The Human-provided Services Framework *

Daniel Schall, Hong-Linh Truong, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology, Austria
schall|truong|dustdar@infosys.tuwien.ac.at

Abstract

The collaboration landscape evolves rapidly by allowing people to participate in ad-hoc and process-centric collaborations. Thus, it is important to support humans in managing highly dynamic and complex interactions. The problem currently with managing interactions is that humans are unable to specify different interaction interfaces for various collaborations, nor able to indicate their availability to participate in collaborations. This paper introduces the Human-provided Services (HpS) framework, which allows users to provide services based on their skills and expertise. Such services can be used by human actors and software services in both ad-hoc and process-centric collaborations. With the HpS framework, people can offer multiple services and manage complex interactions, while requesters can find the right experts and available users for performing specific tasks. In this paper, we present the HpS middleware, which is the core of the HpS framework. We show how HpS services can be used in Web-scale ad-hoc collaboration scenarios.

1 Introduction

Today's collaboration landscape has changed by allowing a large number of users to communicate and collaborate using Web-based platforms and messaging tools. Users collaborate with each other by sharing content that is made available on the Web. Also, collaborations within organizations are no longer closed ecosystems as collaborations and interactions span multiple organizations or business units that are scattered around the globe. However, it becomes increasingly challenging to manage collaborations that involve a number of people and comprise a large set of exchanged messages. In addition, users demand access to collaboration resources using pervasive devices in an always-on fashion.

To address these challenges, collaboration platforms must support the user in managing complex interactions that span multiple organizations, and hide the complexity due to different message formats and divers types of collaboration resources, and furthermore be able to support different devices. This paper introduces the Human-provided Services (HpS) framework that lets users publish their capabilities and skills as services. Using the HpS framework, users are able to define and provide services for different collaborations. HpS allows users to control their interactions beyond the simple exchange of messages by defining multiple service interfaces and interaction rules to manage complex interactions. The novelty of HpS is that collaborations take place in a service-oriented framework, thus enabling a dynamic mix of human- and software services. User- and service related information are maintained in a service registry, which allows HpSs to be discovered by both human collaborators and software (processes) services. Thus, HpS allows business processes, that require human input or intervention, to interact with humans using standardized Web services protocols, making the HpS framework a versatile collaboration and interaction framework.

1.1 Approach

Our approach is to build an HpS middleware platform that integrates Web technologies and Web services with the goal of providing a framework to enable humans to publish services, thereby allowing humans and software to find and interact with HpS users. Figure 1 shows our approach, which comprises the specification and deployment of services and service discovery and interactions with HpS services. To this end, the features supported by the HpS framework must be:

Ability to define services. Anyone has to be able to define services and corresponding interfaces, or simply reference or copy an existing interface and reuse or modify it. In step 1 users specify profile information and define service interfaces.

Specification of interactions. Users must be able to specify their interaction protocols. Customized protocols allow interactions to be managed in a given context, that is in a collaboration through services.

*This work has been partially supported by inContext (FP6-034718).

User-centric service publishing/provisioning. Encompasses the ability to easily publish and interact with services. In step 2 users deploy and register personal services.

Discovery and interactions with users/processes. Processes and humans actors must be able to discover HpSs. HpS simplifies interactions with user-provided services by abstracting from service location and deployment. In step 3 requesters discover humans and services and interact with the selected human and services through the HpS middleware.

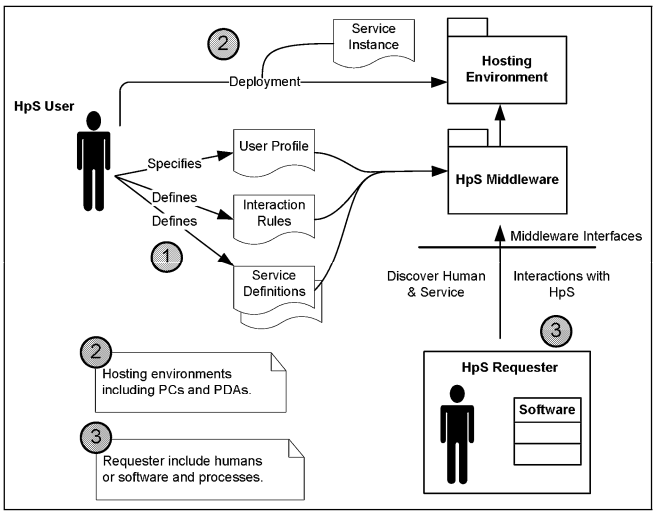

Figure 1. The HpS approach.

1.2 Contributions

Our contributions center around the definition of a novel framework that uses Web services in interactions and collaborations between people or people and software services. This paper discusses the design, implementation, and evaluation of the HpS middleware platform. The goal of this paper is to provide an insight on the various components and services provided by the middleware. Out-of-scope are legal or privacy issues as well as security issues.

Structure of the Paper: Interactions models applicable to human collaboration, ranging from ad-hoc to process-cenctric, are presented in Section 3. The HpS system architecture is detailed in Section 4, followed by a discussion on implementation aspects in Section 5. Section 6 describes how to use the HpS framework in ad-hoc collaboration scenarios.

2 Related Work

The work in this paper tackles several issues related to services on the Web and human computation. In the following we discuss significant related work in those areas.

Human computation is a technique to let humans solve tasks, which cannot be solved by computers (see [4] for an overview). An application of genetic algorithms has been presented in [6]. The computer asks a person or a large number of people to solve a problem and then collects their solutions (e.g., see games that matter [11]). Human computation applications can be realized in the HpS framework as people are able to provide user-defined services. Additionally, the HpS framework allows users to manage their interactions. Web-based platforms inspired by human computation include for example Yahoo! Answers[1] [9] and Amazon Mechanical Turk[2], which employ *human tasks* that are claimed and processed by users. There are several limitations which cannot be addressed by these platforms, but by HpS: (1) how to manage interactions, (2) how to find the right person (expert), (3) how can users define their availability to participate in collaborations. Recently, specifications have been released which allow processes (i.e., BPEL) to be extended with human interactions, defined in *WS-HumanTask* specification [1]. Additionally, work presented in [10] aimed at integrating humans into processes. The HpS framework can be used in such process-centric collaborations as well (e.g., *human task* in process). However, the HpS framework allows user-define services for ad-hoc and process-centric collaborations, and also allows humans (services) to be discovered. *Expert-finder* systems [2] commonly utilize semantic technologies to express users' expertise and skills as ontologies. In the HpS framework, we focus on interactions between humans and software using Web services technologies. However, the HpS framework can be extended by using semantic technologies, for example, to express skills and social relations using ontologies.

3 Interaction Models

The interaction models in collaboration range from ad-hoc (informal) to predefined formalized process models (see [3], [8]). Table 1 gives an overview of these different models. In the following we discuss concepts used to control interactions.

In this paper we show how the HpS framework can be used in *ad-hoc* collaboration scenarios. The only requirement for users is to define *human activities* (at design time), which can be automatically mapped to specific Web services and *actions*. During the actual collaboration (runtime), requests to perform certain activities are being sent to HpSs as XML documents that parameterize the request. In contrast to workflow-based systems, interactions need not comprise predefined process models. In HpS, there is distinction between a task announcement and an *interaction control* task, both using the `Human Task` structure.

[1] http://answers.yahoo.com/
[2] http://www.mturk.com/

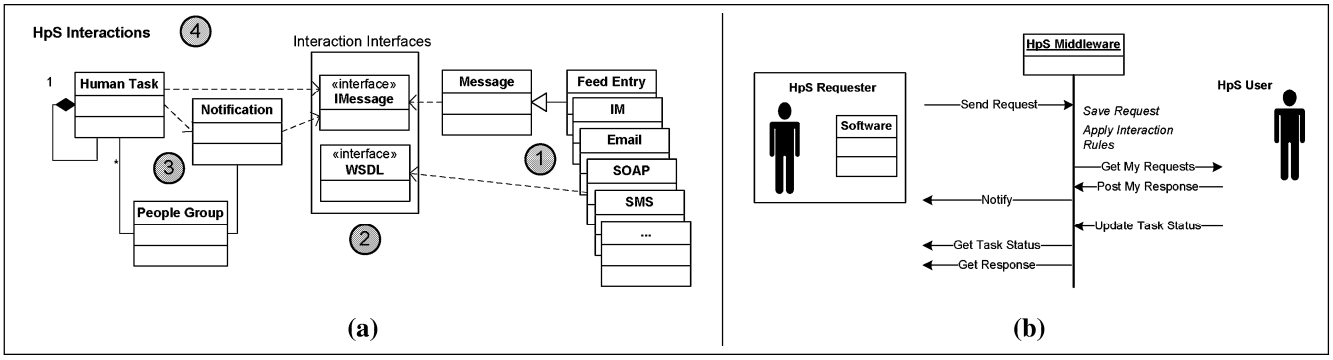

Figure 2. (a) Conceptual model HpS interactions. (b) Example interaction flow.

Ad-hoc	Interactions are ad-hoc if there is no predefined control flow associated with an interaction. For example, interactions between requesters and HpS users simply take place by exchanging messages.
State-awareness	Tasks can be used to control the status of an interaction. Requesters have the ability to impose certain constraints on tasks such as start-time (when should users start processing tasks) or deadlines (maximum time when tasks have to be finished).
Process-centric	Process-centric collaboration can be established by defining *interaction rules*. Tasks can be split into sub-tasks and forwarded to other people. Multiple HpSs could be potentially involved in interactions to solve complex problems.

Table 1. Interaction models in human collaboration.

Task announcements. Requesters have the ability to create a `Human Task` and to specify the number of available tasks. Tasks can be linked to HpS service-categories to express which service (i.e., which expert) is needed to process the given task. This case is indicated by the link between `Human Task` and the *Interaction Interfaces* in Figure 2 (a) (Listing 1 shows an actual XML example of task announcements). Linking tasks announcements to services is accomplished by *tagging* task descriptions with keywords. Tasks can be linked to a logical `People Group` to specify conditions associated with the users that should be able to claim and process the task (e.g., user groups in an organization's human resources directory).

Interaction control tasks. If tasks are used in interactions, defined by using `Human Tasks` in Figure 2 (a), requesters are aware of the state of a given request (e.g., *accepted*, *inprogress*, or *completed*). Task-state information can be retrieved via pull mechanisms or, alternatively, various actions can be automatically triggered such as sending `Notifications` upon state changes.

Interactions. HpS interactions comprise a multitude of `Messages` in different formats (e.g., indicated as `Email` or `SOAP` messages in Figure 2 (a)). In addition, interactions generally comprise notifications, tasks, and people/services that are involved in an interaction. Discussions on complex interaction flows are not in the scope of this paper.

3.1 HpS Interactions

In HpS, Web services are used to define interaction interfaces to humans. Typical interaction patterns found in the Web services domain such as the synchronous exchange of messages are not sufficient for modeling human interactions. Therefore, we introduce a new human-based service interaction model, allowing users to deal with requests in, for example, offline mode or using different devices to process requests. Since today's collaboration landscape increasingly shifts toward pervasive collaboration and interactions, a system supporting HpS services must give users the flexibility to deploy user-defined services on a variety of devices (e.g., mobiles). Such devices are not always online or connected to the network. Thus, the HpS framework allows requests to be saved and retrieved whenever the users are available. An exemplary interaction flow is shown in Figure 2 (b). Indeed, the number of actors involved in an interaction can be greater than two and multiple tasks can be defined. As mentioned before and like in most collaboration systems, interactions encompass a large number of messages in various formats (see *HpS FS* in Section 4, an XML-based file system, which has been designed to accommodate those messages). Requests are sent toward the HpS middleware, which allows messages to be exchanged either synchronously or asynchronously. Requests can be forwarded to the corresponding user instantaneously (e.g., users is available) or saved in an XML-based repository.

4 HpS Framework

4.1 Middleware Platform

HpS allows a seamless integration of human actors in service-oriented systems and collaborations that may require human input in process-centric collaborations. However, in contrast to existing work and specifications as WS-HumanTask [1], people have the ability to define a set of user-provided services that can be used in ad-hoc collaborations and interactions between humans. Figure 3 depicts the HpS middleware platform.

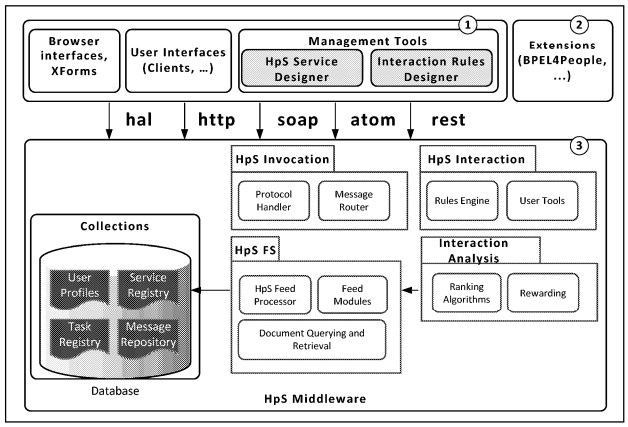

Figure 3. HpS middleware platform and architecture.

HpS Middleware Interfaces. The middleware offers interfaces for discovery of services and interactions with HpS users. The `hal` interface (**H**pS **A**ccess **L**ayer) is a REST interface that routes requests in various formats to the corresponding user/service. An `atom` interface can be used to discover services by retrieving Atom feeds[3] that contain service related information. Additionally, the service lookup can be performed using the `soap` interface, facilitating the integration of the HpS framework with other Web services-based platforms.

HpS Invocation. Processes requests and sends messages in the appropriate format toward the HpS user. By specifying user or group identifiers (e.g., email address or distribution lists) and service name, HpSs can be located and and an interaction initiated by directing the request toward the access layer (`hal`). Every request is then passed through the validation phase in which an authorization check is performed. The user can specify white/black lists and routing and interaction rules. White/black lists are used, for example, to prevent certain users from interacting with HpS services. The `hal` interface routes service requests to the desired service, thereby abstracting from actual service endpoints and service location. Requests can be delivered to the corresponding service immediately, or through an offline interaction as illustrated in Figure 2 (b). In the latter case, requests are saved in the *Message Repository*.

HpS FS. Manages a set of collections of diverse type of XML-based information. Collections in HpS are conceptually designed as a native XML-based file system that allows artifacts, messages, tasks, user and service related information to be managed and retrieved. An XML database stores and manages XML collections. XML documents can be retrieved by using XQuery to filter and aggregate information.

HpS Interaction Component. HpS users may define a set of interaction rules to manage their collaborations (based on a set of provided services). The HpS framework does not mandate which rules users can specify. The framework allows users to specify rule languages, which can be mapped into the *Rules Engine*. Therefore, rules can be tailored to the needs of specific domains by creating Domain Specific Languages (DSL) to describe interaction models. For example, see [7] for related work in domain interaction models.

Interaction Analysis. Human and service interactions are recorded, archived, and analyzed. This information is used for ranking services based on a set of human-metrics such as task processing performance, availability, or expertise-rank based on the interaction network structure. Ranking algorithm help to recommend the most relevant HpS and the right expert to perform a given task/request.

4.2 Data Collections

Collections are managed by the *HpS FS* as XML documents. These collections can be manipulated by using the Atom Publishing Protocol, e.g., the standard protocol model includes `get, post, put, delete, head`, to allow resources/messages to be retrieved and updated.

User Profile and Metrics. Profiles are used to manage and store user related information, described in XML. HpS users can specify basic information or simply import personal data that is already available (e.g., vCard format). We categorize *User Profile* information in *hard-facts* and *soft-facts*. Hard-facts comprise information typically found in resumes such as education, employment history including organizational information and position held by the user, and professional activities. Soft-facts are represented as competencies. A competency comprises *weights* (skill level of a user), *classification* (description of area or link to taxonomy), and *evidence* (external sources acting as references or recommendations). Soft-facts can be automatically generated by the HpS middleware based on users' activities to indicate a user's expertise or skill level.

Service Registry. The registry maintains a number of XML documents describing services and allowing human

[3] Atom Syndication Format - RFC 4287.

and software services to be discovered. This information includes a set of service definitions, the list of available services, and information regarding a specific service provided by a user. A detailed discussion on these XML collections is given in Section 6.

Task Registry. Manages `Human Tasks` that can be either public tasks, used to advertise the need for HpS users to work on tasks, or private tasks that are added to interactions as control elements. Public tasks are associated with an interaction upon claiming and processing tasks. In addition, tasks can be added to an interaction without defining public tasks beforehand.

```
<feed xmlns="http://www.w3.org/2005/Atom"
     xmlns:ht="http://www.myhps.org/schemas/task.xsd">
 <title>HpS Tasks</title>
 <updated>2007-09-24T18:30:02Z</updated>
 <id>urn:uuid:63a99c80-d399-12d9-b93C-0003939e0a</id>
 <entry>
  <title>HpS Public Tasks</title>
  <updated>2007-09-19T18:30:02Z</updated>
  <id>urn:uuid:1223c696-cfb8-4ebb-aaaa-80da344ea6</id>
  <link href=".../atom?tasks" rel="self"
        type="application/atom+xml"/>
  <category term="humanrevieweddata"
          label-"documentreview"/>
   <!-- tags applied to tasks -->
  <link rel="alternate" type="application/atom+xml"
        href="/services/reviewservice.xml"/>
  <description>
   <![CDATA[ basic task rendering definitions ]]>
  </description>
  <ht:task>
   <!-- task instance data -->
  </ht:task>
 </entry>
</feed>
```

Listing 1. Human task-to-service mapping.

Listing 1 shows an example of a task announcement. The announcement contains a list of public tasks that reference the type of HpS service that should process available tasks. In this example, task related information is encapsulated as `<ht:task>` elements in Atom feed entries. The `category` element can be used to add tags to `Human Tasks`.

5 Implementation

The HpS middleware comprises the implementation of the XML based file system (*HpS FS*) and XQuery-based filtering and retrieval of XML documents through the implementation of the XQuery API for Java (XQJ). Furthermore, the `atom` interface, that supports the Atom Protocol Model to manipulate resources, and the `hal` interface to support complex interactions with HpSs and dispatching of messages are implemented. The *HpS Interaction* component is currently under development. We utilize the JBoss Drools[4] system which supports graphical Web-based editing tools based on which HpS users can define interaction rules. User interfaces (e.g., Web browser clients) allow services to be discovered and enable service requesters to interact with HpS users. At the implementation level, we use a set of state-of-the-art Web 2.0 technologies such as AJAX to enable asynchronous interactions between the client and the middleware. In addition, context information can be used in the service discovery process, for example, by filtering XML documents based on users' availability.

Service Deployment. Services are deployed in the hosting environment, for example PCs, Smartphones or PDAs. This deployment strategy allows the HpS framework to scale to a large audience without being restricted to any specific technology. The framework supports the option to deploy services in a *platform independent* manner.

In our experiments, we have used an Apache Axis2 Web services environment embedded in an Equinox OSGi[5] container. This solution is well suited for PCs, but not for mobile devices such as Smartphones. For resource constraint devices, a combination of OSGi technology and SOAP servers with small footprint can be used. Specifically for the Windows platform, the Windows Communication Foundation (WCF) can be used to develop Web services for Windows XP and Vista. We have developed SOAP and REST (XML and JSON) based services using the API provided by WCF.

User Interface Aspects. In the service discovery phase, the requester (client) receives an XML document from the middleware (registry). In Listing 2 and Listing 3 we see an example where user interfaces are represented using XForms technologies[6]. XForms are automatically generated by the HpS framework based on WSDL descriptions (see Listing 3 *category* and *term* specification).

Listing 2 shows the model specifying SOAP as the interaction message format and the HpS middleware access layer as the submission target.

```
<xforms:model id="model-envelope">
<xforms:instance id="instance-envelope" src="review.xml"/>
 <xforms:submission id="submit-envelope"
    action="{HPS Access Layer}" method="post"
    mediatype="text/xml" replace="instance">
  <xforms:toggle ev:event="xforms-submit-done"
    case="response"/>
 </xforms:submission>
</xforms:model>
```

Listing 2. SOAP interaction model.

Listing 3 shows the actual interface representation that allows human requesters insert the request parameters and also request messages to be rendered on various devices. The `switch/case` construct defines the behavior of the form – *request* and *response* representation. These forms are platform and device independent and can be displayed

[4] http://labs.jboss.com/drools/

[5] http://www.osgi.org/osgi_technology/

[6] http://www.w3.org/MarkUp/Forms/

on, for example, mobile devices or in standard Web browser using a suitable forms plugin.

```
<entry>
 <category term="/services/reviewservice#WSDL" />
 <content type="xhtml">
  <xforms:switch>
   <xforms:case id="request" selected="true">
    <xforms:label>Input definitions</xforms:label>...
    <xforms:submit submission="submit-envelope">
     <xforms:message ev:event="xforms-submit-done"
        ev:observer="submit-envelope" >...
     </xforms:message>
    </xforms:submit>
   </xforms:case>
   <xforms:case id="response" />
    <xforms:output ref="..." model="model-envelope">
     <xforms:label>Submission output.</xforms:label>
    </xforms:output>
  </xforms:switch>
 </content>
</entry>
```

Listing 3. Snippet request input form.

The actual instance model – i.e., the request message – is an XML document (SOAP envelope) as defined in Listing 2, which is dispatched by `hal` upon submission (`submit-envelope`).

6 Using the HpS Framework in Ad-hoc Collaborations

We discuss the required steps to publish HpSs and show how requesters discover and interact with personal services using middleware interfaces for HpS interactions. However, due to space limits, process-centric collaboration scenarios and interactions with (business) processes are not addressed in this paper. There are three phases in ad-hoc based collaborations:

Service Definition. The user specifies messages and collaborative activities (at a high level) using the *Management Tools* provided by the middleware (see Figure 3). Based on messages and activities, the middleware automatically generates low-level HpS interfaces using interface description languages such as WSDL or WADL. These descriptions are deployed as XML documents in the *Service Registry*.

Service Discovery. Requesters discover human and software services by browsing/filtering XML documents that contain the relevant users/services.

HpS Interaction. Requesters interact with services by issuing requests toward the middleware. Requests can be converted by the *Protocol Handler* to match different service interface types. For example, messages that are encoded in JSON notation can be converted to XML messages, and back. However, the *Protocol Handler* does not support message conversion from, for example, SOAP/XML to REST/JSON notation. Messages are being routed by the *Message Router* to the corresponding user-provided service or saved in the XML *Message Repository*. The actual interaction with the HpS – receiving and processing the request – can take place depending on the user's context (e.g., availability or also specified interaction rules).

6.1 Defining Service Interfaces

A HpS interface definition is an XML document that contains four entries (see Figure 4).

Addressing information of personal services to describe how to interact with a particular user providing the service. This information is used by requesters to locate and interact with personal services using the `hal` interface. Figure 4 (3) shows the addressing information entry. The Web Services Resource Catalog (WS-RC) meta endpoint definition[7] is used to express addressing information of personal services. WS-RC endpoint descriptions can be annotated using `mex` elements to describe meta data, for example, taxonomies, that are applicable to all personal services of the same service type; regardless of the specific underlying protocol (SOAP or REST). The `ParameterMap` element defines tokens in the service address, for example, a `uri` that is replaced at run-time by HpS user information (e.g., user id or Email address). The entry in Figure 4 (1.a) shows an excerpt of the WSDL definition of a HpS, which contains a link to the WSDL file and a meta data section defining the service interface (i.e., available *human activities*).

Service interface definitions shown in Figure 4 (1.a), (1.b), and (1.c). Entry (1.a) shows a WSDL interface definition encapsulated in an Atom feed entry. Entry (1.b) and (1.c) show REST interface definitions using the Web Application Description Language (WADL) [5]. (1.b) denotes the interface that defines messages in XML format (full entry has been omitted) and (1.c) shows an entry as REST/JSON service entry defined in WADL.

The technology choice depends on the specific application domain of user-provided services. At this point, the HpS middleware supports formats including SOAP/XML or REST/XML and the corresponding interface descriptions, which can be annotated with human-related information. As an example, Figure 4 (1.b) shows the definition of a REST HpS interface that defines the usage of JSON as the message format. This technology choice facilitates HpS service-interactions in Web browser-based client environments. A request can be created by using Javascript to issue JSON-requests toward the HpS middleware.

6.2 XML Collections of Services

In this particular scenario, shown in Figure 4, requesters are able to retrieve a list of services encoded as Atom feeds. Feeds have been designed for access of (and subscriptions

[7] Namespaces have been abbreviated for readability.

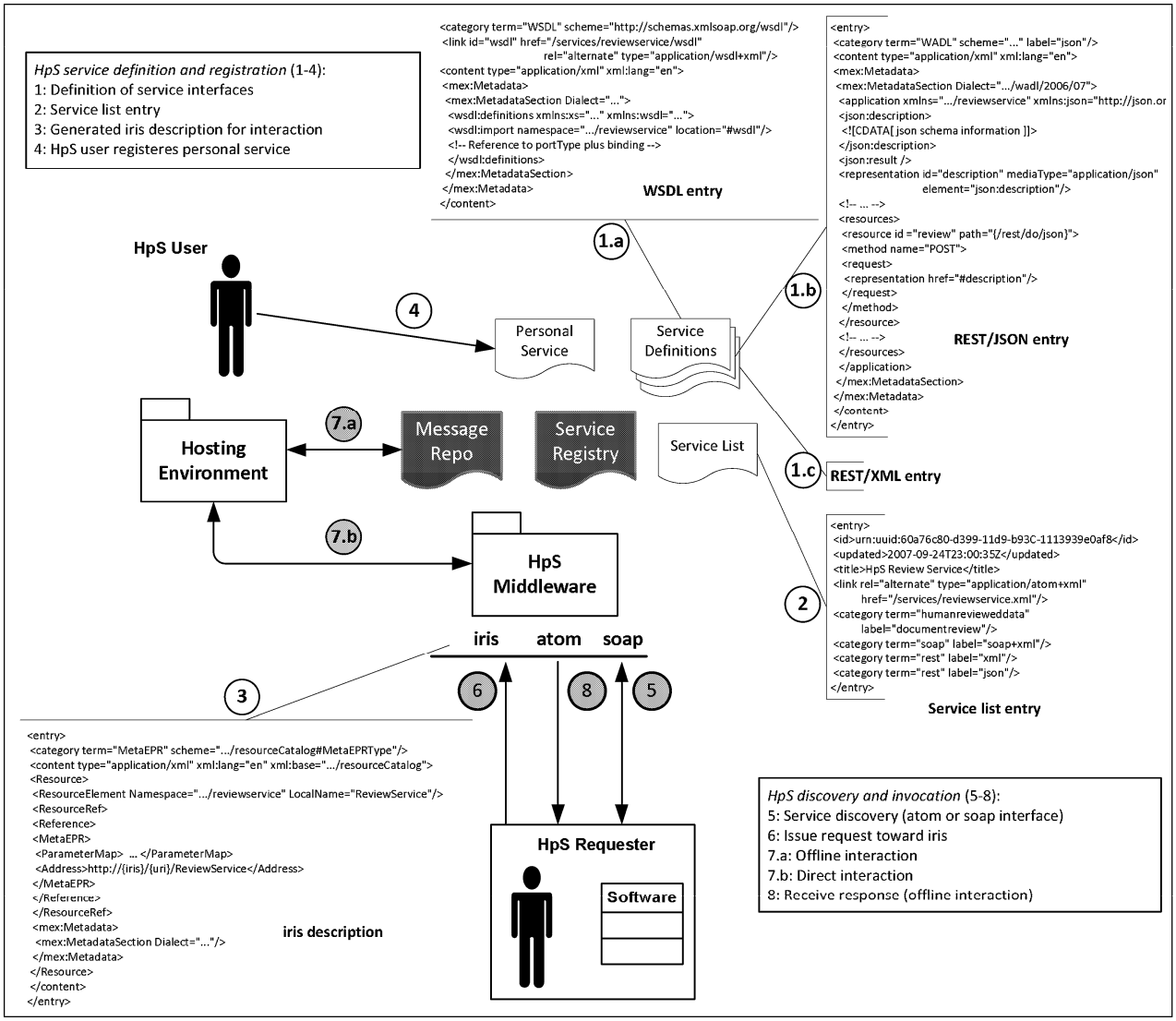

Figure 4. HpS discovery and interaction.

to) content, which is frequently being updated. Thus, requesters can subscribe to different categories of HpSs; content which changes frequently as HpSs rely upon the availability of human actors. `Category` elements describe the type of available service interaction models (see Figure 4 (2)). Note that, for scalability reasons, XML collections of services can be created for specific categories, which can be distributed and hosted by different *Service Registries*. In addition, multiple copies of service collections can be stored on different servers and replicated.

6.3 Personal Services

Personal services are user-defined services that can be provided by designing different services suitable for various collaborative activities. Example services are *"document review"* service, *"expert opinion"* service, or *"news reporter"* service, just to name a few. These services can be used in various collaboration scenarios and for complex problems, services can be composed by defining processes that span multiple users. However, the actors that should execute activities/tasks do not need to be determined beforehand as personal services can be discovered on demand and thereby following a service-oriented approach to collaboration. Given a requester's (consumer) query to discover services, the framework helps to find and select the most relevant personal services by (1) matching services that satisfy a given query, (2) filter services based on context (e.g., availability, workload, etc.) and (3) ranking each service based on a set of metrics.

An example description of a personal service is given in Listing 4. The XML description includes *user related* information such as name, address, and additional contact information, which can be specified by the user and/or selected from the user's profile. The *service model* defines how to contact the user. In the given example, the `category` element contains information regarding the supported models. Note, the `category` element references elements in the *Service Definitions* document (e.g., a user-defined *Review* service whose service interface is defined in WSDL). Since interactions with services travers the middleware platform, the *endpoint information*, encoded as `description` element, is used to forward requests to a service endpoint. However, this information is only used within the middleware platform and not exposed to potential service consumers.

```
<entry>
 <title>My HpS Review Service</title>
 <author>
  <name>Daniel Schall</name>
  <email>d.schall@infosys.tuwien.ac.at</email>
 </author>
 <updated>2007-09-24T18:30:02Z</updated>
 <id>urn:uuid:1223c696-cfb8-4ebb-aaaa-80da34efa6a</id>
 <link rel="alternate" title="EndpointReference"
       href="http://myHost/ReviewService"/>
 <category term="/services/reviewservice#WSDL"
       schema="http://schemas.xmlsoap.org/soap/http"/>
 <category term="/services/reviewservice#WADL"
       label="json"/>
 <description><![CDATA[
  <EndpointReference xmlns="..."></EndpointReference>]]>
 </description>
 <content type="xhtml"><!-- Classification -->
 </content>
 <geo:lat>48.19766</geo:lat>
 <geo:long>16.37146</geo:long>
</entry>
```

Listing 4. Personal service.

Personal services can be annotated with *expertise* information using various taxonomies. This information is encapsulated in `content` elements, which is valid for a *particular personal service*. Note, a user may want to provide different services and associate with each service a different set of skills or expertise. Nonetheless, expertise information can be specified in the user's profile, thereby being applicable to all personal services defined by a user. *Context information* such as location (e.g., `geo` tags) and user's availability status can be used to find and filter services in the discovery phase.

7 Conclusion and Future Work

The convergence of human- and software services in a single framework requires novel tools and platforms. By utilizing the HpS framework, users can manage their (complex) interactions, while requesters are able to find (discover) the right service. In this paper we focused on the architectural aspects of the HpS framework and implementation details of the middleware platform. However, we have not yet addressed legal or privacy issues, which are important to open the HpS framework to a larger audience. The next steps include a detailed performance and scalability analysis of the HpS framework. By gaining insights in performance aspects, the HpS framework will be able to accommodate a large number of users by federating multiple middleware platforms. We will conduct a more detailed user validation considering different ad-hoc and process-centric interaction models. Another important aspect of the HpS framework is ranking and recommending services. We are currently defining a set of HpS related metrics and algorithm to determine the most relevant service.

References

[1] M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. Knig, F. Leymann, R. Mller, G. Pfau, K. Plsser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller. Web Services Human Task (WS-HumanTask), Version 1.0., 2007.

[2] I. Becerra-Fernandez. Searching for experts on the Web: A review of contemporary expertise locator systems. *ACM Trans. Inter. Tech.*, 6(4):333–355, 2006.

[3] S. Dustdar. Caramba a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. *Distrib. Parallel Databases*, 15(1):45–66, 2004.

[4] C. Gentry, Z. Ramzan, and S. Stubblebine. Secure distributed human computation. In *EC '05: Proc. of the 6th ACM conference on Electronic commerce*, pages 155–164, New York, NY, USA, 2005. ACM.

[5] M. Hadley. Web Application Description Language (WADL). Technical report, Sun Microsystems, April 2006.

[6] A. Kosorukoff and D. E. Goldberg. Genetic Algorithms for Social Innovation and Creativity. Technical report, University of Illinois at Urbana-Champaign, 2001.

[7] M. Nussbaumer, P. Freudenstein, and M. Gaedke. Stakeholder Collaboration: From Conversation to Contribution. In *ICWE '06: Proc. of the 6th int. conference on Web engineering*, pages 117–118, New York, NY, USA, 2006. ACM Press.

[8] D. Schall, H.-L. Truong, and S. Dustdar. Unifying Human and Software Services in Web-Scale Collaborations. *IEEE Internet Computing*, 12(3):62–68, 2008.

[9] Q. Su, D. Pavlov, J.-H. Chow, and W. C. Baker. Internet-scale collection of human-reviewed data. In *WWW '07: Proc. of the 16th int. conference on World Wide Web*, pages 231–240, New York, NY, USA, 2007. ACM Press.

[10] J. Thomas, F. Paci, E. Bertino, and P. Eugster. User Tasks and Access Control over Web Services. In *Int. conf. on Web Services (ICWS'07)*, pages 60–69, Salt Lake City, USA, 2007. IEEE Computer Society.

[11] L. von Ahn. Games with a Purpose. *IEEE Computer*, 39(6):92–94, 2006.