Distrib Parallel Databases
DOI 10.1007/s10619-011-7081-z

A human-centric runtime framework for mixed service-oriented systems

Daniel Schall

Abstract A wide range of Web-based tools and socially-enhanced collaboration services have changed models of work. In today's collaboration systems, interactions typically span a number of people and services to work on joint tasks or to solve emerging problems. Finding the right collaboration partner in Web-based interactions remains challenging due to scale and the temporary nature of collaborations. We argue that humans need different ways to indicate their availability and desire to join collaborations. In this work, we discuss collaboration scenarios where people define services based on their dynamically changing skills and expertise by using Human-Provided Services. This approach is motivated by the need to support novel service-oriented applications in emerging crowdsourcing environments. In such open and dynamic environments, user participation is often driven by intrinsic incentives and actors properties such as reputation. We present a framework enabling users to define personal services to cope with complex interactions. We focus on the discovery and provisioning of human expertise in service-oriented environments.

Keywords Human provided services · Mixed service-oriented systems · Crowdsourcing · Social computing

1 Introduction

The collaboration landscape has changed dramatically over the last few years by allowing users to shape the Web and availability of information. In the past collaborations were bound to, for example, intra-organizational collaborations using a company's specific platform. It is now possible to utilize the knowledge of millions of

Communicated by Wil van der Aalst.

This work is supported by the European Union through the IP project COIN (FP7-216256).

D. Schall (✉)
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
e-mail: schall@infosys.tuwien.ac.at

Published online: 29 March 2011

people participating in interactions on the Web. Web services and service-oriented architectures (SOA) are the ideal technical framework to automate interactions spanning people and services. However, the transformation of how people collaborate and interact on the Web has been poorly leveraged in existing SOA. In SOA, compositions are based on Web services following the loose coupling and dynamic discovery paradigm. We argue that people should be able to define interaction interfaces (services) following the same principles to avoid the need for parallel systems of Software-Based Services (SBS) and Human-Provided Services (HPS) [32, 33]. We discuss *mixed service-oriented systems* that are composed of both humans and software services, interacting to perform certain activities. Here, user-provided services are well-defined interfaces to interact with people. The problem is that current systems lack the notion of human capabilities in SOA. The challenge is to support the user in providing services in open Web-based environments. HPSs can be discovered in a manner similar to SBS. Following this approach, humans are able to offer HPSs and manage interactions in dynamic collaboration environments.

Unlike traditional process-centric environments in SOA, we focus on flexible and open collaboration scenarios. In this work, we present the following novel key contributions:

- We present a motivating scenario for utilizing human capabilities described as HPSs in flexible service-oriented crowdsourcing applications.
- People need to be able to be able to provide services and to manage interactions in service-oriented systems. We present the HPS architecture and its core components: a *Middleware Layer* providing features for managing data collections and XML artifacts, the *API Layer* comprising services for user forms generation and XSD transformations, a *Runtime Layer* enabling basic activity and user management features as well as support for interactions using Web services technology.
- In open and dynamic environments, expertise profiles need to be maintained in an automated manner to avoid outdated information. We introduce a context-sensitive expertise ranking approach based on interaction mining techniques.
- We evaluate our approach by discussing results of our expertise mining approach.

This paper is organized as follows. We overview related work in the following Sect. 2. We present a scenario in Sect. 3 to motivate the need for Human-Provided Services in *crowdsourcing* environments. In Sect. 4, we present the HPS activity and task model enabling dynamic interactions in service-oriented systems. In Sect. 5, we discuss the Human-Provided Services architecture and framework. The discovery and selection of HPS is strongly influenced by human expertise. Our expertise ranking approach based on interaction mining techniques is presented in Sect. 6. Section 7 presents experiments and implementation details. We conclude the paper in Sect. 8.

2 Related work

We structure our discussion regarding related work in three topics: (i) *crowdsourcing* to clearly motivate the problem context of our work, (ii) *interaction modeling* to overview different techniques for structuring collaborations, and (iii) *metrics and expertise mining* to track user interest and skills in open Web-based platforms. Our

work is specifically based on the assumption that evolving skills and expertise influence how interactions are performed (for example, delegations) in crowdsourcing environments.

Crowdsourcing In recent years, there has been a growing interest in the complex 'connectedness' of today's society. Phenomena in our online-society involve networks, incentives, and the aggregate behavior of groups [11]. *Human computation* is motivated by the need to outsource certain steps in a computational process to humans [13]. An application of human computation in genetic algorithms was presented in [19]. A variant of human computation called games that matter was introduced by [40]. Related to human computation are systems such as Amazon Mechanical Turk[1] (MTurk). MTurk is a Web-based, task-centric platform. Users can publish, claim, and process tasks. For example [38], evaluated the task properties of a similar platform in cases where large amounts of data are reviewed by humans. In contrast to common question/answer (Q/A) forums, for example Yahoo! Answers[2], MTurk enables businesses to access the manpower of thousands of people using a Web services API. Mixed service-oriented systems target flexible interactions and compositions of Human-Provided and Software-Based Services [32]. This approach is aligned with the vision of the Web 2.0, where people can actively contribute services. In such networks, humans may participate and provide services in a uniform way by using the HPS framework [28]. A similar vision is shared by [25] who defines *emergent collectives* which are networks of interlinked valued nodes (services). In such collectives, there is an easy way to add nodes by distributed actors so that the network will scale. Current crowdsourcing platforms do not support complex interactions (e.g., delegation flows) that require joint capabilities of human and software services.

Questions include: how can people control flexible interaction flows in emerging crowdsourcing environments?

Interaction modeling In business processes (typically *closed* environments), human-based process activities and human tasks can be modeled in a standardized service-oriented manner. WS-HumanTask (WS-HT) [4] and BPEL4People (B4P) [3] are related industry standards released to address the need for human involvement in service-oriented systems. These standards and related efforts specify languages to model human interactions in BPEL [3], the lifecycle of humans tasks [4] in SOA, resource patterns [27], and role-based access models [21]. A concrete implementation of B4P as a service was introduced in [39]. A *top-down* approach, however, demands for the precise definition of roles and interactions between humans and services. The application of such models is therefore limited in crowdsourcing scenarios due to the complexity of human tasks, people's individual understanding, and unpredictable events. Other approaches focus on ad-hoc workflows [10], self-contained subprocesses (worklets) [1] based on *activity theory*, and task-adaptation [12] to cope with changing environmental conditions. In [22], business activity patterns were introduced to design flexible applications.

[1] Amazon Mechanical Turk: http://www.mturk.com/.

[2] Yahoo! Answers: http://answers.yahoo.com/.

Questions include: how can one control interactions in open and dynamic environments that are governed by the emergence of social preferences, skills and reputation?

Metrics and expertise mining Human tasks metrics in workflow management system have been discussed in [20]. A formal approach to modeling and measuring inconsistencies and deviations, generalized for human-centered systems, was presented in [8]. Studies on distributed teams focus on human performance and interactions [5, 24], as well as in *Enterprise 2.0* environments [7]. Models and algorithms to determine the expertise of users are important in future service-oriented environments. Task-based platforms allow users to share their expertise [42]; or users offer their expertise by helping other users in forums or answer communities [2]. By analyzing email conversations [9], the authors studied graph-based algorithms such as *Hyperlink-Induced Topic Search* (HITS) [18] and PageRank [23] to estimate the expertise of users. The authors in [35] used a graph-entropy model to measure the importance of users. The work by [43] applied HITS as well as PageRank in online communities (i.e., a Java Q/A forum). Approaches for calculating personalized PageRank scores were introduced in [15, 16] to enable topic-sensitive queries in search engines, but have not been applied to interaction analysis (social networks). Most existing link-based expertise mining techniques do not consider information related to the *interaction context*.

Questions include: how can interaction mining algorithms track users' expertise, interest, and skills in an automated manner considering context information?

3 Crowdsourcing

3.1 Overview

The shift toward the Web 2.0 allows people to write blogs about their activities, share knowledge in forums, write Wiki pages, and utilize social platforms to stay in touch with other people. Task-based platforms for human computation and crowdsourcing, including CrowdFlower[3], Google's Smartsheet[4], or Yahoo's Predictalot[5] enable access to the manpower of thousands of people on demand by creating human-tasks that are processed by the crowd. Human-tasks include activities such as designing, creating, and testing products, voting for best results, or organizing information. The notion of crowdsourcing describes an online, distributed problem solving and production model with increasingly interested business parties in the last couple of years [6]. One of the main motivations to outsource activities to a crowd is the potentially considerable spectrum of returned solutions. Furthermore, competition within the crowd ensures a certain level of quality. According to [41], there are two dimensions in existing crowdsourcing platforms. The first categorizes the function of the platform. Currently these can be divided in communities (i) specialized on novel designs and innovative ideas, (ii) dealing with code development and testing, (iii) supporting marketing and sales strategies, and (iv) providing knowledge support. Another dimension

[3] CrowdFlower: http://crowdflower.com/.

[4] Smartsheet: http://www.smartsheet.com/.

[5] Predictalot: http://pulse.yahoo.com/y/apps/vU1ZXa5g/.

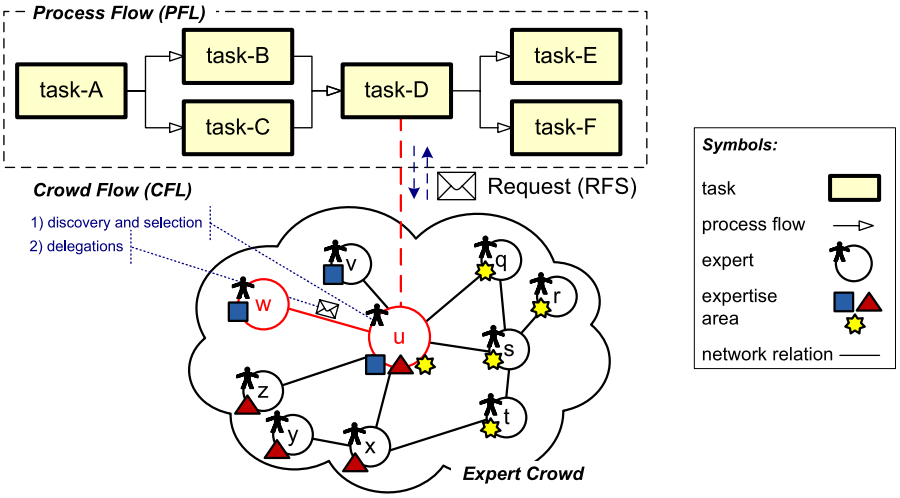


Fig. 1 Utilizing crowdsourcing in process flows

describes the crowdsourcing mode. Community brokers assemble a crowd according to the offered knowledge and abilities that bid for activities. Purely competition based crowdsourcing platforms operate without brokers in between. Depending on the platform, *incentives* for participation in the crowd are either monetary or simple credit-oriented. Even if crowdsourcing seems convenient and attracts enterprises with scalable workforce and multilateral expertise, the challenges of crowdsourcing are a direct implication of human's ad-hoc, unpredictable behavior and a variety of interaction patterns.

3.2 SOA for crowdsourcing

Service-oriented architecture (SOA) is an emerging paradigm to realize extensible large-scale systems. As interactions and compositions spanning multiple enterprises become increasingly commonplace, organizational boundaries appear to be diminishing in future service-oriented systems. In such open and flexible enterprise environments, people contribute their capabilities in a service-oriented manner. We consider mixed service-oriented systems based on two elementary building blocks: (i) Software-Based Services, which are fully automated services and (ii) Human-Provided Services [32] for interfacing with people in a flexible service-oriented manner. Here we discuss service-oriented environments wherein services can be added at any point in time. Following the open world assumption, humans actively shape the availability of HPSs by creating services. Interactions between HPSs are performed by using Web service-based technology (XML-based SOAP messages).

A motivating scenario for discovering members of the crowd in process-centric flows is depicted in Fig. 1. The **Process Flow** (PFL) may be composed of single tasks that are either processed by corresponding Web services or are assigned to responsible persons. In this scenario, a task (`task-D`) may be outsourced to the crowd. This is done by preparing a *request for support* (RFS) containing various artifacts to be

processed by the crowd and additional metadata such as time constraints and complexity of the task. The first step in a mixed service-oriented systems is to discover and select a suitable HPS. Discovery and selection is based on both, matching of functional capabilities (the service interface) and non-functional characteristics such as the degree of human expertise. In the depicted case, the actor u has been selected as the responsible service for processing the given request. The selection is based on u's expertise (visualized by the size of the node in the network), which is influenced by u's gradually evolving expertise and dynamically changing interests. The novelty of our approach is that members of the crowd may also interact with each other by, for example, simply delegating requests to other members (e.g., member u delegates the request to the peer w) or by splitting the request into sub-tasks that are assigned to multiple neighboring peers in the network. In our approach, the discovery of neighbors is based on the social structure of networks (e.g., friend or buddy lists). How decisions within the crowd are made (delegation or split of tasks) emerges over time due to changing interaction preferences and evolving capabilities of people (depicted as *expertise areas*). These dynamic interactions are defined as **Crowd Flow** (CFL). Flexible interaction models allow for the natural evolution of communities based on skills and interest. Our expertise mining approach and techniques help to address flexible interactions in crowdsourcing scenarios.

4 HPS interaction model

The availability of interaction models in open, Web-based platforms such as the motivating crowdsourcing scenario is currently limited. Most existing crowdsourcing platforms do not support interactions and collaborations between users (tasks are typically assigned to individuals). Other Web-based tools (e.g., bulletin boards, email, instant messaging) lack the ability to compose the capabilities of human- and software-based services, which typically requires standardized (XML-based) message formats, interfaces, etc. The main purpose of the proposed interaction model is:

1. Provisioning of human expertise in a service-oriented manner using SOA principles (Sect. 4.1). Examples are 'document review' [32] or 'document translation' [34] services provided by human actors.
2. Model to support flexible interactions between crowd members (Sect. 4.2).
3. Task model that can be used to link PFL artifacts (task descriptions) to flexible crowd-activities which are provisioned through HPS (Sect. 4.3).

4.1 HPS activity model

Activities are used for different purposes. People use activities to structure collaborations in a flexible manner. Also, activities enable users to define Human-Provided Services. We now turn to the activity model enabling the use of HPS in various interaction scenarios, for example CFLs. The presented activity model in Fig. 2 depicts the most important elements to support basic interaction scenarios.

– An *ActivityDeclaration* defines the name and description of an activity, URI, and a set of tags that can be applied to the declaration. Tags are applied by users to associate keywords to declarations.

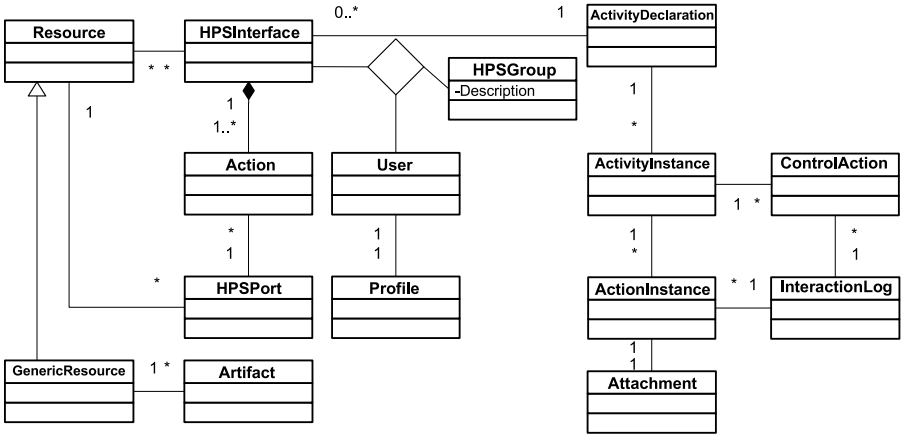


Fig. 2 Overview of HPS activity model

- The *HPS Interface* relates to an *ActivityDeclaration*. *Name* in the *HPSInterface* depicts the HPSs name, for example, a *review service*. The *HPSInterface* (description) is very similar to the description of conventional SBS. We perform a simple mapping to depict declarations as Web service descriptions (e.g., using WSDL).
- An *HPSGroup* defines the set of people providing a certain type of service established as the relation between *User*, *HPSInterface*, and *HPSGroup*. An HPS requester can be a human seeking the opinion of experts or a composed software service (PFL) requiring human input. A ranking procedure must be used to select the best available HPS. We term the relation between *User* and *HPSInterface personal service*, which is technically an instance of an HPS. Each user has a *Profile* identifying people and storing user preferences.
- A *Resource* is used for different purposes. As mentioned before, *HPSInterfaces* are depicted using languages such as WSDL. Thus, the interface is an XML document that can be modified by using resource identifiers (URIs) to retrieve or update resources. Other resources are *type definitions*, for example, activity types and/or parts of complex data types.
- A *GenericResource* is a special type of *Resource*, which we use to wrap *Artifacts*. *Artifacts* include collaboration documents and all sorts of files that are used and created during collaborations. The *GenericResource* defines metadata associated with *Artifacts*.
- The *Action* concept is used to interact with HPSs in the scope of an activity. The *HPSInterface* is composed of a set of *Actions*. Notice, there are different action concepts in our model. On the one hand, *Action*, as discussed here, is defined by the user in the scope of an *HPSInterface*. The definition of an *Action* is done at *design time*.
- The *HPSPort* depicts the technical—in a Web services sense—realization of an HPS interface. (The details are not needed at this point and will be discussed in the HPS framework section.) The *HPSPort* relates to a set of resources (e.g., typed messages), which are used in certain *Actions*.

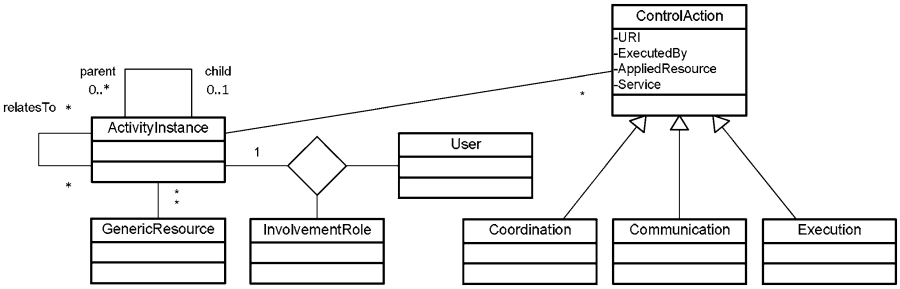


Fig. 3 Excerpt of hierarchical activity model

The previous concepts were introduced as models to depict and design HPSs. The following concepts describe activity and HPS-centric interactions at *run-time*.

- An *ActivityInstance* represents an actual work item. An activity can be performed many times, which are called instances of the activity. Each instance corresponds to a declaration. Instances represent the context of interactions.
- An *ActionInstance* is connected to an *ActivityInstance*. Each *ActionInstance* is defined by an *Action*. An *Attachment* is something generic to associate XML documents, for example, XML messages that are exchanged between services, and other content-types with an *ActionInstance*. *Attachments* usually convey typed messages that are defined in an *HPSInterface* and *Resources*.

  Both *ControlAction* and *ActionInstance* are used at run-time. A *ControlAction*, however, depicts common action types in human collaboration. *ControlActions* include *coordination*, *communication*, and *execution* actions that are associated with instances of activities, for example *delegations* of activities. However, such actions are not part of an *HPSInterface*.

  A *ControlAction* is always used between two or more people to, for example, coordinate the execution of activities; whereas an *ActionInstance* may be the result of interactions between human and software services. Each action, *ControlAction* as well as *ActionInstance*, is logged to keep a history of interactions. The *InteractionLog* captures traces of interactions (activities and their actions) performed in collaborations. Also, interactions between software services are logged to maintain a history of the collaboration context.

4.2 Hierarchical activities

Activities can be structured as hierarchies (see Fig. 3) using *parent* and *child* relations. Child activities specify the details with respect to the (sub-)steps in collaborations, for example, sub-activities in the scope of a parent activity. This allows for the refinement of collaboration structures as the demand for a new set of activities (e.g., performed by different people and services) increases. The need for the dynamic refinement of collaboration structures is especially required when people have limited experience (the history of performed activities) with respect to a given objective or goal. Furthermore, some people tend to underestimate the scale and complexity of an activity; thus the hierarchical model enables the segmentation of activities into sub-activities, which can be, for example, delegated to other people.

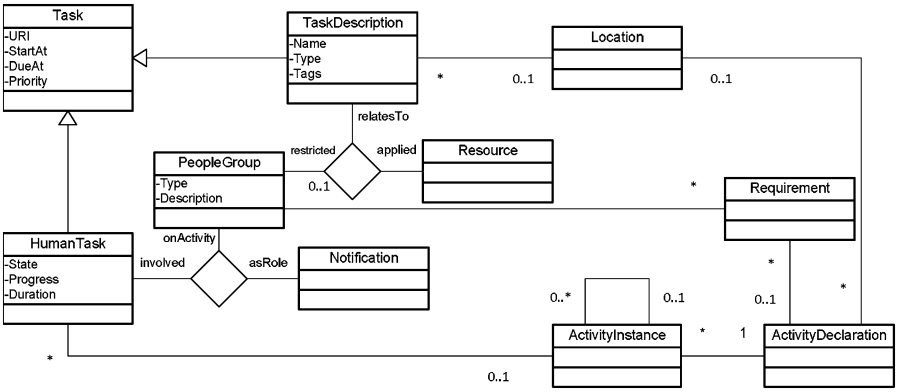


Fig. 4 Overview task model

The basic HPS activity model (cf. Fig. 2) did not define any notion of activity hierarchies because, currently, we do not support the mapping of activity hierarchies onto HPSs. For example, hierarchically structured activities in activity declarations would require a mapping of such hierarchies into a set of *Actions*. Activities have a *relatedTo* property which provides a mechanism to link to any other activity. Typically, multiple members work on the same activity with different roles. The *InvolvementRole* identifies the creator, observer, contributor, responsible, and supervisor of an activity. Involved workers apply a set of *GenericResources* to perform their work. As mentioned before, objects such as documents are represented as a shared *Artifact*. A *ControlAction* captures activity-change events, interactions between members, and work carried out. Actions can trigger events describing the progress of activities.

4.3 Task model

In most collaborations, activities need to be controlled by capturing temporal aspects such as *progress* of activities and monitoring of *deadlines*. In this section, we define an extended task model, which can be used in open collaboration scenarios; for example, in HPS-based collaborations on the Web. Figure 4 shows task-related concepts and their relation to previously introduced concepts.

Controlling the execution of activities The most fundamental aspect is to control the execution of activities by associating a *HumanTask* with an *ActivityInstance*. Multiple tasks can be created because activity instances can be divided into sub-activities. A *HumanTask* is derived from a generic *Task* defining basic task-properties—*StartAt*, *DueAt*, and, *Priority*. If tasks are used in HPS-based collaborations, requesters are aware of the state of a given interaction (e.g., *accepted*, *inprogress*, or *completed*). Based on these execution parameters, for example, the properties *Priority* and *DueAt*, *Notifications* can be sent to a set of people. Examples include, notify a set of people (*PeopleGroup*) about the status of an activity, escalate deviations in the execution of activities, or notify the supervisor of an activity when the activity (or one of its sub-activities) has been completed.

This model is well aligned with the WS-HumanTask (WS-HT) specification [4]. Moreover, functional properties can be associated with *ActivityDeclarations*, depicted as *Requirement* in Fig. 4; for example, role models controlling whether users are allowed to work on activities. Again, a generic *PeopleGroup* is used which is populated with a set of people depending on specified requirement. Notice, requirements typically do not change over time. For example, if we use a role model to control the set of people who can work on an activity, we follow a *top-down* view—modeling how an activity should be performed. In contrast, *constraints* change over time depending on the run-time context. Constraint are, for example, the minimum set of skills or level of expertise a potential worker must have. Indeed, skills and level of expertise change over time depending on performed activities.

Creating announcements The idea of the HPS model is not only to support enterprise collaboration scenarios but also Web-based collaborations. In enterprises, a corporate directory usually holds all information regarding employees, their role in the company, and additional contact information, which can be accessed to populate a *PeopleGroup*. However, these announcements are well applicable to enterprise collaborations as well because in global corporations it is impossible to maintain expertise, roles, interests of employees in a central directory.

Here we define two scenarios showing the usefulness of announcements:

– We can imagine a *TaskDescription* as an announcement to express the need for a set of HPSs to work on tasks. The notion of task descriptions is similar to marketplaces of work in task-based platforms on the Web, for example, Amazon's Mechanical Turk where Human Intelligence Tasks (HITs) are used for this purpose. See the relation between *TaskDescription*, *Resource*, and *PeopleGroup* in Fig. 4. A *Resource* describes an HPS as previously discussed in the basic HPS activity model.

  Task descriptions comprise constraints such as task availability information (beginning and expiration time of the task) and the number of available task instances (how many of those tasks can be claimed by users). In this case, it is clear that a particular type of HPS has to be used in the context of a task.
– The relation between *ActivityDeclaration*, *TaskDescription*, and *Location* depicts the need for a service—potentially in a specific location area.

  Therefore, these kind of announcements are opportunities for users to create *new* HPSs or to associate an existing HPS with a description which has not been considered before. Such announcements are different with respect to the previous case (marketplace example) because *ActivityDeclaration* and *TaskDescription* do not demand for a particular type of HPS.

4.4 Task execution model

The next step is to introduce a *task execution model* defining the possible task states. The task execution model is depicted by Fig. 5. It is relevant for both cases, announcements of task and the control of activity executions.

– *Claiming Tasks:* Announcements allow requesters to denote the availability of work items (i.e., activities) without explicitly selecting a particular HPS. Announcement can be generated if there is not any matching HPS available; or if

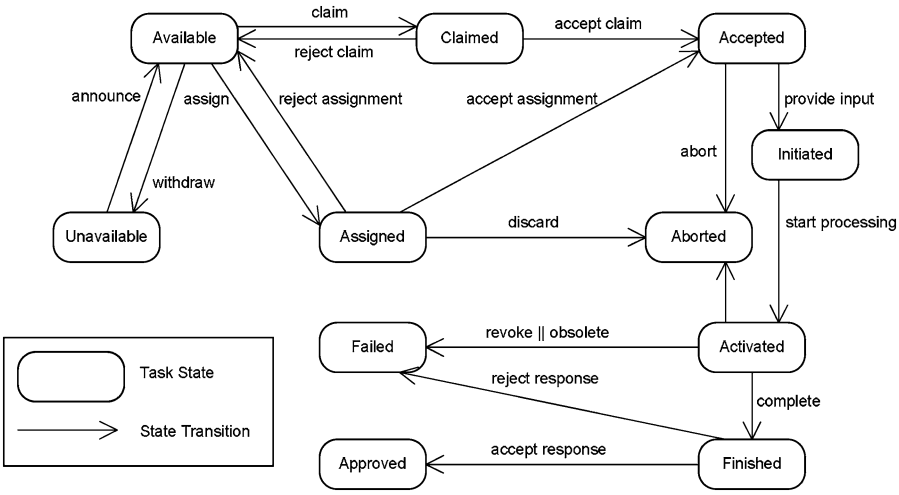


Fig. 5 HPS Task execution model

the demanded HPS is currently not provided by users. Initially, a task is set to *Available* and becomes *Unavailable* when the announcement expires. Based on announcements, tasks can be *Claimed*, *Accepted* or rejected by requesters (becoming *Available* again).

– *Task Assignments:* A task can be *Assigned* to HPSs without issuing announcements, specifically when software services generate tasks that need to be processed based on, for example, deadlines. An *Assigned* task may go into the *Accepted* state, otherwise to *Aborted* when the assignment procedure times out. For example, the user is not responding to an assignment request.

The task state changes from *Accepted* to *Initiated* when an action is performed in the context of an activity (e.g., sending a request to an HPS). The task changes its state to *Aborted* if the initiation fails (*Initiated* state). The state *Activated* indicates that the request is processed, followed by the *Finished* state or *Failed* if the HPS was unable to deliver the desired output—the expected information, which can be validated by, for example, a (human) requester reviewing the output. A task is successful if the output of an HPS is *Approved* by the requester.

5 Architecture

Most systems based on SOA-principles (registration, discovery, and interaction) typically lack the notion of human capabilities that can be provisioned as service. Traditional service-oriented systems provide support for software-based services only. We propose *mixed service-oriented systems*. Moreover, existing tools for designing services usually require 'expert' knowledge in terms of understanding various WS-standards. HPS harnesses human capabilities within service-oriented environments while leveraging Web 2.0 innovations.

In addition, our architecture provides an approach for interaction monitoring that captures the *context of interactions* through activity identifiers. Monitored interac-

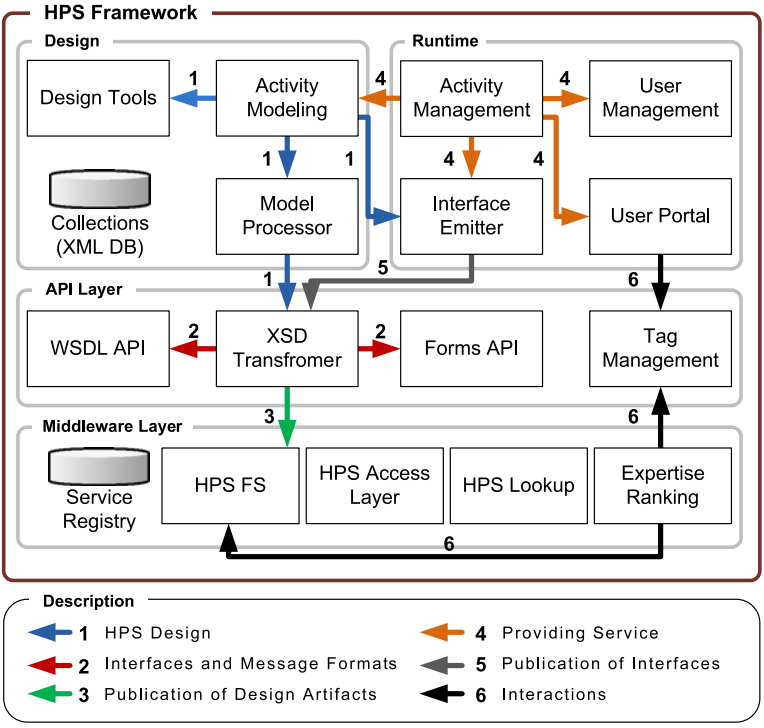


Fig. 6 HPS framework and architecture

tions are logged and analyzed to calculate various metrics such as reputation and interest profiles of users. The automatic calculation of reputation and skills through mining is a novel technique to support the discovery of human capabilities in SOA considering changing expertise and interests of people.

5.1 HPS framework

This section details the HPS framework by introducing various services to enable human interactions in SOA. The HPS platform allows requesters (people and software services) to find and interact with HPSs. The framework offers a set of tools to support the design of HPSs and a middleware hosting various services such as the HPS Access Layer (HAL). Figure 6 shows the main components of the HPS framework. The arrows in the figure depict a 'using' relation between various blocks.

Design tools HPS Design tools allow users to create service interfaces (annotation 1) in a simplified manner. These tools are hosted in a Web portal (see [28] for details). Figure 6 illustrates the design flow:

– *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 (annotation 2) without requiring the user to understand underlying technologies such as XML or WSDL.

– *Publication of Design Artifacts:* artifacts such as message formats and activity definitions are saved in XML collections (annotation 3).

API layer The framework includes services and tools for the design of HPS as well as runtime support for the automatic generation of interfaces. The *API Layer* includes the following core services:

– *WSDL API* service to generate service descriptions; in particular, to create WSDLs based on human activities and user specified interface elements (parameters and complex elements).
– *Forms API* implementing support for XML Forms (XForms).
– *XSD Transformer* service utilizing the *Forms API* to automatically generate XForms based on XML schema definitions, for example, as defined in WSDL documents.
– *Tag Management* service associating tags with HPS artifacts (activities, actions, and WSDLs).

Runtime infrastructure services The following services have been designed and implemented to enable HPS-based collaboration.

– The *Activity Management* service maintains activity declarations and activity instances (annotation 4).
– The *User Management* service holds data related to profiles and contact details.
– The *Interface Emitter* generates HPS interfaces depending on the interaction scenario (annotation 5); for example, interactions between humans or interactions requiring WSDL interfaces (e.g., compositions of HPS and software services). Since collaboration scenarios include enterprise collaborations, for example, Web-based portals implementing rich user interfaces, and also mobile collaboration scenarios, interface generation can be customized based on the user's current context. Therefore, based on the requirements and constraints of the current or preferred user device, different interface representations can be generated.

Middleware layer The *HPS FS* is an XML based, distributed file system to manage user profiles, human tasks, service related information such as WSDL descriptions, and personal services (see also [31]). The HPS FS offers a set of APIs to manage XML artifacts and collections via the Atom Protocol Model[6] to retrieve and update HPS related information. We embed HPS interfaces, described using WSDL, as elements in Atom-based XML documents (see Atom Syndication Format[7]). Atom-formatted representations contain HPS 'information items' with the advantage that various Web 2.0 authoring tools and APIs can be used to retrieve and update Atom-based elements. HPS information includes: (i) which services are registered with the HPS framework, (ii) how to interact with services, (iii) the geographic location of services; if location information is shared by the user, and (iv) other context information of an HPS including the current availability of a particular service.

[6] Atom Protocol Model: http://tools.ietf.org/html/rfc5023.

[7] Atom Syndication: http://tools.ietf.org/html/rfc4287.

HAL dispatches and routes SOAP requests to the corresponding service. Thus, humans and software services (i.e., HPS requesters) are able to interact with HPSs by issuing requests toward the HPS middleware. HAL implements security features to prevent unauthorized access and allows requests to be routed according to user-defined rules (e.g., automatic delegations based on load-conditions [37]). The *HPS Ranking* algorithms are used for the analyses of human and service interactions to recommend the most suitable HPS based on various interaction and task metrics. Ranking results and recommendations can be requested from a *Expertise Ranking* service (annotation 6). The *HPS Lookup* supports various ways to discover HPSs. Web browsers can be used to obtain a list of services as 'news items' embedded in Atom elements. For example, the middleware implements a service which returns XML documents as news feeds containing HPS-related information. We have implemented this mechanism to support the integration of HPS with other Web 2.0 platforms. Also, a Web services-based API can be used to support typical lookup operations to get a list of available services. The middleware hosts a *Service Registry* that is used when the lookup is performed.

5.2 Data collections

The HPS framework utilizes Web services technology to enable HPS at the technical level. Therefore, various XML-based collections and resources need to be managed in an efficient manner. In HPS, XML-based collections are managed by the HPS FS. Basic create, read, update, and delete (CRUD) operations can be performed on HPS-related information. As mentioned before, the Atom protocol is used for this purpose. Resources and collections include:

- *User Profile and Metrics:* Profiles contain *hard* and *soft-facts*. Hard-facts includes information as 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 consists of *weights* (skill level of a user), *classification* (description of area or link to taxonomy), and *evidence* (external sources acting as reference or recommendation). Soft-facts can be generated by the middleware based on users' activities to indicate expertise or skill level. We use friend-of-a-friend (FOAF[8]) profiles to manage social networks structures (e.g., buddy lists) and other user information.
- *Service Registry:* The registry maintains a number of XML documents describing HPS. This information includes a set of service definitions, the list of available services, and information regarding personal services. The term *personal service* was introduced as a metaphor for a service instance. Service instance is a purely technical term to denote the number of physically deployed services that have the same (syntactic) interface characteristics.
- *Task Registry:* Manages human tasks that can be either public tasks (i.e., announcements used to advertise the need for HPSs) or private tasks that are associated with HPS-based interactions to control the status of collaborations. Public tasks are associated with an interaction upon claiming and processing tasks.

[8] FOAF: http://xmlns.com/foaf/spec/.

5.3 Interactions and monitoring

The HPS framework dynamically generates interfaces for the discovery of services and interactions with users. Next, we show a (simplified) WSDL-based interface description to realize HPS-based *support services* (as introduced in the crowdsourcing scenario).

```
1  <?xml version="1.0"?>
2  <wsdl:definitions name="SupportService" ...>
3   <wsdl:types>
4    <xsd:schema targetNamespace="http://myhps.org/rfs">
5     <xsd:complexType name="GenericResource">
6      <xsd:sequence>
7       <xsd:element name="Location" type="xsd:anyURI" />
8       <xsd:element name="Expires" type="xsd:dateTime" />
9      <xsd:sequence>
10    </xsd:complexType>
11    <xsd:complexType name="Request">
12     <xsd:sequence>
13      <xsd:element name="SupportResource" type="GenericResource" />
14      <xsd:element name="Comments" type="xsd:string" />
15     </xsd:sequence>
16    </xsd:complexType>
17    <!-- further types ... -->
18    <xsd:element name="SupportRequest" type="Request" />
19    <xsd:element name="AckSupportRequest" type="xsd:string" />
20    <xsd:element name="GetSupportReply" type="xsd:string" />
21    <xsd:element name="SupportReply" type="Reply" />
22   </xsd:schema>
23  <wsdl:types>
24  <wsdl:message name="GetSupport">
25   <wsdl:part name="part1" element="SupportRequest" />
26  </wsdl:message>
27  <wsdl:message name="AckSupportRequest">
28   <wsdl:part name="part1" element="AckSupportRequest" />
29  </wsdl:message>
30  <!-- further messages ... -->
31  <wsdl:portType name="HPSSupportPortType">
32   <wsdl:operation name="GetSupport">
33    <wsdl:input xmlns:wsaw="http://.../addressing/wsdl"
34     message="GetSupport" wsaw:Action="urn:GetSupport" >
35    </wsdl:input>
36    <wsdl:output message="AckSupportRequest" />
37   </wsdl:operation>
38  </wsdl:portType>
39  <wsdl:binding name="HALSOAPBinding" type="HPSSupportPortType">
40   <soap:binding style="document"
41    transport="http://xmlsoap.org/soap/http" />
42  </wsdl:binding>
43 </wsdl:definitions>
```

Listing 1 HPS WSDL definition

Listing 1 shows a complete HPS WSDL example to support the discovery of HPS interfaces. Lines 4–23 define XML type definitions including `GenericResource` and `SupportRequest`. The user can create such definitions by using tools hosted by the HPS platform. In this simplified example, the activity to be performed by a

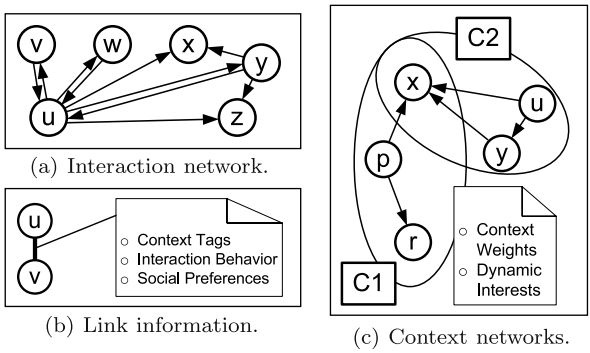
human is the previously mentioned request for support (RFS) activity comprising resources, the actual request, and the reply, which is a complex XML data structure (abbreviated in this example). Lines 24–29 show an excerpt of WSDL messages. However, we only show the request denoted as `SupportRequest`.

The `HPSSupportPortType` is described by lines 32–39. Notice, the HPS Access Layer (HAL) dispatches all interactions. At run-time, HAL extracts and routes messages to the demanded HPS. Since every interaction is entirely asynchronous, interactions (session) identifier are automatically generated by HAL (e.g., `AckSupportRequest`). Finally, lines 40–43 show the `HALSOAPBinding` of the `HPSSupportPortType`.

```
1  <?xml version="1.0"?>
2  <soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
3   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
4   xmlns:hps="http://myhps.org/"
5   xmlns:types="http://myhps.org/types"
6   xmlns:rfs="http://myhps.org/rfs">
7   <soap:Header>
8    <types:timestamp value="2010-03-05"/>
9    <types:delegation hops="3" deadline="2010-03-06"/>
10   <types:activity url="http://www.coin-ip.eu/Activity#42"/>
11   <wsa:MessageID>uuid</wsa:MessageID>
12   <wsa:From>http://.../Actor#Florian</wsa:From>
13   <wsa:ReplyTo>http://.../Actor#Florian</wsa:ReplyTo>
14   <wsa:To>http://.../Actor#Daniel</wsa:To>
15   <wsa:Action>http://.../Type/RFS</wsa:Action>
16  </soap:Header>
17  <soap:Body>
18   <hps:Request>
19    <rfs:subject>WSDL consumption with Axis2</rfs:subject>
20    <rfs:requ>Axis2 reports a parsing error while consuming
21              the given resource. What is wrong?</rfs:requ>
22    <rfs:comments>Used Axis2 1.4</rfs:comments>
23    <rfs:keywords>WSDL, Axis2</rfs:keywords>
24    <rfs:category>Software/SE/General/SE for Internet projects
25    </rfs:category>
26    <rfs:resource>
27       <!-- details omitted -->
28    </rfs:resource>
29   </hps:Request>
30  </soap:Body>
31 </soap:Envelope>
```

Listing 2 Simplified RFS via SOAP example

The HPS Access Layer logs each service interaction (request and response message) through a logging service. RFSs and their responses, exchanged between crowd members, are modeled as traditional SOAP calls, but with header extensions, as shown in Listing 2. The most important SOAP-header extensions include: The `Timestamp` captures the actual creation of the message and is used to calculate temporal interaction metrics, such as the average response time. The tag `Delegation` holds parameters that influence delegation behavior, such as the number of subsequent delegations `numHops` (to avoid circulating RFSs) and `deadlines`. The `Activity uri` describes the context of interactions that is based on the previously introduced activity model. The `MessageID` enables message correlation to



(a) Interaction network.

(b) Link information.

(c) Context networks.

Fig. 7 Collaborative networks: (**a**) Interactions are performed between nodes in the network; (**b**) Metadata and metrics are associated with links between nodes; (**c**) Context networks are created based in link information

match request/response pairs. `WS-Addressing` tags, besides MessageID, are used to route RFSs through the crowd.

Interactions are periodically analyzed to calculate metrics such as reputation and trust between community members. While the depicted architecture follows a centralized approach, the logging facilities are replicated for scalability reasons, and monitoring takes place in a distributed form. Interactions are purged in predefined time intervals, depending on the required depth of history needed by metric calculation plugins (e.g., for trust inference [36]).

6 Expertise ranking

Evolving skills, interests and expertise need to be maintained in an automated manner to avoid outdated profile information. Top-down approaches define interest and expertise areas using taxonomies and ontologies. Here we follow a interaction mining approach that addresses inherent dynamics of flexible collaboration environments.

6.1 Context-sensitive interaction mining

Our expertise ranking approach is based on observed interactions (from logs) and analysis of the structure and dynamics of interaction networks. Therefore, an interaction network (see Fig. 7(a)) is modeled as a graph $G = (N, E)$ composed of the set of nodes N and the set of edges E. Note, here the terms edge and link have the same meaning.

We argue that context information is essential for expertise mining. The context of an interaction can be captured by, for example, extracting relevant keywords from messages exchanged between users or by tags applied to various collaboration artifacts. In this work, we focus on *tags* (Fig. 7(b)) serving as input for contextual link information. Interactions such as delegation requests are tagged with keywords. As delegation receivers process tasks, our system is able to learn how well people cope with certain tagged tasks; and therefore, able to determine their centers of expertise. The profile $P(u) = \langle f_u(t_1), f_u(t_2), f_u(t_3), \dots \rangle$ describes the frequencies f_u of tags $T = \{t_1, t_2, t_3, \dots\}$ that are applied in collaborations by and with u. Interaction metrics such as weights depicting the interest and focus of a user to collaborate with other peers in a specific context are automatically calculated through mining. Figure 7(c) shows networks for context $C1$ and $C2$. Each context network may have one or more tags associated with it.

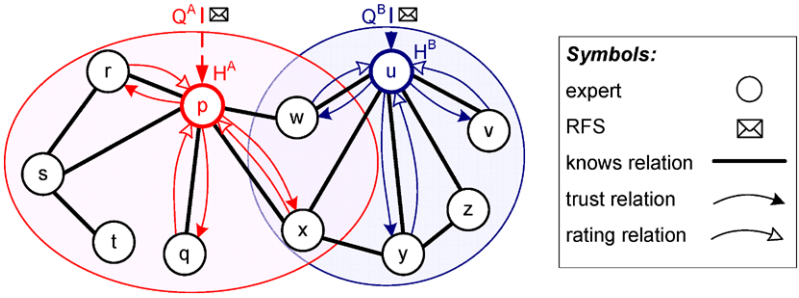


Fig. 8 Hubs in different personalized expert queries

Existing work in the area of expertise mining (e.g., [43]) typically focuses on a graph representation as depicted by Fig. 7(a). In contrast, we present an approach and algorithm that is suitable for scenarios as shown in Fig. 7(c). We base our expertise mining algorithm on well proven and theoretically sound techniques (i.e., [18] and [23]). Specifically, we take the notion of *hubs and authorities* as introduced by Kleinberg [18] as a starting point to derive a context-sensitive expertise mining approach.

6.2 Hubs and authorities

The notion of *authorities* in social or collaborative networks can be interpreted as a measure to estimate the relative standing or *importance* of individuals in social networks. Applying this idea in our crowdsourcing scenario (see Fig. 8), a member of the Expert Crowd may receive an RFS and delegate work to some other peer in the network (characterizing hubs in the network). For example (as depicted in Fig. 8), u delegates the received RFS to w. Receivers of the delegated work, however, expect RFSs fitting their skills and expertise (i.e., being an authority in the given domain). Careless delegations of work will overload these peers resulting in degraded processing time due to missing expertise.

Within the Expert Crowd, authorities give feedback using rating mechanism (e.g., a number on the scale from 1 to 5) to indicate their satisfaction; i.e., whether a particular hub distributes work according to their skills and interest. Thus, a 'good hub' is characterized by a neighborhood of peers that are satisfied with received RFSs. Also, delegation of work is strongly influenced by *trust*, for example, whether the initial receiver of the RFS (hub within the Expert Crowd) expects that a peer will process work in a reliable and timely manner. RFS receivers need to be trusted by influential hubs that are highly rated to be recognized as authoritative peers in the Expert Crowd.

6.3 Personalized expert queries

Here we utilize the concept of *personalized expert queries* (as introduce in [30]) to discover *expert hubs* that are well-embedded[9] in expertise networks given a particular query context. Delegation is important in flexible, interaction-based systems

[9] A hub is thereby characterized by the social network structure (node degree) *and* connection strength (e.g., count of delegated or processed RFSs) based on joint collaborations.

because expert hubs typically attract a large amount of RFSs over time (due to their distinguished expertise). From a network perspective, this means that hubs will be 'bottlenecks' due to the limited capacity and processing speed of the HPS. However, being a hub in the Expert Crowd means that a person *knows* many other experts in similar expertise areas. The main argument of our approach is that the likelihood of a successful delegation of RFSs to other experts increases based on the *hubness* of a person (embedding of a person in expert areas such as communities and interest groups).

Let us start formalizing this concept. A personalized expert query Q^C is defined as $Q^C = (KW, W(kw))$ where $KW = \{kw_1, kw_2, kw_3, \dots\}$ is the set of keywords or terms determining the context C of a query. Each keyword kw may have a weight associated with it depicted by $W(kw)$. Consider the scenario in Fig. 8. First, a query (see Q^A and Q^B) is specified either manually by a (human) expert seeker or derived automatically from a given process context (PFL), for example a predefined rule denoting that a particular set of skills is needed to solve a problem. The purpose of a query is to return a set of experts who can process RFSs, either by working on the RFSs or delegation. Thus, Q^A would return H^A as the user who is well-connected to *authorities* in query context Q^A. Being well-connected means that H^A has the highest number of links *and* has performed interactions over these links (e.g., delegations) that are relevant for a given query context. There are two influencing factors, i.e., relations, determining hub- and authority scores: (i) how much hubs *trust* authorities (depicted as filled arrows from hubs to authorities) and (ii) *ratings* hubs receive from authorities (open arrows to hubs). Trust mainly influences the potential number of users (e.g., known by H^A) who can process delegated RFSs. On the other hand, receivers can associate ratings to RFSs to express their opinion whether the delegated RFSs fit their expertise. Q^B may demand for a different set of skills. Thus, not only matching of actors is influenced, but also the set of interactions and ratings considered for calculation expertise scores (i.e., only the set of RFSs and ratings relevant for Q^B). Note, single *interactions* that lead to trust relations, as well as single *rating actions* that lead to rating relations are not depicted by Fig. 8. A single arrow may in fact depict a number of interactions or ratings.

6.4 Ranking model

One of the main pillars of our work is to consider the *context* in which interactions take place. In our previous work we defined two independent expertise ranking approaches, one called *DSARank* [29] and the other approach called *ExpertHITS* [30]. Here we introduce a generalized ranking approach based on our previous discussions on the concept of hubs and authorities in evolving Expert Crowds. The starting point for our ranking algorithm is (1) (see [18, 30]).

$$H(u) = \sum_{(u,v)\in E} A(v) \qquad A(v) = \sum_{(u,v)\in E} H(u) \tag{1}$$

The edge (u, v), which reads u `knows` v, is established based on links in the social network (FOAF profiles). Notice, by ranking nodes in a graph G using this method, each node $n \in N$ receives both hub *and* authority scores. However, we are

Table 1 Interaction weights and related symbols

Symbol	Description
w^Q_{vu}	The link weight based on ratings given by v to RFSs received from u
w^Q_{zv}	The connection strength of a hub z to authority v. Delegation behavior of hubs is based on the success of interactions (successful completion of delegated task)

primarily interested in computing the *hub importance* $H(u)$ of a particular node. This is motivated by the need to find coordinators who distribute requests by delegating tasks within the Expert Crowd [30] (emerging CFLs). However, we argue that an expertise mining algorithm must consider a person's interest and activity level in a certain collaboration context. As proposed in [29], *preferences* that are based on mining of interaction metrics can be used to compute contextual expertise profiles.

$$H(u;Q) = (1-\lambda_h)\,p(u;Q) + \lambda_h \sum_{(u,v)\in E} w^Q_{vu} A(v;Q) \tag{2}$$

Computing contextual expertise profiles is accomplished by expanding (1) in terms of adding $(1-\lambda_h)\,p(u;Q)$ to the standard HITS model as shown in (2). The parameter λ_h is used to balance between preferences $p(u;Q)$ and the propagation of global importance scores denoted by the term $\sum_{(u,v)\in E} w^Q_{vu} A(v;Q)$. The link weight w_{vu} based on Q is discussed in Table 1. From the network point of view, the definition in (2) can be interpreted as *influence propagation* based on a node's outgoing links. This is similar to TrustRank [14] where trust scores are propagated along neighboring outlinks. TrustRank is based on an *inverse PageRank* model that utilizes *good seeds* to influence trust flows. Also (2) permits a similar interpretation because $H(u;Q)$ can be computed as the inverse PageRank [14]. However, our approach closely follows the *personalized PageRank* model [23] by assigning preferences to the personalization vector $p(u;Q)$ to create context-aware importance rankings.

Similarly, importance scores for authorities $A(v;Q)$ are determined using (3):

$$A(v;Q) = (1-\lambda_a)\,p(v;Q) + \lambda_a \sum_{(z,v)\in E} w^Q_{zv} H(z;Q) \tag{3}$$

Without considering the dual nature of HITS (assigning hub and authority scores to each node in the network), we can regard (3) as the personalized PageRank model that is biased towards a particular interaction context using the contextual preference vector $p(v;Q)$. Again, the weight w^Q_{zv} is detailed in Table 1. Notice, (3) permits an interpretation of delegation behavior within the Expert Crowd as a stochastic process as hubs may choose to interact with known authorities or decide to pick a *newcomer* for task delegation either randomly[10] or based on, for example, interest similarities (see also [36] for bootstrapping newcomers in collaborations).

[10]The probabilistic interpretation of PageRank is known as the *random surfer* model [23].

To create a unified equation for $H(u; Q)$, we substitute $A(v; Q)$—as defined in (3)—in (2) and define the hub importance of u as follows:

$$H(u; Q) = (1 - \lambda_h) p(u; Q) + \lambda_h (1 - \lambda_a) \sum_{(u,v) \in E} w_{vu}^{Q} p(v; Q) + \lambda_h \lambda_a \sum_{(u,v) \in E} \sum_{(z,v) \in E} w_{vu}^{Q} w_{zv}^{Q} H(z; Q) \quad (4)$$

Equation (4) provides the basic formalism to determine coordinators based on contextual preferences. Next, we reformulate the context-sensitive personalization vector $p(u; Q)$ as follows (based on (4)):

$$p'(u; Q) = \frac{(1 - \lambda_h)}{(1 - \lambda_a)} p(u; Q) + \lambda_h \sum_{(u,v) \in E} w_{vu}^{Q} p(v; Q) \quad (5)$$

Equation (5) essentially consists of two components: preferences given to a particular hub u, for example based on the PFL problem context, and how well u is rated by authorities expressed by the weight w_{vu}^{Q}. The authority preference vector $p(v; Q)$ is personalized based on interaction dynamics captured by metrics such as the *interaction intensity* of v. We refer interested readers to [29] for a detailed description on these metrics.

Here we focus on personalizing $p'(u; Q)$ based on ratings to reduce the complexity of preference parameters (i.e., determining $p(u; Q)$). By setting $\lambda_h = 1$ we have:

$$p'(u; Q) = \sum_{(u,v) \in E} w_{vu}^{Q} p(v; Q) \quad \text{with } \lambda_h = 1 \quad (6)$$

Based on (4) and (6), let us define the following equation to estimate the hub importance of a given network node u:

$$I^H(u; T') = (1 - \lambda) p'(u; T') + \lambda \sum_{(u,v) \in E} \sum_{(z,v) \in E} w_{vu}^{T'} w_{zv}^{T'} I^H(z; T') \quad (7)$$

Equation (7) introduces various new concepts (detailed in Table 2). In particular, we define I^H as the hub importance of a node u since our approach does not require two types of rankings (hub and authority scores) anymore. Given (7), we have derived an expertise ranking model that is similar to the basic idea of PageRank. While such a model has been extremely successfully applied to search engines on the Web, the drawback is the complexity of computing the PageRank equation[11]. Especially in crowdsourcing scenarios that require on-demand discovery of experts based on a set of specified skills, computation of expertise scores taking up to several hours is not acceptable. We have first raised this issue in [28] and proposed a combination of *offline mining* and *online aggregation* of expertise ranking scores based on query

[11] In large social networks (for example network size >10000 nodes), it may take up to several hours to compute PageRank importance scores.

Table 2 Topic-sensitive hub importance and related symbols

Symbol	Description
I^H	The topic-sensitive hub importance score of a given node in G
T'	Topic $T' \subseteq T$ based on a set of tags applied to interactions. T' can be calculated automatically based on tag-clustering techniques (e.g., see [36]) or by using a predefined skill-based taxonomy for tags [30]

preferences. Here we apply this approach to solve the problem of context-sensitive hub discovery in Expert Crowds. The first step (as shown in (7)) was to introduce predefined topics T' that are *query independent*.

To create topic-sensitive expertise profiles offline through mining that can be aggregated online, we propose the PageRank linearity theorem:

Theorem 1 (Linearity) *For any personalization vectors* p_1, p_2 *and weights* w_1, w_2 *with* $w_1 + w_2 = 1$, *the following equality holds*:

$$PPV(w_1 p_1 + w_2 p_2) = w_1 PPV(p_1) + w_2 PPV(p_2) \tag{8}$$

The above equality states that personalized PageRank vectors PPV can be composed as the weighted sum of PageRank vectors. The linearity theorem has been originally introduced by [15, 16] to create topic-sensitive importance scores for Webpages, but has not been applied in existing (related) approaches for expertise mining.

$$I^H(u; Q) = w_1 I^H(u; T_1) + w_2 I^H(u; T_2) \quad \text{with } Q = \{T_1, T_2\} \tag{9}$$

Equation (9) shows how to create query-dependent rankings established upon topic-sensitive expertise importance scores using (7) and (8).

7 Evaluation

We structure our evaluation in three sub-sections. First, we discuss a SOA-based testbed environment allowing us to simulate crowdsourcing scenarios. Second, we present performance experiments based on logged interaction data to test the efficiency of our online ranking approach considering concurrent expertise queries. Third, we analyze the effectiveness of our ranking approach based on synthetic interaction data gathered through simulations.

7.1 SOA testbed environment

Our evaluations were gathered using the features of the Genesis2 framework [17] and infrastructure services (e.g., logging) as introduced in [26]. Genesis2 has a management interface and a controllable runtime to deploy, simulate, and evaluate SOA designs and implementations. A collection of extensible elements for these environments are available such as models of services, clients, registries, and other SOA components. Each element can be set up individually with its own behavior, and steered

during execution of a test case. For the experiments in this work, we deployed Genesis2 Backends to the *Amazon Elastic Compute Cloud*[12]. We launched, depending on the amount of involved services instances, two or three *Community AMIs* of the type *High-Memory Extra Large Instance* (17.1 GB of memory) running a Linux OS. In the following, we provided each instance with the same Genesis2 Backend snapshot via mountable volumes from the *Elastic Block Store*. Finally, we deployed the following environment setup from a local Genesis2 Frontend. It included SOA-based HPS communities established by Genesis2 Web services equipped with simulated behavior and predefined relations to provide communication channels and instantiate communities. Services act like HPSs when delegating each other new tasks, processing tasks, redelegating existing tasks, or reporting tasks' progress status. Tasks are not delegated arbitrarily but must match the receivers capabilities. Therefore, they are tagged with three keywords one of which must match the picked receivers capabilities. Task processing and delegation decisions happen individually and in random time intervals (1–8 seconds). A hub combines capabilities of multiple communities by distributing tasks according to expertise areas of a given community (brokering of tasks). A hub avoids task processing and only forwards tasks. Finally, the deployed testbed environment has a variable number of services and participants per community. Consequently, the number of hubs varies depending on disparate expertise communities hubs are connected to (through knows relations).

7.2 Performance aspects

We performed several experiments to test the performance of our expertise ranking algorithms under varying characteristics such as number of nodes and expertise communities. Graph-based modeling and ranking algorithms have been implemented in C# and were deployed on our local (lab-based) blade servers accessible via a query Web service.

Hardware setup Our servers are equipped with Intel Xeon 3.2 GHz CPUs (quad core) and 10 GB RAM hardware. Interaction logs are managed by MySQL 5.0 databases. A client request pool (RP, see Fig. 9(a)) is created on a separate machine (Intel Core2 Duo CPU 2.50 GHz, 4 GB RAM) to perform parallel invocations of the query Web service. Clients are connected with the server via a local 100 MBit Ethernet.

Performance results The results for *online expertise queries*[13] are summarized in Fig. 9. The first experiment is based on a graph containing 198 nodes, 200 edges, and a total number of 10 distinct tags applied to interactions between nodes. The query service processing time for this environment is shown in Fig. 9(a). We vary the number of concurrent requests, denoted as RP, by launching multiple threads. Given a size of $\mathbf{RP = 50}$ and a total amount of # 100 requests to be processed, setting $RP = 100$ does not speed up the processing time of requests (i.e., the total

[12] Amazon EC2: http://aws.amazon.com/ec2/.

[13] Performance of the offline mining procedure as discussed previously is not shown here.

Experiment	# Req.	MIN	AVG	MAX	Total
1 (RP = 10)	50	3167	9083	10368	52543
	100	1669	9369	10576	101244
	200	1825	9211	10748	190647
1 (**RP = 50**)	50	1606	15955	29952	50762
	100	1482	27440	48562	98685
	200	1638	36313	47689	188573
1 (RP = 100)	50	1606	15955	29952	50762
	100	1544	28560	57501	105331
	200	1591	55185	100370	202394
2 (RP = 50)	100	2308	37891	63258	123677
3 (RP = 50)	100	2854	42041	67516	136266
4 (RP = 50)	100	3276	55058	84739	167778

(a) Processing time

Applied Tags in Exp. 4 ($n = 1029$ and communities = 230)	Frequ.
self-*	295
Robustness	306
Testbed	311
DB	314
Healing	321
Trust	322
WS	327
Autonomic	335
Similarity	341
Logging	353

(b) Tag frequency

Query ID	Query keywords	# Hubs	AVG proc. time
Q1	Robustness Logging	105	3993
Q2	Robustness Logging DB Testbed	134	3666
Q3	Robustness Logging DB Testbed Similarity	146	3478

(c) Queries in Exp. 4, number of discovered hubs and AVG processing time

Fig. 9 Processing statistics in simulated environment (in milliseconds)

time needed to process a number of requests). The average processing time increases by comparing RP = 100 and RP = 50 due to the overhead when handling a larger amount of requests simultaneously. Thus, we use RP = 50 for all further experiments.

Also, by processing a larger amount of requests, say # 200, the total processing linearly increases with the number of requests. We increased the number of nodes and interactions to understand the scalability of the query Web service under different conditions: experiment 2 with 579 nodes, experiment 3 comprising 774 nodes, and experiment 4 with 1029 nodes in the tested. HPSs in the testbed have been deployed equally on multiple hosts, e.g., 3 cloud hosts in experiment 4 to achieve scalability. In subsequent experiments detailed in Fig. 9 (experiments 2–4) we focus on a request pool with RP = 50 and 100 requests to be processed by the query service using different keywords (see Fig. 9(c)). To compare the experiments 1–4, we query the interaction graph using the keywords $Q = \{\text{Robustness, Logging}\}$. Increasing the number of nodes by a factor ≈ 3 (see experiment 1 and 2), the processing time goes up by 30%. Comparing the experiments 2 and 3 (node addition of $\approx 30\%$), the processing time increases by 10%. By comparing the experiments 3 and 4 (node addition of $\approx$30%), the processing time increases by 20%. Our experiments show that the online creation of expertise profiles based on different queries scales with larger testbeds linearly.

Furthermore, we used different query keywords as shown in Fig. 9(c). The number of discovered hubs increases if multiple keywords are used (see Fig. 9(b) for the set of available tags). The average processing time is not significantly influenced by the number of used keywords.

7.3 Quality of expertise rankings

Next, we analyze the effectiveness of our ranking approach based on synthetic interaction data since real interaction logs have not been available at time when performing this research.

Ranking evaluation metrics To study the results of our ranking approach, we define a set of ranking evaluation metrics in the following.

- The absolute ranking change $RC(u)$ returns the ranking change in a given query:

$$RC(u) = pos(u)_{BLR} - pos(u)_{CSR} \tag{10}$$

  BLR are the *base-line rankings* (here we use the standard HITS algorithm to obtain the base-line results) compared with CSR *context-sensitive rankings* using our with (cf. I^H as defined by (9)).
- We define *quality* $\mathcal{Q}(u)$ as the aggregated link weights of u's neighbors as:

$$\mathcal{Q}(u) = \sum_{(u,v)\in E} \sum_{(z,v)\in E} w_{zv} \tag{11}$$

  We have studied the calculation of link weights extensively in our previous work. For example, weights can be calculated based on trust metrics (e.g., delegation behavior) or link intensity [29]. Thus, we refer the interested reader to [30, 36].

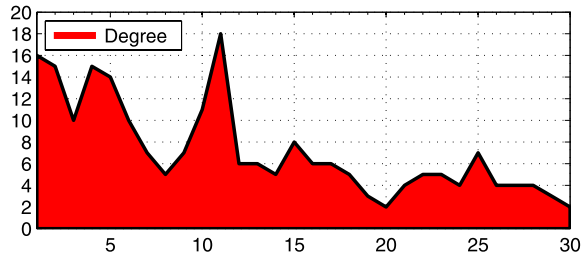
Algorithm parameters CSR are obtained based on both link weights and the assignment of preferences to personalization vectors $p'(u;T') = \sum_{(u,v)\in E} w_{vu}^{T'} p(v;T')$. In our experiments, preferences are assigned as follows:

$$p(v;T') = \begin{cases} 1 & \text{if } T'[v] \neq \texttt{null} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

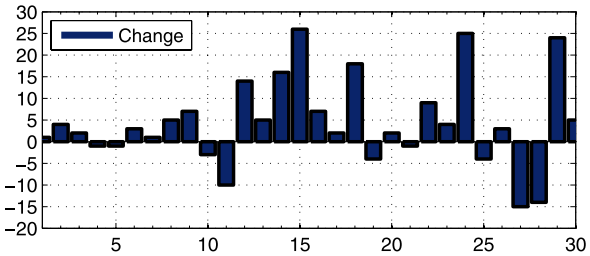
$T'[v]$ holds those users who have interacted with other users with focus on a particular topic T'. For example, users have performed tasks tagged with keywords related to T'. However, not only interaction-based profiles must be used to assign preferences. In addition, a user's manually maintained profile (e.g., FOAF) may be used to account for the user's interest (i.e., the authority v) in a given topic.

Ranking results To test the effectiveness of CSR, we performed experiments to study the impact of ratings and link weights on expert rankings. In the following figures, we show the top-30 ranked experts in a small-scale network (100 nodes). Results are sorted based on the *position* within the result set (see horizontal axis of Fig. 10 and column **Rank** in Fig. 11). Figure 10(a) shows the node degree and Fig. 10(b) ranking changes obtained by comparing CSR results with BLR (i.e., ranking results without accounting for metrics and ratings).

Figure 11 shows that all nodes within the top segment received high ratings given a high degree of links which is the desired property of CSR. Different levels of quality (i.e., quality mainly being 1 of ranked nodes between the positions 6–30) can be



(a) Node degree.



(b) Ranking change.

Fig. 10 Node degree and results of CSR/BLR comparison

Rank	Quality $\mathcal{Q}$	Rating
1	3.7	0.8
2	3.0	0.7
3	3.0	1.4
4	3.0	0.5
5	3.0	0.5
6	1.0	0.8
7	1.0	1.6
8	1.0	1.8
9	1.0	0.8
10	1.0	0.3
11	0.4	0.9
12	1.0	1.1
13	1.0	0.9
14	1.0	1.1
15	1.0	0.3

(a) Hub quality and ratings (1–15)

Rank	Quality $\mathcal{Q}$	Rating
16	1.0	1.3
17	1.0	0.9
18	1.0	0.9
19	1.0	2.7
20	1.0	4.1
21	1.0	1.5
22	1.0	1.0
23	1.0	0.9
24	1.0	1.1
25	1.0	0.2
26	1.0	1.3
27	1.0	1.2
28	1.0	0.8
29	1.0	1.5
30	1.0	2.5

(b) Hub quality and ratings (16–30)

Fig. 11 CSR ranking results: rank, quality, and ratings

explained by the impact of node degree on quality. Some nodes are demoted (negative ranking change) since the node (e.g., see 11) has received low ratings even though the node has a high degree of links. Nodes get promoted (positive ranking change) if they exhibit sufficient high ratings (see 15) or high quality (see 20 which was promoted a few positions only due to limited degree). Overall, CSR exhibit the demanded properties of promoting well-connected and rated hubs, thereby guaranteeing the discovery of reliable entry points to the Expert Crowd.

8 Conclusion and future work

The Web is evolving rapidly by allowing people to publish information and services. At the heart of this trend, interactions become increasingly complex and dynamic spanning both humans and software services. However, the transformation of how people collaborate and interact on the Web has been poorly leveraged in existing service-oriented architectures. The benefit of the presented approach is a seamless service-oriented infrastructure of human- and software services. The resulting service-oriented application needs to be flexible supporting adaptive interactions.

In this paper, we have motivated the need for adaptive interactions discussing an Expert Crowd scenario where people can register their skills and capabilities as services. Mixed service-oriented systems are open ecosystems comprising human- and software-based services. We discussed the HPS architecture enabling dynamic interactions in mixed service-oriented systems. We defined a novel expertise ranking approach that is based on context-aware interactions. Our ranking approach shows promising results, but needs to be further validated in real crowdsourcing environments. Our future work includes the deployment and evaluation of the implemented framework in the EU FP7 project COIN. The emphasis of COIN is to study new concepts and develop tools for supporting the collaboration and interoperability of networked enterprises. Also, we will further study the effectiveness and quality our expertise ranking approach in large-scale collaboration environments.

Acknowledgements The author would like to thank Schahram Dustdar, Dimka Karastoyanova and Frank Leymann for fruitful discussions on modeling dynamic interactions in human-centric service-oriented environments. Furthermore, we thank Lukasz Juszczyk for providing the Genesis2 framework, Harald Psaier for setting up the testbed experiments and Florian Skopik for discussions related to metrics and monitoring of mixed service-oriented systems.

References

1. Adams, M., ter Hofstede, A.H.M., Edmond, D., Aalst, W.M.P.V.D.: Worklets: a service-oriented implementation of dynamic flexibility in workflows. In: OTM Conferences (1), pp. 291–308 (2006)
2. Agichtein, E., Castillo, C., Donato, D., Gionis, A., Mishne, G.: Finding high-quality content in social media. In: WSDM, pp. 183–194. ACM, New York (2008)
3. Agrawal, A., et al.: WS-BPEL Extension for People (BPEL4People), V1.0 (2007)
4. Amend, M., et al.: Web Services Human Task (WS-HumanTask), Version 1.0 (2007)
5. Balthazard, P.A., Potter, R.E., Warren, J.: Expertise extraversion and group interaction styles as performance indicators in virtual teams: how do perceptions of it's performance get formed? DATA BASE **35**(1), 41–64 (2004)
6. Brabham, D.: Crowdsourcing as a model for problem solving: an introduction and cases. Convergence **14**(1), 75 (2008)
7. Breslin, J., Passant, A., Decker, S.: Social web applications in enterprise. Soc. Semantic Web **48**, 251–267 (2009)
8. Cugola, G., Nitto, E.D., Fuggetta, A., Ghezzi, C.: A framework for formalizing inconsistencies and deviations in human-centered systems. ACM Trans. Softw. Eng. Methodol. **5**(3), 191–230 (1996)
9. Dom, B., Eiron, I., Cozzi, A., Zhang, Y.: Graph-based ranking algorithms for e-mail expertise analysis. In: DMKD, pp. 42–48. ACM, New York (2003)
10. 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)
11. Easley, D., Kleinberg, J.: Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press, Cambridge (2010)

12. Garlan, D., Poladian, V., Schmerl, B.R., Sousa, J.P.: Task-based self-adaptation. In: WOSS, pp. 54–57 (2004)
13. Gentry, C., Ramzan, Z., Stubblebine, S.: Secure distributed human computation. In: EC'05, pp. 155–164. ACM, New York (2005)
14. Gyöngyi, Z., Molina, H.G., Pedersen, J.: Combating web spam with trustrank. In: VLDB, pp. 576–587. ACM, New York (2004)
15. Haveliwala, T.H.: Topic-sensitive pagerank. In: WWW, pp. 517–526. ACM, New York (2002)
16. Jeh, G., Widom, J.: Scaling personalized web search. In: WWW, pp. 271–279. ACM, New York (2003)
17. Juszczyk, L., Dustdar, S.: Script-based generation of dynamic testbeds for soa. In: ICWS '10. IEEE, New York (2010)
18. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM **46**(5), 604–632 (1999)
19. Kosorukoff, A., Goldberg, D.E.: Genetic algorithms for social innovation and creativity. Tech. rep., University of Illinois at Urbana-Champaign (2001)
20. Kumar, A., Aalst, W.M.P.V.D., Verbeek, E.: Dynamic work distribution in workflow management systems: How to balance quality and performance. J. Manag. Inf. Syst. **18**(3), 157–193 (2002)
21. Mendling, J., Ploesser, K., Strembeck, M.: Specifying separation of duty constraints in bpel4people processes. In: Business Information Systems. LNBIP, pp. 273–284. Springer, Berlin (2008)
22. Moody, P., Gruen, D., Muller, M.J., Tang, J., Moran, T.P.: Business activity patterns: a new model for collaborative business applications. IBM Syst. J. **45**(4), 683–694 (2006)
23. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: bringing order to the Web. Tech. rep., Stanford Digital Library Technologies Project (1998)
24. Panteli, N., Davison, R.: The role of subgroups in the communication patterns of global virtual teams. IEEE Trans. Prof. Commun. **48**(2), 191–200 (2005)
25. Petrie, C.: Plenty of room outside the firm. Internet Comput. **14**, 92–96 (2010)
26. Psaier, H., Juszczyk, L., Skopik, F., Schall, D., Dustdar, S.: Runtime behavior monitoring and self-adaptation in service-oriented systems. In: SASO. IEEE, New York (2010)
27. Russell, N., Aalst, W.M.P.V.D.: Evaluation of the bpel4people and ws-humantask extensions to ws-bpel 2.0 the workflow resource patterns. Tech. rep., BPM Center Brisbane/Eindhoven (2007)
28. Schall, D.: Human interactions in mixed systems—architecture, protocols, and algorithms. PhD thesis, Vienna University of Technology (2009)
29. Schall, D., Dustdar, S.: Dynamic context-sensitive pagerank for expertise mining. In: SocInfo. LNCS. Springer, Berlin (2010)
30. Schall, D., Skopik, F.: Mining and composition of emergent collectives in mixed service-oriented systems. In: CEC '10. IEEE, New York (2010)
31. Schall, D., Truong, H.L., Dustdar, S.: The human-provided services framework. In: EEE '08. IEEE, New York (2008)
32. Schall, D., Truong, H.L., Dustdar, S.: Unifying human and software services in Web-scale collaborations. IEEE Internet Comput. **12**(3), 62–68 (2008)
33. Schall, D., Dustdar, S., Blake, M.B.: Programming human and software-based web services. Computer **43**, 82–85 (2010)
34. Shahaf, D., Horvitz, E.: Generalized task markets for human and machine computation (2010)
35. Shetty, J., Adibi, J.: Discovering important nodes through graph entropy the case of enron email database. In: LinkKDD, pp. 74–81. ACM, New York (2005)
36. Skopik, F., Schall, D., Dustdar, S.: Modeling and mining of dynamic trust in complex service-oriented systems. Inf. Syst. **35**(7), 735–757 (2010)
37. Skopik, F., Schall, D., Dustdar, S.: Trustworthy interaction balancing in mixed service-oriented systems. In: SAC '10, pp. 799–806. ACM, New York (2010)
38. Su, Q., Pavlov, D., Chow, J.H., Baker, W.C.: Internet-scale collection of human-reviewed data. In: WWW '07, pp. 231–240. ACM, New York (2007)
39. Thomas, J., Paci, F., Bertino, E., Eugster, P.: User tasks and access control over Web services. In: ICWS'07, pp. 60–69. IEEE, New York (2007)
40. von Ahn, L.: Games with a purpose. IEEE Comput. **39**(6), 92–94 (2006)
41. Vukovic, M.: Crowdsourcing for enterprises. In: IEEE Congress on Services, pp. 686–692. IEEE, New York (2009)
42. Yang, J., Adamic, L., Ackerman, M.: Competing to share expertise: the taskcn knowledge sharing community. In: Int. Conf. on Weblogs and Social Media (2008)
43. Zhang, J., Ackerman, M.S., Adamic, L.: Expertise networks in online communities: structure and algorithms. In: WWW, pp. 221–230. ACM, New York (2007)