform in SOA-based collaboration environments. We believe that users should be empowered to define and provide services. HPS use cases include

* Part of this work was supported by the EU project inContext (IST-034718).

G. Feuerlicht and W. Lamersdorf (Eds.): ICSOC 2008, LNCS 5472, pp. 91–102, 2009.

i) obtaining input or opinion from experts, ii) interactions with infield workers to perform certain tasks, iii) mass collaboration of globally distributed teams or communities by gathering results/output from users depending on expertise, geographic location, and availability.

We have designed and implemented a framework [2] with the aim of supporting HPSs as *user-contributions* in collaborations. While approaches such as BPEL4People [3] target the support of human interactions as part of business processes (i.e., workflows) by designing and executing a set of *human tasks* (cf. WS-HumanTask [4]), HPS enables the definition of services offered by humans independent of any process or task. We distinguish between **(a)** service — at the technical level encompassing the definition of domain specific interaction interfaces and input/output parameters — defined by users and/or communities and **(b)** tasks to model the demanded (human) input as part of a workflow.

The goal of HPS is to provide a framework and tools enabling the engagement of humans in distributed, large-scale collaborations using SOA. To enable human participation through HPS, users must be able to utilize tools to design and model their participations. These tools must be simple yet powerful enough to deal with complexities in the service-design process. To date, most effort has been spent on tools for Web service professionals and developers which however cannot be used by novice users. Mashup editors[1] are examples of how simple tools can facilitate user participations and user driven processes by gathering and aggregating different sources of knowledge. We argue that similar tools should be provided for HPS design, enabling novice users to design their personal services, thus creating a novel blend of SOA comprising human and software services. In our previous work, we introduced a middleware enabling HPSs (i.e., registry, discovery, and interactions). In this paper, we present methods and tools supporting the user in the design of HPSs in SOA-based environments. We analyze the complexity and challenges of the design process and present our solution.

Our goal is to provide powerful yet simple tools for users to define and provide services. Such tools should automatically generate all the artifacts needed to allow users to fully participate in interactions in SOA using Web services. This paper presents an architecture and its implementation allowing humans to design services for ad-hoc and process-centric collaborations, with the following key contributions, (i) a methodology for incorporating human interactions in SOA, (ii) design and implementation of the HPS designer architecture, giving users the ability to design activity-centric interfaces which can be translated into low level services descriptions (e.g., described in WSDL), and finally (iii) a method for helping users in providing the right service by using tagging methods.

This paper is organized as follows: in Sec. 2 we outline our approach and propose our solution to designing HPSs in SOA. We given an overview of the HPS framework in Sec. 3, followed by Sec. 4 describing transformations and mappings of human activities onto services. We introduce our recommendation algorithm in Sec. 5 and show implemented tools for ranking and design in Sec. 6. Finally, we discuss related work in Sec. 7 and conclude the paper in Sec. 8.

[1] E.g., http://pipes.yahoo.com/pipes/

2 Collaboratively Designing HPS

Dynamic collaborations typically take place using various communication channels and tools. The HPS middleware is a platform targeting SOA-based collaboration scenarios involving both human and software services. In this section, we first discuss the challenges in designing HPSs and present our approach. Then, we provide an overview of the steps in the design process and show how users are supported in finding/reusing exsting service artifacts.

- *Interface transformation and generation:* Designing and providing a service should be as simple as writing a "blog entry". Mapping human activities onto Web services is challenging. Novice users have to be supported in designing services in an easy and intuitive manner by hiding underlying complex processes, constituting automatic service interface generation and translation of service interfaces (e.g., WSDL) into GUI representations. Since standard Web service technologies are used — at the technical level — to enable HPS, versatile collaborations can be supported including interaction between humans as well as using HPSs in, for example, formalized processes. Human input in a process is depicted in Fig. 1 (a) as `Sends request`.
- *Recommendations for HPS Design:* We argue that humans should be able to design and provide their capabilities as services. Many HPSs may be available and registered as services, potentially with different interface characteristics and expertise of users. Users have to be supported in the design process by recommending resources and interfaces which may already exist. We propose tagging mechanism helping users in expressing their *expertise* and an algorithm based on collaborative filtering methods for ranking HPSs. Furthermore, by tagging SOA artifacts, human activities, and actions – defining

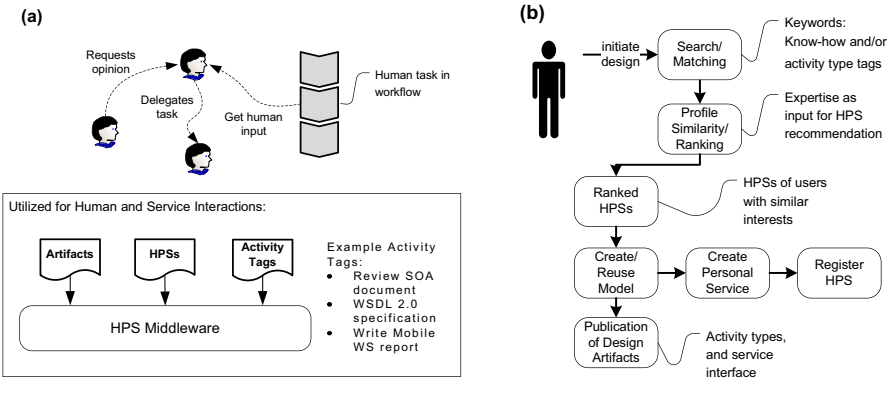

Fig. 1. (a) Typical collaboration scenario involving human- as well as human-service interactions. Tags are applied to various artifacts allowing for classification of services and user activities. **(b)** The design: users can reuse existing models or define new interfaces. Services are registered as personal services.

the type of collaborations in which an HPS is used – allows search based on user-defined keywords.

In Fig. 1, we show how the design is supported by utilizing tagged information and collaborative filtering methods. Tagging becomes increasingly important in today's collaborations allowing people to associate metadata to various artifacts — Web documents, links, messages, etc. (e.g., see [5] for usage patterns of tagging) and keyword-based search of user annotated content. Similarly, tags in the HPS framework are used to identify the *context* in which services and artifacts are used.

Generally speaking, tags are keywords/terms associated to information. In this work, we distinguish between tags assigned to *activities*, *services*, and *actions*. In the following we discuss how to utilize this metadata in the design process and detail the steps in Fig. 1.

1. Let's assume a user initiates the design process by deciding to define a service Fig. 1 (b). The common thing between the design of HPSs and, for example, creating service mashups is that both are user-defined services or processes. Like in most collaboration platforms, user-profiles managed by the HPS middleware contain information such as past/current activities and user preferences. Examples — in the computer science context — include "reviewer for a conference", "J2EE consultant", etc. We refer to such keywords available in the profile as activity information (e.g., predefined by the user).
2. The search for service artifacts is performed by matching the user's query vector against existing HPSs. A matching function takes service metadata as input, either automatically extracted keywords or, again tagged information. Hence, tagging information is not only used during collaboration, but also at design time. In contrast to above mentioned activity tags, service tags express how HPSs are used for collaborations (i.e., for actual interactions).
3. The next step is the ranking of HPSs. We compare a user's activities with the expertise of those HPSs matching the demanded set of keywords.
4. The user can create new models, publish related artifacts and type definitions using tools discussed in Section 4, or reuse existing HPS definitions. The model defines human activities which are mapped into *actions* (cf. [6]) performed using Web services.
5. Finally, *Personal Services* (the mapping of user profiles to services) are published in the HPS registry.

3 Overview HPS Framework

To enable HPS, we need a framework supporting the management of related artifacts, user profiles, and HPS interactions. The HPS framework has been developed for this purpose (cf. [2]). However, in this paper we focus on design related APIs and components which have been implemented on-top of the core framework. In the following, we provide a description of components in Fig. 2.

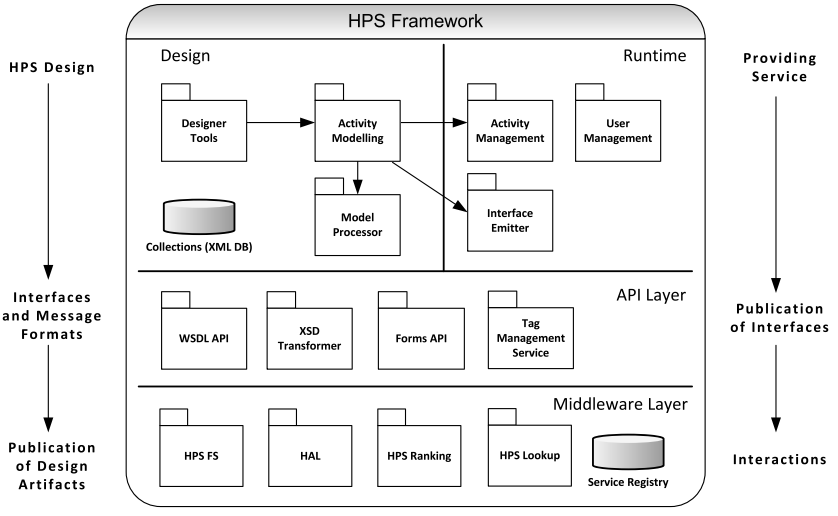

Fig. 2. HPS framework and architecture

The *API Layer* includes the core services for WSDL document generation (*WSDL API* service) which specify human activities and user specified interface elements (parameters and complex elements), the *Forms API* implementing support for XML Forms (XForms[2]), an *XSD Transformer* service utilizing the *Forms API* to automatically generate XForms based on XML schema definitions, for example, as defined in WSDL documents, and a *Tag Management* service associating tags with HPS artifacts (activities, actions, and WSDLs).

Design: Fig. 2 shows the design flow on the left side (top down) starting with *HPS Design* – designer tools such as a Web portal allowing users to create service interfaces in a simple manner.

1. *Interface and Message Formats:* the HPS framework provides tools to automatically translate high level specifications (e.g., activities and interface elements) into low level service descriptions without requiring the user to understand underlying technologies such as XML or WSDL.
2. *Publication of Design Artifacts:* artifacts such as message formats and activity type definitions are saved in XML collections.

Runtime: The *User Management* service holds user-related data such as profiles and contact details and is utilized by the *Activity Management* service. The *Interface Emitter* generates HPS interfaces depending on application scenario, for example, human interactions using HPS or human interactions in processes.

4 HPS Interface Transformation and Generation

The design process and methodology presented in this paper has to be supported by a set of tools and models. We start with the definition of the process, which

[2] W3C Markup Forms: http://www.w3.org/MarkUp/Forms/

allows users to define services without having to understand Web services technologies (depicted in Fig.3). Several papers focus on automatic GUI generation based on WSDL descriptions [7,8]. However, these works assume that the WSDL description of a service already exists and simply needs to be parsed and mapped into some GUI language/representation.

4.1 Design Process

We define a process allowing users to create an activity model serving as the input for automatic generation of service artifacts. The HPS framework provides *User Controls* and a corresponding *Meta Model* which enable people to design their *Activity Model* and *HPS Interfaces*. We discuss interface mappings and the meta model in Sec. 4.2.

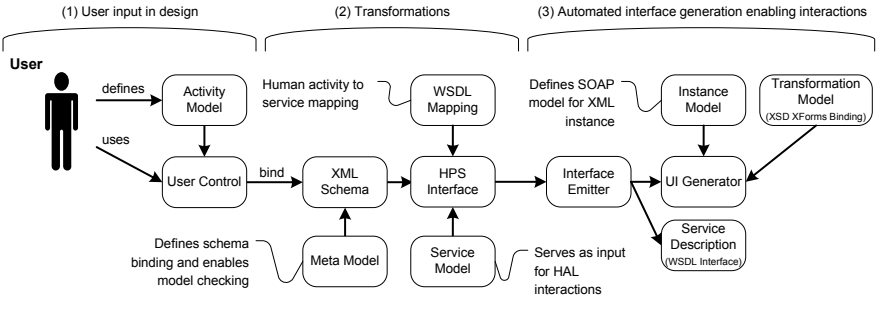

Fig. 3. Conceptual approach and interface design

Step 1 in the design: users define their *Activity Model* using controls (a simple example is shown in Fig. 5).

Step 2 comprises automatic transformations of the user's input into XML artifacts: i) invoke the `XSD Transformer` translating the activity model into XML schemes (i.e., *Schema Binding*) using definitions and constraints defined in the *Meta Model*. An example of such constraints and mappings is given in Table 1. Currently we express model mappings in XML schemes. ii) *HPS Interfaces* are created by associating activity types with the *Service Model*, defining how activity types (and human actions) are mapped into services definitions (WSDL). The mapping of an *HPS Interface* into WSDL is the binding of activity type definitions and actions with the HPS Access Layer (HAL, see [1]). At run-time, HAL acts as a proxy service dispatching requests by performing security checks, routing, message transformations (if needed), and persistency management of messages (i.e., saving request/response messages in XML collections).

Step 3 comprises the automated generation of interfaces at run-time. The *Interface Emitter* generates: i) interfaces allowing processes to interact with HPSs by generating WSDLs. Thus, HPSs may be included in (software) process by defining human activities in the process definition, which are enacted as HPS actions

Table 1. Excerpt interface mapping

Model & Binding	Description
XSD	`<xs:choice/>`
Form	`<xf:select1 ref="" appearance="$model"/>` with appearance as $model parameter ("full" or "compact")
Restriction/Model	`//*[@value='Choice']/prop[@name='type']/@value` and `<xs:choice minOccurs="$model" maxOccurs="$model">` with $model as parameters (minOccurs and maxOccurs being "1")

(interaction through HAL). GUIs are generated automatically by transforming WSDLs and XSDs into XML forms using the `Forms API`.

4.2 Interface Mappings

Meta Models define interface mappings of XML schemes into XForms. Table 1 shows an exemplary mapping of an XML type into an XForms representation, i.e., (1) *XSD* type, (2) *Form* model, and (3) mappings and *Restriction/Model* as defined in the meta-model.

The above mappings support transformations into a forms representations. We continue our discussion of HPS interfaces by showing a concrete XML example of a WSDL description. We start with type definitions in Listing 1.

Of course, no user input is needed for mapping activities onto services or to create WSDL descriptions. This is automatically performed by services in the framework. Listing 1 shows `GenericResource`, `ReviewRequest` type definitions.

Listing 1. Review-activity types example.

```
<xsd:schema targetNamespace="http://services.myhps.org/reviewservice">
  <xsd:complexType name="GenericResource">
    <xsd:sequence>
       <xsd:element name="Location" type="xsd:anyURI" />
       <xsd:element name="Expires" type="xsd:dateTime" />
    <xsd:sequence>
  </xsd:complexType>
  <xsd:element name="ReviewRequest" type="Request" />
  <xsd:complexType name="Request">
     <xsd:sequence>
       <xsd:element name="ReviewResource" type="GenericResource" />
       <xsd:element name="Comments" type="xsd:string" />
     </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="AckReviewRequest" type="xsd:string" />
  <xsd:element name="GetReviewReply" type="xsd:string" />
  <xsd:element name="ReviewReply" type="Reply" />
</xsd:schema>
```

Such definitions can be created using tools, as we shall discuss later. In this simplified example, the activity to be performed by a human is *review* comprising resources, the actual request, and the reply, which is complex XML data structure (abbreviated in this example).

Finally, Listing 2 shows an excerpt of a WSDL representing a review HPS.

Listing 2. HPS WSDL example.

```
<wsdl:message name="GetReview">
  <wsdl:part name="part1" element="ReviewRequest" />
</wsdl:message>
<wsdl:message name="AckReviewRequest">
  <wsdl:part name="part1" element="AckReviewRequest" />
</wsdl:message>
<wsdl:portType name="HPSReviewPortType">
 <wsdl:operation name="GetReview">
   <wsdl:input xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
         message="GetReview" wsaw:Action="urn:GetReview" >
   </wsdl:input>
   <wsdl:output message="AckReviewRequest" />
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HALSOAPBinding" type="HPSReviewPortType">
 <soap:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http" />
</wsdl:binding>
```

Notice, `PortType` (i.e., the interface) for all interactions is the HPS access layer. At run-time, the access layer extracts and routes messages to the demanded HPS. Since every interaction is entirely asynchronous, interactions (session) identifier are automatically generated by the access layer (e.g., `AckReviewRequest`).

5 Recommendations for HPS Design

In the spirit of collaborative tagging systems, we perform ranking and recommendations based on tagged resources. An entry is a triple `(user, resource, {tags})`. We perform matching of relevant users who provide a particular HPS of interest. The matching function is based on applied tags. To obtain rankings, we measure the similarity of the user's preferences with activities and actions of matching users, i.e., those users already offering HPSs to find definitions of services that can be reused.

Algorithm 1 outlines our ranking approach. We assume several functions. For example, we calculate the similarity of preferences/activities as the Pearson correlation coefficient. Furthermore, we calculate the frequency of tags using additive smoothing, a simple yet effect method to account for missing/misplaced tags.

Algorithm 1. Recommendation algorithm comparing the preferences of a person v searching for existing HPSs and artifacts.

Require: U_{tag} is a set of users that applied the demanded set of tags. S is a vector holding the scores of services by users $\in U_{tag}$ with similar interests.
Require: We determine the correlation between users as:
$correl(u_1, u_2) = \sum (u_1 - \overline{u_2})(u_1 - \overline{u_2})/(N * stdev(u_1)stdev(u_2))$
for each user $u \in \mathcal{U}$ **do**
  for each $tag \in AT$ **do**
    /* Get the frequency for a specific tag used by u. */
    $\Omega[tag] \leftarrow getFrequency(u, tag)$
  end for
  /* Assign the weight using the smoothing factor $0 < \gamma < 1$. */
  $w_{(u,tag)} \leftarrow \frac{\Omega[tag]+\gamma}{\sum_{f_{tag} \in \Omega} f_{tag} + \gamma}$
  /* If any of u's interactions/HPSs contain tag. */
  Add u to U_{tag}
end for
for each user $u \in U_{tag}$ **do**
  /* Calculate correlation between u and v. */
  $\phi \leftarrow correl(w_u, w_v)$
  $S[u] \leftarrow p(u) * score(\phi)$
  /* $p(u)$ is personalization vector to assign additional preferences. */
end for
/* Sort preferred users by decreasing score and truncate S. */
return the top-k list of services by users

The input of Algorithm 1 is a query vector and the preferences of the user initiating the design process. By matching existing HPSs, we determine the initial set U_{tag}. We determine the score of a particular user by mapping the correlation factor ϕ into a linear function $score(\phi) = \frac{\phi+1}{2}$ since $(-1 \leq \phi \leq 1)$ would give us negative scores if there is no correlation between users. Additional personalization can be performed by assigning preferences (i.e., $p(u)$) for particular users. The output of the recommendation algorithm is a ranked list of services S.

6 Implementation

In this section we show the tools to manage HPSs, artifacts, etc. supporting lookup of services. The first tool illustrates how a user can search for existing services. The first screenshot in Fig. 4 shows the result of a user query. In this example we show the same HPS WSDL as discussed in previous sections (Sec. 4.2). If none of the existing HPSs fits the desired activities, users have the ability to create new artifacts and activities which are then mapped into HPSs.

The framework provides a set of designer tools which can be used to define control elements, options, and definitions of artifacts. A screenshot of a control is given in Fig. 5, allowing users to define complex data structures. This tool is used

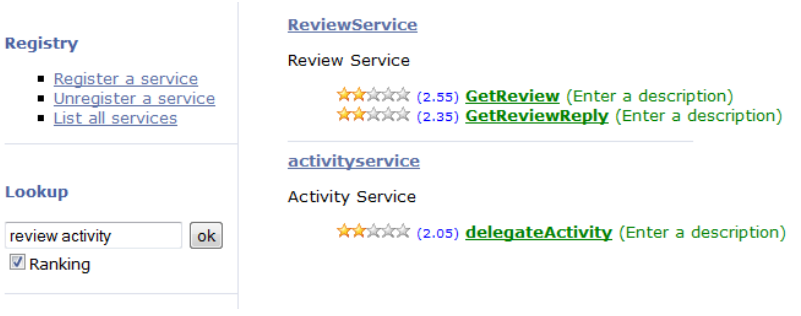

Fig. 4. Registry maintaining HPSs and service artifacts: users can search for services, which are displayed in a ranked list

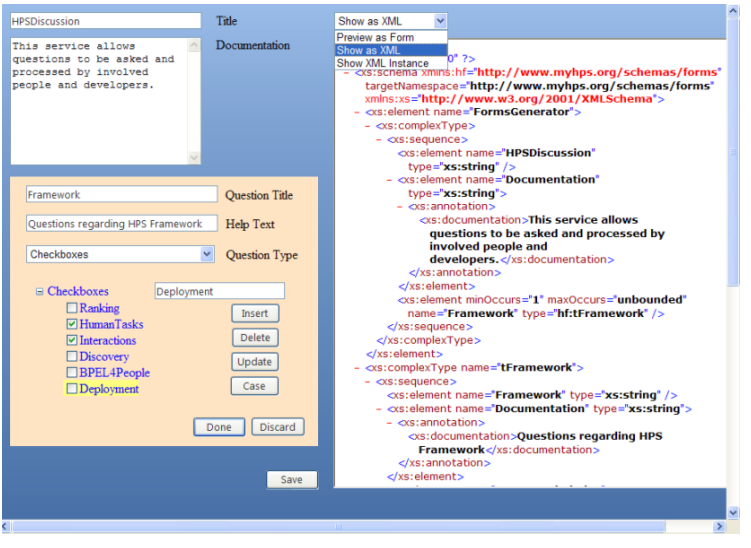

Fig. 5. Example control allowing users to define activities and complex data structures

in **Step 1** as defined by the design process in Sec. 4.1. The prototype version of this tools automatically translated user input into low-level XML artifacts such as schemes, instance documents (e.g., encapsulated in SOAP envelopes), WSDL descriptions and XML forms depending on the HPS use case. For example, interactions between humans in collaborations or using HPS in a process-centric scenario. `Show as XML` (for demo purposes) translates user defined elements into XML schemes, `Show as XML Instance` displays the associated XML schema instance document, and `Preview as Form` renders XForm presentation.

7 Related Work

The work presented in this paper focuses on a methodology and tools allowing humans to define service for various collaborations. We structure our discussion

into related work in the area of SOA, Web services, and process-centric collaboration. Second, we present related research in engineering methods in SOA. Third, we discuss tagging mechanisms and approaches for resource recommendations.

BPEL4People Task Model: BPEL4People defines human interactions in business processes via the human task specification [4]. A concrete implementation of BPEL4People as a service has been introduced in [9], but without supporting the design of services. In [10], the relation of various Web standards and resource patterns is discussed.

In contrast to above mentioned work, we describe a design approach allowing people to define services. While BPEL4People defines how developers can define human interactions in processes, related BPEL4People specifications do not describe how humans define services and how people manage interactions in SOA. The difference between the task model, such as defined by BPEL4People, and HPS is that tasks are usually defined for controlling interactions (e.g., start, end, deadline, etc.). HPS targets collaboration scenarios where users contribute their skills and expertise as services. Another point is that HPS reflects the composable nature of the Web, for example, reusing and composing services. However, we have not yet addressed compositions of HPSs.

Approaches for Interface Transformations and Generation: Our framework utilizes open standards such as WSDL, to describe HPS interfaces, and XForms to automatically generate user interfaces. GUI generation and mappings for WSDL has been presented in [7,8] and forms generators such as IBM's XML Forms Generator[3] for the Eclipse environment are available. In this paper we not only focus on generating GUIs based on WSDL, but also the mapping of human activities into HPS interfaces and generation of presentations (WSDL or XForms).

Tagging and Resource Recommendations: Recently, models for collaborative tagging have been presented [11] and personalized recommendations-based tagging models introduced in [12]. In [13], the authors present an evaluation of tag recommendations in folksonomies using collaborative filtering methods and an algorithm called FolkRank. Similarly, we follow a tag-based recommendation approach. However, we propose *activity tagging* to express expertise of users. Based on these tags, we implemented a collaborative filtering algorithm for recommendation of HPSs.

8 Conclusion and Future Work

The methodology and tools presented in this work offer support in the design of HPSs for users without having to implement code. The framework hosts a Web 2.0 portal, implemented in ASP.NET AJAX and C# APIs, allowing users to search for service artifacts (interfaces already provided by other users) and to design new services. The next steps include further implementation of XForm specifications and deployment of a rendering run-time on mobile devices. Also, we are working on composition aspects of HPSs and usability improvements of user tools.

[3] http://www.alphaworks.ibm.com/tech/xfg

References

1. Schall, D., Truong, H.L., Dustdar, S.: Unifying Human and Software Services in Web-Scale Collaborations. IEEE Internet Computing 12(3), 62–68 (2008)
2. Schall, D., Truong, H.L., Dustdar, S.: The Human-provided Services Framework. In: IEEE 2008 Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2008), Crystal City, Washington, D.C., USA. IEEE Computer Society, Los Alamitos (2008)
3. Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M.: WS-BPEL Extension for People (BPEL4People), Version 1.0 (2007)
4. Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M.: Web Services Human Task (WS-HumanTask), Version 1.0 (2007)
5. Golder, S., Huberman, B.A.: Usage patterns of collaborative tagging systems. Journal of Information Science 32(2), 198–208 (2006)
6. Dustdar, S.: Caramba a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. Distrib. Parallel Databases 15(1), 45–66 (2004)
7. Song, K., Lee, K.H.: An Automated Generation of XForms Interfaces for Web Services. Web Services. In: Song, K., Lee, K.H. (eds.) IEEE International Conference on ICWS 2007, pp. 856–863, July 9-13 (2007)
8. Kassoff, M., Kato, D., Mohsin, W.: Creating GUIs for Web Services. IEEE Internet Computing 07(5), 66–73 (2003)
9. Thomas, J., Paci, F., Bertino, E., Eugster, P.: User Tasks and Access Control over Web Services. In: Int. conf. on Web Services (ICWS 2007), Salt Lake City, USA, pp. 60–69. IEEE Computer Society, Los Alamitos (2007)
10. Russell, N., Van Der Aalst, W.M.P.: Evaluation of the bpel4people and ws-humantask extensions to ws-bpel 2.0 using the workflow resource patterns. Technical report, BPM Center Brisbane/Eindhoven (2007)
11. Cattuto, C., Loreto, V., Pietronero, L.: Semiotic dynamics and collaborative tagging. PNAS 104(5), 1461–1464 (2007)
12. Byde, A., Wan, H., Cayzer, S.: Personalized tag recommendations via tagging and content-based similarity metrics. In: Proceedings of the International Conference on Weblogs and Social Media (2007)
13. Jäschke, R., Marinho, L.B., Hotho, A., Schmidt-Thieme, L., Stumme, G.: Tag recommendations in folksonomies. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) PKDD 2007. LNCS, vol. 4702, pp. 506–514. Springer, Heidelberg (2007)