Expert Discovery and Interactions in Mixed Service-Oriented Systems

Daniel Schall, Florian Skopik, and Schahram Dustdar

Abstract—Web-based collaborations and processes have become essential in today's business environments. Such processes typically span interactions between people and services across globally distributed companies. Web services and SOA are the defacto technology to implement compositions of humans and services. The increasing complexity of compositions and the distribution of people and services require adaptive and context-aware interaction models. To support complex interaction scenarios, we introduce a mixed service-oriented system composed of both human-provided and Software-Based Services (SBSs) interacting to perform joint activities or to solve emerging problems. However, competencies of people evolve over time, thereby requiring approaches for the automated management of actor skills, reputation, and trust. Discovering the right actor in mixed service-oriented systems is challenging due to scale and temporary nature of collaborations. We present a novel approach addressing the need for flexible involvement of experts and knowledge workers in distributed collaborations. We argue that the automated inference of trust between members is a key factor for successful collaborations. Instead of following a security perspective on trust, we focus on dynamic trust in collaborative networks. We discuss Human-Provided Services (HPSs) and an approach for managing user preferences and network structures. HPS allows experts to offer their skills and capabilities as services that can be requested on demand. Our main contributions center around a context-sensitive trust-based algorithm called ExpertHITS inspired by the concept of hubs and authorities in web-based environments. ExpertHITS takes trust-relations and link properties in social networks into account to estimate the reputation of users.

Index Terms—Human-provided services, service-oriented expertise provisioning, crowdsourcing, social trust, hubs and authorities.

1 Introduction

Web services have paved the way for a new blend of composable systems. Services already play an important role in fulfilling organizations' business objectives because process stakeholders can design, implement, and execute business processes using web services and languages such as the Business Process Execution Language (BPEL) [1]. A broad range of services is increasingly found in open web-based platforms. Users and developers have the ability to use services in various applications because services offer well-defined, programmable, interfaces. In process-centric collaboration, a top-down approach is typically taken by defining process activities and tasks prior to deploying and executing the process. Before creating the model, the designer must fully understand each step in the process. Flexibility in such composition models is limited since unexpected changes require remodeling of the process. Such changes may cause exceptions, disrupting the normal execution of the process. It is important to support *adaptivity* in collaborations and compositions. An important role toward adaptive processes is the ability to support the execution of ad hoc activities and flexibility in human interactions to react to unexpected events. While the process-centric collaboration approach follows a top-down methodology in modeling flows, ad hoc flows in flexible collaborations emerge at runtime. A runtime environment constraints the execution of flows. Such constraints are, for example, the availability of resources, services, and people.

In this paper, we utilize Human-Provided Services (HPSs) [2] enabling flexible interactions in service-oriented systems. We discuss the discovery and interactions in *mixed service oriented systems* [3] comprising HPS and software-based services (SBS). Experts offer their skills and capabilities as HPS that can be requested on demand. In this work, we present the following key contributions: 1) estimation of user reputation based on a context-sensitive algorithm. Our approach, called *ExpertHITS*, is based on the concept of hubs and authorities in web-based environments. 2) An approach for community reputation (the hub-expertise of users) influenced by trust relations. Dynamic link weights are based on trust and user rating influenced by the query context. ExpertHITS is calculated online, thus fully personalized based on the expert-requester's preferences (i.e., the demanded set of skills). 3) Implementation and evaluation of our approach demonstrating scalability and effectiveness of our proposed algorithm.

This paper is organized as follows: in Section 2, we present a motivating example detailing the need for flexible interaction models. Section 3 introduces the fundamental idea and basic concepts of ExpertHITS. A detailed description of the ExpertHITS discovery approach is provided in Section 4 followed by an overview of our implemented architecture in Section 5. In Section 6, we discuss ranking experiments using ExpertHITS. Related work is presented in Section 7 and finally we conclude the paper in Section 8.

• *The authors are with the Distributed Systems Group, Information Systems Institute, Vienna University of Technology, Argentinierstrasse 8/184-1, A-1040 Wien, Austria.*
E-mail: {schall, skopik, dustdar}@infosys.tuwien.ac.at.

Manuscript received 25 Mar. 2010; revised 6 Aug. 2010; accepted 25 Dec. 2010; published online 2 Feb. 2011.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2010-03-0026. Digital Object Identifier no. 10.1109/TSC.2011.2.

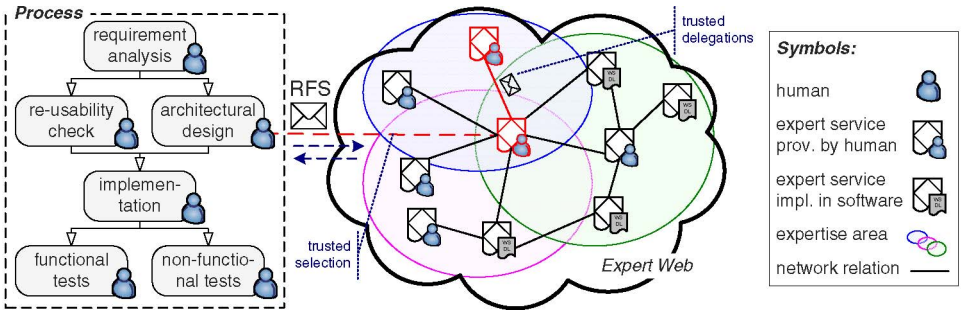


Fig. 1. Discovering and including experts for online help and support.

2 Motivating Scenario

A motivating use case for discovering experts on demand and flexible interaction support is depicted in Fig. 1. The process model may be composed of single tasks assigned to responsible persons, describing the steps needed to produce a software module. After finishing a common requirements analysis, an engineer evaluates the reusability of existing work, while a software architect designs the framework. The implementation task is carried out by a software developer; and two software testers evaluate the prototype implementation with respect to functional properties (e.g., coverage of requirements) and nonfunctional properties (e.g., performance and memory consumption). We assume that the task owners in this process exchange only electronic files and interact by using communication tools.

While various languages and techniques for modeling such processes already exist, for example BPEL, we focus on another aspect in this scenario: *discovery and interactions with trusted experts*. A language such as BPEL demands for the precise definition of flows and input/output data. However, even in carefully planned processes with human participation, for example modeled as BPEL4People activities [4], ad hoc interactions and adaptations are required due to the complexity of human tasks, people's individual understanding, and unpredictable events. According to Fig. 1, the software architect receives the requirement analysis document from a preceding step. But if people have not yet worked jointly on similar tasks, it is likely that they need to set up a meeting for discussing relevant information and process artifacts. Personal meetings may be time and cost intensive, especially in cases where people belong to different geographically distributed organizational units. Various Web 2.0 technologies, including forums, Wiki pages and text chats, provide well-proven support for online work in collaborative environments. Several challenges remain unsolved that are addressed in this paper.

1. Who is the right expert that can assist in solving problems which people face while participating in the process?
2. How can third parties (experts) be contacted and informed about the current situation and how can they easily be involved in ongoing collaborations?
3. Based on which decision are experts selected, which information needs to be exchanged, and how can such scenarios be supported in service-oriented systems?
4. How can one support trusted interactions in such dynamically changing environments?

2.1 Manual Discovery

Discovering support typically requires the *expert seeker* to start an inquiry for an expert by asking other people for their opinion or to provide recommendations; who is able to help *and* who is trustworthy. Recommendations are typically performed by asking friends or colleagues who may have faced similar problems in the past. Once the expert seeker has identified a potential candidate, contact can be established using standard tools such as e-mail, instant messaging, or telephone. People tend to know reputable and trusted experts in small environments and also what data needs to be exchanged to solve a particular problem. The drawbacks are that people need extensive knowledge about the skills of colleagues and internal structures of the organization (e.g., the expertises of people in other departments). The manual discovery of an expert becomes a very daunting task when the number of people increases, for example, the web or large-scale enterprises. With a huge set of people available, given that reputation, skills, and trust between people changes dynamically, discovering experts and support becomes a major undertaking that cannot be performed any longer in a manual way.

2.2 The Expert Web

Today's information and communication technologies cannot fully address the above mentioned challenges. Here, we propose the *Expert Web* consisting of connected experts that provide help and support in a service-oriented manner. Examples are *crowdsourcing* applications in enterprise environments or open Internet-based platforms. These online platforms distribute problem-solving tasks among a group of humans [5]. The members of the Expert Web are either *humans*, such as company employees offering help as online support services or can in some cases be provided as software-based services. Applied to enterprise scenarios, such a network of experts, spanning various organizational units, can be consulted for efficient discovery of available support. The expert seekers, for example the software engineers or architect in our use case, send *Requests For Support* (RFSs). Experts may also delegate RFSs to other experts in the network, for example when they are overloaded or not able to provide satisfying responses. Following this way, not only users of the expert network establish trust in experts, but also trust relations between experts emerge.

3 Expertise Model

In this section, we will detail the basic concepts enabling the discovery of experts. Our approach is based on the following idea: given a search query containing the set of

relevant skills, who is the expert 1) satisfying these demanded skills and 2) how well is this expert connected to other people having similar expertise. From the Expert Web's point of view, finding the right expert by performing skill matching is not sufficient. We also need to consider whether the expert will be able to *delegate* RFSs to other peers in the Expert Web.

3.1 Trust Emergence

Traditional rating and ranking models usually neglect social aspects and individual preferences. However, actors in the Expert Web may not be compatible with respect to working style and behavior. As a consequence, social aspects need to be considered and require dynamic interaction models. In this paper, we focus on *social trust* to support and guide delegations of requests. In contrast to a common security perspective, social trust refers to the flexible interpretation of previous collaboration behavior [6], [7], [8] and the similarity of dynamically adapting interests [9], [10]. Especially in collaborative environments, where users are exposed to higher risks than in common social network scenarios [11], and where business is at stake, considering social trust is essential to effectively guide human interactions. Relying on these works, we define trust in the Expert Web as follows:

> *Trust reflects the expectation one expert has about another's future behavior to perform delegated RFSs dependably, securely, and reliably based on experiences collected from previous interactions.*

3.2 Hubs and Authorities

In this work, we utilize the concept of hubs and authorities in web-based environments. This concept has been introduced by Kleinberg [12] to rank web pages in search queries using the Hyperlink-Induced Topic Search (HITS algorithm). 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 scenario, a member of the Expert Web may receive an RFS and delegate work to some other peer in the network (characterizing hubs in the network). 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 Web, authorities give feedback using rating mechanism (e.g., a number on the scale from 1 to 5) to indicate their satisfaction—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. On the other hand, delegation of work is strongly influenced by *trust*, for example, whether the initial receiver of the RFS (hub within the Expert Web) expects that the peer will process work in a reliable and timely manner. RFS receivers need to be trusted by influential hubs that are highly rated in order to be recognized as authoritative peers in the Expert Web. Notice, hub and authority scores are available for each member in the network. Thus, a member may act as a hub *and* authority by processing or delegating tasks.

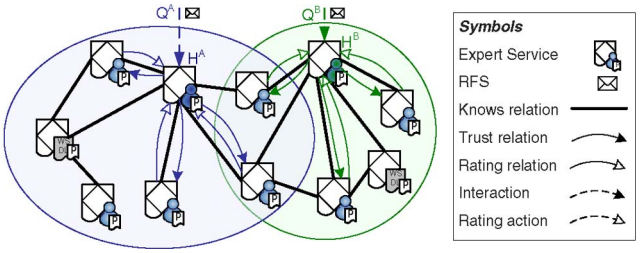


Fig. 2. Hubs with different areas of expertise.

3.3 Personalized Expert Queries

Following the previous discussion, we define this concept as *expert hubs* that are well connected (i.e., social network structure *and* connections based on joint collaborations) given a particular query context. Delegation is important in flexible, interaction-based systems because expert hubs will attract many RFSs over time, thus presenting bottlenecks in terms of processing or delegating RFSs. On the other side, being a hub in the Expert Web also means that a person *knows* many other experts in similar fields of interest. We argue that the likelihood of being able to delegate RFSs to other experts greatly increases depending on the *hubness* of a person arising from being a member in expert areas (e.g., communities or interest groups). The major challenge in this scenario is that hubness needs to be calculated on demand based on a given query. A query determines the *context* specified as the set of relevant skills. Let us formalize this concept by discussing two scenarios as shown in Fig. 2.

The following steps in Algorithm 1 outline our approach at a high level, which will be detailed in the following sections. First, *matching* is performed based on the query context. In this step, a set of skills is specified to retrieve qualified users. Second, expert hubs are discovered using link and interaction information. We will further elaborate on this concept in the following sections.

Algorithm 1. Outline discovery approach
Input: Given a query context Q to discover expert hubs
1) Find users matching demanded set of skills.
2) Calculate hub-expertise of users given query context Q.
   a) For each user calculate hub score in Q.
   b) For each user calculate authority score in Q.
3) Rank users by hub score.
Output: Ranked experts in Q

First, a query (see Q^A or Q^B) is specified either manually by a (human) expert seeker or derived automatically from a given process context, 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. There are two influencing factors, i.e., relations, determining hub- and authority scores: 1) how much hubs *trust* authorities (depicted as filled arrows from hubs to authorities) and 2) *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

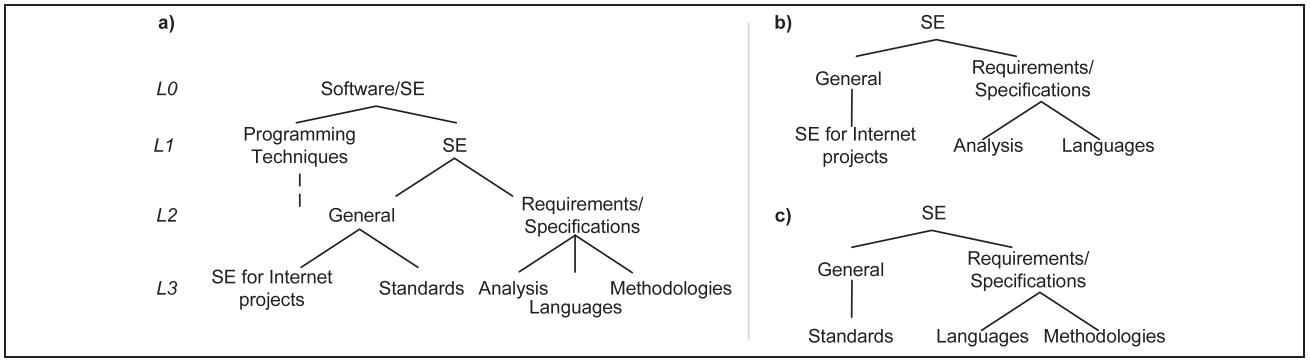


Fig. 3. (a) Excerpt skill taxonomy in the software engineering domain. (b) Illustrates an example query specifying the demanded skills as formulated by an expert seeker. (c) Gives an example of a skill profile.

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 ExpertHITS (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 here, but explained in detail in the next section. This approach provides a powerful tool for expert discovery because reputation (for example, within communities) is expressed as hub-expertise by weighting trust relations in personalized scopes (through the query context) and feedback ratings. Also, we believe that our approach is difficult to cheat on because hub-expertise is influenced by how well hubs are connected to multiple authorities who propagate their expertise back to hubs.

3.4 Skill Model

Our proposed skill model is based on the ACM Computing Classification System [13]. This simple model is sufficient to serve as a basic classification scheme for skills in the computer science domain which is well aligned with the requirements of the previously introduced motivating scenario. More advanced skill or competency models (e.g., ontological systems [14]) are not within the scope of this work. In Fig. 3, we show an excerpt of a taxonomy that can be used to classify skills in, for example, the software engineering domain. However, our ranking model can be extended to other domains as well by using different taxonomies. The basic idea of our approach is to define different weights for each level in the tree—see Fig. 3a. The top-most level (the root node) has the lowest weight since top levels in the skill tree denote broad areas of expertise. The weights increase depending on the tree depth because lower levels contain fine-grained skill and expertise information (specialism). We define the set of levels $L = \{L_0, L_1, \ldots, L_n\}$ with $\sum_{i:1\ldots n} w_{L_i} = 1$. Note, having all level weights sum up to 1 means that there is a mutual dependency between weights. All nodes in the skill tree that do not have successor nodes are called *leaf nodes*.

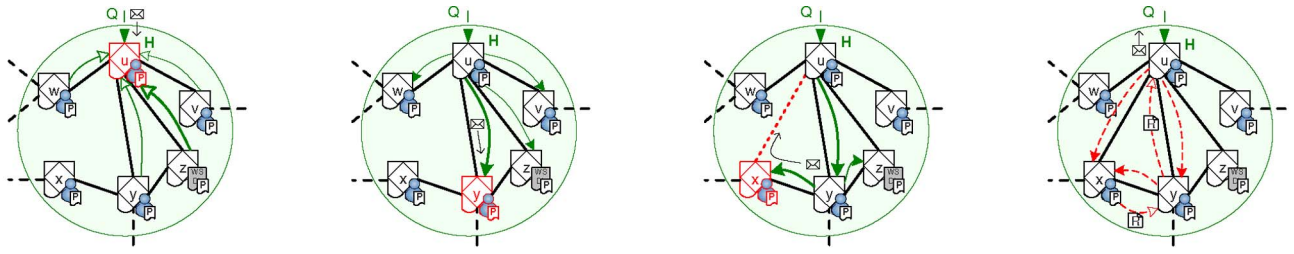
A subset of the tree may be selected by a query Q to discover experts. Thus, the provided query needs to be matched against user profiles to determine how well users match the demanded set of skills. Each node in the tree is called *skill property*. We introduce *query preferences* enabling the expert seeker to express *strong* or *weak* matching preferences of skill properties and *optional* ("nice to have") properties. A strong preference might denote that the expert seeker requires the user to have certain skill properties, whereas weak preferences would express that the expert *should* have as many skill properties. Optional means that higher preferences are given to those experts that have a higher degree of similarity with a set of optional skill properties. In Fig. 3b, an example query is shown that can be formulated as, for example, `SE for Internet projects` and `[Requirements [Analysis][Language]]` specified within the skill subtree `SE`. For a given user profile, e.g., Fig. 3c, matching is performed according to different preferences. Specifying strong preferences for the first part of the query expression would mean no overlap between specified skill property `[SE for Internet projects]` and the user expertise in `Standards`, whereas a weak preference expresses the need to have *some* specialism in a given area (i.e., `[SE [General]]`). Considering the second part of the query expression, strong preferences are calculated as the overlap similarity between the set of skill properties `[Analysis][Language]` and `[Languages][Methodologies]`.

4 Expert Discovery

In this section, we detail our discovery approach by defining a matching procedure and an algorithm for calculating ExpertHITS. An important aspect of the presented approach is to select interactions based on (query) context information. We assume that each interaction (e.g., based on delegated RFSs) is associated with context *tags* based on the skill taxonomy.

4.1 Skill Matching Algorithm

The basic approach is to use a metric to calculate the overlap of two sets A and B. A straightforward way to define overlap similarity is $\frac{|A \cap B|}{n}$ [15]. In this work, we present an algorithm supporting the notion of *strong*, *weak*, and *optional* matching preferences through alternate approaches for calculating overlap similarities of sets of properties. These preferences have impact on matching of skill properties on lower levels. As mentioned before, all nodes in the skill tree that do not have successor nodes are called *leaf nodes*. For simplicity, we do not consider



(a) Discovery of expert hub. (b) Trusted selection of authority. (c) Delegation of RFS. (d) RFS reply and rating.

Fig. 4. ExpertHITS discovery model, advanced interaction patterns, and feedback ratings.

unbalanced trees or complicated branching structures. Matches at leaf-node level have higher impact on similarities due to the following property: weights increase depending on the tree depth such that $w_{L_0} < w_{L_1} < \cdots < w_{L_n}$. In the following, we will derive an algorithm for matching *elements* which may depict interaction data (tagged RFS-based interactions) and user profiles holding skill information. The function `childN` returns the set of child nodes associated with a given property in the query or the global skill taxonomy tree G_T. $|P(L_i)|$ denotes the number of properties in L_i.

The numerator of the set metric (i.e., $|A \cap B|$) is calculated by the Steps 1-3. The set similarity is divided by the number n based on different matching preferences. Minimum match (Step 1) means that user profiles and interaction data matching the query root node are taken into account for subsequent ranking. For example, the root node of a query in Fig. 3b is `[SE]`. All profiles and interaction data that have been tagged with elements underneath `[SE]` will then be considered for matching and ranking. As shown in Algorithm 2 (Step 4), n is appended to the matching result to obtain similarity scores based on the different preferences *strong*, *weak*, or *optional* as defined in the following:

$$n = \begin{cases} |\texttt{childN}(q_p(L_i))| \cup |\texttt{childN}(e_p(L_i))|, & \text{(a)} \\ |\texttt{childN}(G_T(L_i))|, & \text{(b)} \\ |P(L_i)|. & \text{(c)} \end{cases} \quad (1)$$

Condition (a) is satisfied if *strong* preferences are selected, (b) if *weak* or *optional* and (c) otherwise.

Algorithm 2. Topic tree matching algorithm.

Input: Given a query context Q containing a set of properties q_p and elements E

Compute:

1) Get all elements $e \in E' \subseteq E$ whose properties provide a minimum match of topics.
2) Extract topic tree matching query root node.
3) Iterate through each level and calculate overlap similarity of property in query at current level i. Given current property $q_p(L_i)$:
   a) If $\texttt{childN}(q_p(L_i)) \neq \emptyset$ then do $|\texttt{childN}(q_p(L_i))| \cap |\texttt{childN}(e_p(L_i))|$.
   b) If $\texttt{childN}(q_p(L_i)) = \emptyset$ and weak preference then use $|\texttt{childN}(e_p(L_i))|$.
   c) Otherwise perform $q_p(L_i) \cap e_p(L_i)$.
4) Divide similarity by n and append score with w_{L_i} to previous score sum.

Output: Ranked elements according to similarity

4.2 Discovery of Expert Hubs

Here, we present our expert discovery algorithm that is influenced by *social trust* and *rating mechanisms*. Our algorithm accounts for context information and weighted links between actors. Context is utilized by considering relations of experts in different scopes. Thus, the goal of our algorithm is to find hubs with respect to context. In the following, we discuss the basic flow of *actions* in the Expert Web. The actions include delegation of RFSs, ratings of requests, and advanced delegation patterns. First, we discuss the discovery and selection of expert hubs and authorities (Figs. 4a and 4b) followed by the definition of delegation patterns and ratings (Fig. 4c and 4d).

Hub discovery. Let us assume that a query Q is specified to discover an expert hub (see Fig. 4a). Every query influences the set of prior ratings (arrows pointing to u) and interactions (i.e., actions) that need to be considered when calculating hub- and authority scores. Consider the query context Q comprising actions related to the demanded set of skills. Notice, the previously defined matching algorithm is used to select related actions. In this case, u has been discovered as the entry point denoted as H^B.

Delegation actions. In Fig. 4a, user u receives an RFS issued toward the Expert Web. Since u represents the hub expert, u may decide to delegate the request to one of its neighbors v, w, y, z, which can be discovered through `knows` relations[1] (Fig. 4b). In our Expert Web application scenario, `knows` is a bidirectional relation between users. A relation becomes active if both users acknowledge that they are connected to each other (v `knows` u *and* u `knows` v), a simple yet effective mechanism to support growth in social networks (e.g., newcomers and bootstrapping problem) while preserving user control. Notice, `knows` relations do not contain context related information such as tags. The context of interactions is derived from delegated RFSs (tags or type of RFS classified by using the skill taxonomy). To support growth in networks (e.g., how can newcomers become members of communities), we introduce an advanced interaction pattern in the Expert Web depicted by Fig. 4c.

Triadic delegation pattern. An authority may need to delegate an RFS received from the hub to somebody who is known to the authority, but not the hub. This pattern is shown in Fig. 4c. Hub u delegates an RFS to y, which is in turn delegated to x and, thus, being responsible for processing the RFS.

1. The knows property in FOAF profiles can be used for discovering social relations; see http://xmlns.com/foaf/spec.

If ties (i.e., through knows relations) between the pairs (u, y) and (y, x) exist, it is likely that x will attempt to establish a connection to u as well. This concept is known as *triadic closure* in social networks [16] and can be applied to support interaction patterns in service-oriented systems. The *triadic interaction pattern* enables x to connect to hubs and helps increasing its authority in the Expert Web. As mentioned previously, knows is a bidirectional connection and needs to be acknowledged by u.

Rating procedure. An RFS is delivered back to the expert seeker from the Expert Web; i.e., the selected hub u depicted in Fig. 4d. The main argument of our model is to determine those hubs that are well embedded in expertise areas (e.g., communities). Thus, the hub-score should be influenced by feedback ratings denoting the level of satisfaction of authorities. Ratings are subjective opinions of authorities with respect to RFSs received from hubs, i.e., whether RFSs fit the expertise area of authorities. In the final step, RFSs are rated (see dashed open arrows) expressing the precision of received delegations. Indeed, such ratings are also given to RFSs traversing the Expert Web through triad delegation patterns. Given the scenario in Fig. 4d, automatic propagation of ratings (e.g., if a delegated RFS from u to y was further delegated to x) is currently not considered in our model. Thus, x rates the RFS received from y and similarly y from u.

Trust updates. Trust relations, based on experts' behavior are periodically updated with recent interaction data. Those interactions (reflected by filled dashed arrows) are aggregated to interaction metrics that are interpreted by predefined rules to infer trust.

4.3 ExpertHITS Model

In this section, we discuss the formal model for our proposed expertise ranking algorithm consisting of two components 1) hub score $H(u; Q)$ of user u in query context Q and 2) authority score $A(v; Q)$ of user v in the same query context Q

$$H(u; Q) \leftarrow \sum_{v \in \text{knows}(u)} w^{Q}_{vu} A(v; Q), \tag{2}$$

$$A(v; Q) \leftarrow \sum_{u \in \text{knows}(v)} w^{Q}_{uv} H(u; Q). \tag{3}$$

- $H(u; Q)$: Hub score of user u acting as a reliable entry point to the Expert Web brokering RFSs to authoritative users. Hubs are identified based on the demanded expertise, knows relations connecting u to other experts *and* feedback ratings received from prior delegations.
- $A(v; Q)$: Authority score of user v. Authorities are skilled users (experts) that are connected to influential hubs. In our model, authority means that users process RFSs received from hubs in a reliable, trustworthy manner.
- w^{Q}_{uv}: Trust influences the delegation behavior of hubs by selecting authorities based the success of interactions; in our example successfully delegated and processed RFSs.

TABLE 1
Metrics Utilized for Trust Inference

Metric Name	Range	Description
Availability	[0,100]	ratio of accepted to received RFSs
Response time	[0,96]	average response time in hours
Success rate	[0,100]	amount of successfully processed RFSs
Experience	$[0,\infty]$	number of RFSs processed
RFS reciprocity	[-1,1]	ratio of processed to sent RFSs
Manual reward	[0,5]	optional manually assigned scores
Costs	[0,5]	price for processing RFSs

- w^{Q}_{vu}: Denotes the connection strength of an authority v to hub u. The weight can be calculated using information from ratings given by v to RFSs received from u.

Considering the loosely structured and dynamically bound Expert Web example, it is important to derive metrics that can be used to infer trust relations in an automated manner. The weight w^{Q}_{uv} can be interpreted as how much u trusts v in processing RFSs in a reliable manner. Specifically, experts' behavior in terms of reliability, availability, or RFS processing successes, are periodically updated with recent interaction data. The weight is calculated as

$$w^{Q}_{uv} = \frac{\text{successful del. from } u \text{ to } v}{\sum_{w \in \text{knows}(u)} \text{successful del. from } u \text{ to } w}. \tag{4}$$

4.4 Metric Calculation

Metrics describe the interaction behavior and dynamically changing properties of actors. Interactions such as delegations are aggregated to metrics that are interpreted by rules to infer trust (see [8] for the detailed mechanisms). Currently, we account for the metrics described in Table 1 for trust interpretation upon logged SOAP calls in the Expert Web scenario. Note, as described before, these metrics are determined for particular query scopes; i.e., based on a subset of interactions that meet certain constraints. The availability of a service, either provided by humans or implemented in software, can be high in one query context, but much lower in another one. Furthermore, these metrics are calculated for each directed relation between pairs of network members. An actor u might serve v reliably, but not a third party w.

Our approach relies on mining of metrics, thus, values are not manually entered but are frequently updated by the system. This enables collaboration partners to keep track of the dynamics in highly flexible large-scale networks. Besides interaction behavior in terms of reliability or responsiveness, also context-sensitive expertise mining can be conducted. This approach is explained in detail [3].

We accounted for the *average response time* t_r (5) of a service and its *success rate* sr (6). These are typical metrics for an *emergency help and support environment*, where fast and reliable support is absolutely required, but costs can be neglected. We assume, similar complexity of requests for support in a context Q, thus different RFSs require comparable efforts from services (similar to a traditional Internet forum).

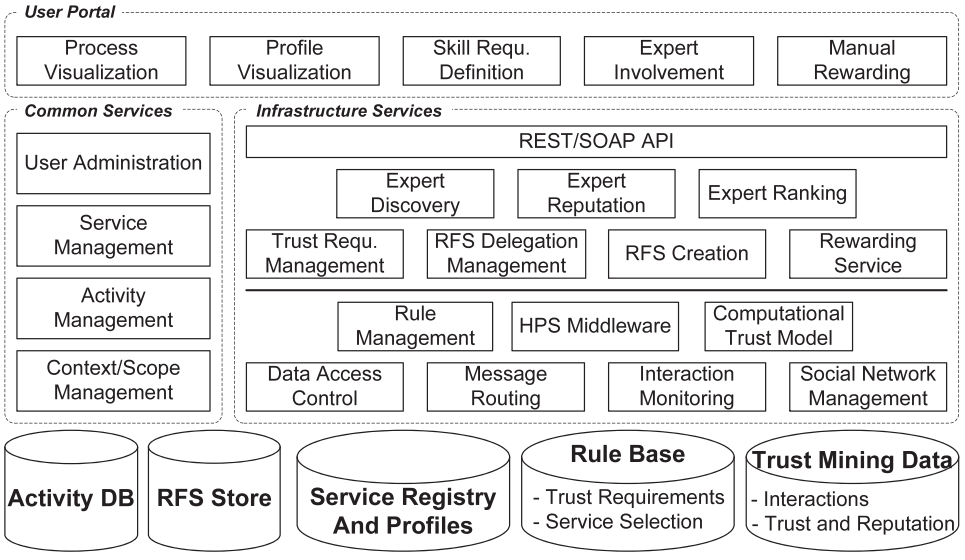


Fig. 5. System architecture enabling trusted online help and support in the expert web.

The response time is calculated as the duration between sending (or delegating) a request (t_{send}) to a service and receiving the corresponding response ($t_{receive}$), averaged over all processed RFSs. Unique IDs of calls (see SOAP header in Listing 3) enable sophisticated message correlation to identify corresponding messages.

$$t_r^Q = \frac{\sum_{rfs \in RFS} (t_{receive}(rfs) - t_{send}(rfs))}{|RFS|}. \quad (5)$$

An RFS is considered successfully processed ($sRFS$) if leading to a result before a predefined deadline, otherwise it fails ($fRFS$)

$$sr^Q = \frac{num(sRFS)}{num(sRFS) + num(fRFS)}. \quad (6)$$

5 Architecture and Implementation

We depict the overview of the architecture in Fig. 5 adopting parts from our previous work, in particular the HPS framework [2], [3] and the VieTE trust emergence framework [8]. The block on the left side contains common services from activity-centric collaboration systems. On the right side, the ExpertHITS services are shown. The lower layer contains services supporting fundamental concepts, including data access, message routing and interaction logging; on higher level services for trust management, RFS creation, and expert ranking are shown. These services are utilized via SOAP- and REST-based web service interfaces from a user portal. We outline some (exemplary) implementation details, focusing on realizing the introduced concepts including the RFS model, and skill requirements for HPS discovery and selection.

Activity management. Activities are structures to describe work and its goals, as well as participating actors, used resources, and produced project artifacts.

Interaction monitoring. Interactions are either captured by interaction sensors, included in infrastructure services, or external access layers. An example for the first case is the activity management service that notifies a logging service about activity delegations and assignments. In the second case, service invocations via SOAP are routed over an access layer that captures the SOAP messages. Interactions are periodically analyzed to calculate higher level metrics. 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.

Computational trust model. A domain expert configures certain properties of the trust inference process that are applied for all participants of the network. For instance, she/he defines meaningful trust scopes in the given domain and business area, configures available metric calculation plugins that provide the metrics for personal trust rules, and sets up the general trust model behavior, such as temporal constraints for interaction analysis and endpoints of logging facilities.

Network management and provisioning. This component enables the registration of human- and software services, including their individual profiles. All registered actors build the set of nodes of a trust graph (see the *Web of Trust* in [6]). We support the discovery of actors during ongoing collaborations (similar to a web service registry), relying on actor capabilities (profiles), and periodically inferred metrics (interaction-, similarity-, collaboration-, and trust metrics).

5.1 Human Provided Services in the Expert Web

Previously, we discussed interaction scenarios in mixed service-oriented collaboration environments. These interactions are governed by dynamics as new HPSs can be registered and flows of activities might change (e.g., delegation patterns) due to actor preferences, trust, and reputation. Activities are used for different purposes: 1) people use activities to structure collaborations in a flexible manner and 2) activities enable users to define HPSs [3]. As stated before, interactions may take place between humans (using HPSs) or between (software) services and

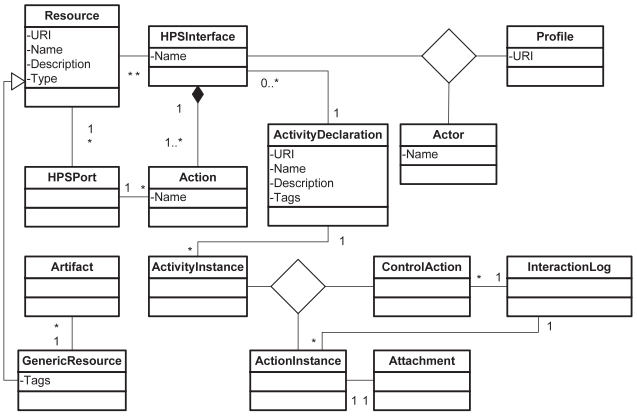


Fig. 6. Overview of HPS activity model.

humans. Both scenarios are enabled by using web services technology. The HPS activity model is shown in Fig. 6.

- 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.
- The `HPSInterface` relates to an ActivityDeclaration. Name in the `HPSInterface` depicts the HPSs name, for example, a *support service*. The `HPSInterface` (description) is very similar to the description of "conventional" software services. Essentially, we perform a simple mapping to depict declarations as web service descriptions (e.g., using WSDL).
- 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. 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 `HPSPort` relates to a set of resources (e.g., typed messages), which are used in certain Actions.

The following concepts describe activity and HPS-centric interactions at runtime.

- An `ActivityInstance` represents an actual work item. Each instance corresponds to a declaration. Instances represent the context of interactions.
- An `ActionInstance` is connected to an `ActivityInstance`. An `Attachment` is generic to associate XML documents, for example, XML messages that are exchanged between HPSs, and other content-types with an `ActionInstance`. Both `ControlAction` and `ActionInstance` are used at runtime. A `ControlAction`, however, depicts common action types in human collaboration (e.g., coordination, communication, and execution actions). Each action, `ControlAction` as well as `ActionInstance`, is logged to keep a history of interactions. The `InteractionLog` captures traces of interactions.

An excerpt of the RFS schema definitions is shown in Listings 1 defining complex data structures.

Listing 1. RFS schema definition.

```
<xsd:schema tns="http://myhps.org/rfs">
 <xsd:complexType name="GenericResource">
  <xsd:sequence>
   <xsd:element name="Location" type="xsd:anyURI"/>
   <xsd:element name="Expires" type="xsd:dateTime"/>
  <xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="Request">
  <xsd:sequence>
    <xsd:element name="subject" type="xsd:string"/>
    <xsd:element name="requ" type="xsd:string"/>
    <xsd:element name="resource" type="GenericResource"/>
    <xsd:element name="comments" type="xsd:string"/>
    <xsd:element name="keywords" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="SupportRequest" type="Request"/>
 <xsd:element name="AckSupportRequest" type="xsd:string"/>
 <xsd:element name="GetSupportReply" type="xsd:string"/>
 <!-- reply details omitted -->
 <xsd:element name="SupportReply" type="Reply"/>
</xsd:schema>
```

`Request` defines the structure of an RFS (here we show a simplified example). A `Reply` is the corresponding RFS response (we omitted the actual XML defintion). The protocol (at the technical HPS middleware level) is asynchronous allowing RFSs to be stored, retrieved, and processed. For that purpose we implemented a middleware service (HPS Access Layer—HAL) which dispatches and routes RFSs. `GetSupport` depicts a WSDL message corresponding to the RFS `SupportRequest`. From the user's point of view XML Forms (XForms[2]) are used to render graphical user interfaces.

Listing 2 shows the binding of the HPS WSDL to the (HPS) infrastructure services.

Listing 2. HPS WSDL binding.

```
<!-- excerpt wsdl interface -->
<wsdl:portType name="HPSSupportPortType">
 <wsdl:operation name="GetSupport">
  <wsdl:input xmlns="http://.../2006/05/addressing/wsdl"
   message="GetSupport" wsaw:Action="urn:GetSupport">
  </wsdl:input>
  <wsdl:output message="AckSupportRequest" />
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HALSOAPBinding"
 type="HPSSupportPortType">
 <soap:binding style="document"
  transport="http://xmlsoap.org/soap/http"/>
</wsdl:binding>
```

Upon receiving such a request, HAL generates a session identifier contained in the output message `AckSupportRequest`. A notification is sent to the requester (assuming a callback destination or notification endpoint has been provided) to deliver RFS status updates for example; processed RFSs can be retrieved via `GetSupportReply`. The detailed notification mechanism can be found in [3].

2. XML Forms: http://www.w3.org/MarkUp/Forms.

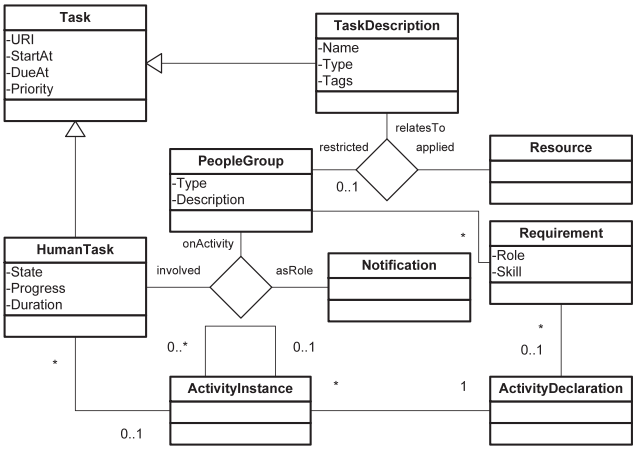


Fig. 7. Overview HPS task model.

Activities can be structured as hierarchies using `parent` and `child` relations. Child activities specify the details with respect to the (sub)steps in collaborations, for example, subactivities 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. For example, an activity (a_1, u) performed by u can be delegated to multiple actors v, w (as discussed previously) resulting in the creation of a set of subactivities $\{(a_2, v), (a_3, w)\}$ to support patterns such as the *4-eyes principle*. The task model is shown in Fig. 7.

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 subactivities. 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 subactivities) has been completed. This model is well aligned with the WS-HT specification [17]. Moreover, functional properties can be associated with `Activity-Declarations`, depicted as `Requirement` in Fig. 7; for example, role models controlling whether users are allowed to work on activities. 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 runtime context. Constraints are, for example, the 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.

5.2 Interaction Monitoring and Logging

The HPS Access Layer logs each service interaction (request and response message) through a logging service. RFSs and their responses, exchanged between community members, are modeled as traditional SOAP calls, but with various header extensions, as shown in Listing 3. These header extensions include for example addressing and message correlation mechanisms.

The most important extensions are

- `Timestamp` capture the actual creation of the message and is used to calculate temporal interaction metrics, such as average response times.
- `Delegation` holds parameters that influence delegation behavior, such as the number of subsequent delegations `numHops` (to avoid circulating RFSs) and hard `deadlines`.
- `Activity uri` describes the context of interactions.
- `MessageID` enables message correlation, i.e., to properly match requests and responses.
- `WS-Addressing` tags, besides MessageID, are used to route RFSs through the network.

Listing 3. Simplified RFS via SOAP example.

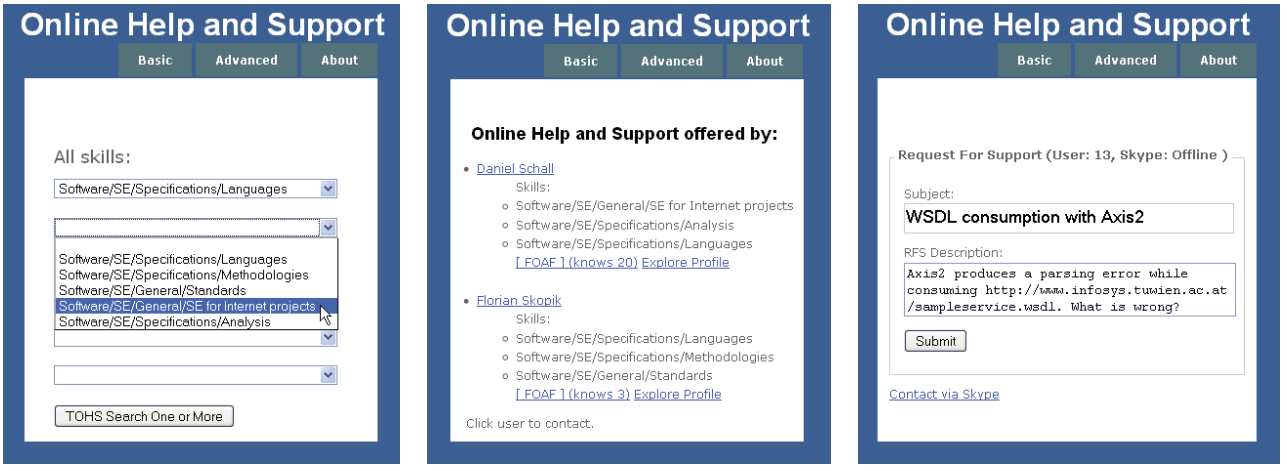
```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wsa="http://.../2004/08/addressing"
 xmlns:hps="http://myhps.org/"
 xmlns:rfs="http://myhps.org/rfs"
 xmlns:vietypes="http://viete.infosys.tuwien.ac.at/Type">
 <soap:Header>
  <hps:activity url="http://.../Activity#42"/>
  <hps:delegation hops="3" deadline="2010-10-15"/>
  <vietypes:timestamp value="2010-10-14"/>
  <wsa:MessageID>uuid</wsa:MessageID>
  <wsa:ReplyTo>http://.../Actor#Florian</wsa:ReplyTo>
  <wsa:From>http://.../Actor#Florian</wsa:From>
  <wsa:To>http://.../Actor#Daniel</wsa:To>
  <wsa:Action>http://.../Type/RFS</wsa:Action>
 </soap:Header>
 <soap:Body>
  <hps:Request>
   <rfs:subject>WSDL consumption with Axis2?</rfs:subject>
   <rfs:requ>Axis2 reports a parsing error while consuming
             the given resource. What is wrong?</rfs:requ>
   <rfs:category>
     Software/SE/General/SE for Internet projects
   </rfs:category>
   <rfs:resource> <!-- details omitted --> </rfs:resource>
   <rfs:comments>Used Axis2 1.4</rfs:comments>
   <rfs:keywords>WSDL, Axis2</rfs:keywords>
  </hps:Request>
 </soap:Body>
</soap:Envelope>
```

5.3 Expert Discovery Application

To conclude our implementation discussion, we present user interfaces demonstrating the integration with infrastructure services (i.e., as shown in Fig. 5) including *Skill Requirements Definition*, *Expert Discovery*, *Expert Involvement*, *RFS Creation*, *Profile Visualization*, *RFS Delegation Management*, and the *Social Network Management*. The screenshots at the top in Fig. 8 visualize the input data provided by the expert seeker and the figure at the bottom shows a simple HPS-based RFS form. All user interfaces have been implemented using state-of-the-art web technologies.

The following steps are performed:

1. The expert seeker specifies a set of demanded skills (Fig. 8a) using dropdown lists. For simplicity, we do not visualize selection options for matching preferences (as introduced in Section 4.1).
2. A list of experts is retrieved matching the search criteria (Fig. 8b). The set of expert skills are visualized.



(a) Discover experts based on skills, contextual constraints, and personal preferences. (b) Retrieve a list of experts that match the search criteria, and manually select one. (c) Compile and send RFS to selected expert or contact directly via Skype.

(d) Experts browse through the list of received RFSs and process, delegate or reject them.

Fig. 8. Flexible expert involvement with the *Trusted Online Help and Support Service*.

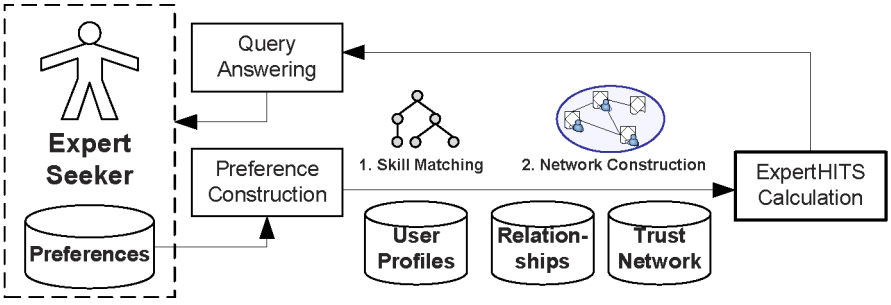


Fig. 9. ExpertHITS calculation steps.

Additionally, the experts' profile can be retrieved (see `[ FOAF ]` link). Such profile information is typically available via public web sites containing information about collaborators, joint projects or scientific papers published by an expert. As mentioned before, FOAF profile information and `knows` relations are using by ExpertHITS. The expert seeker can also visualize the social network as a graph (see `Explore Profile`). We use a JavaScript library (http://thejit.org) for graph visualization.

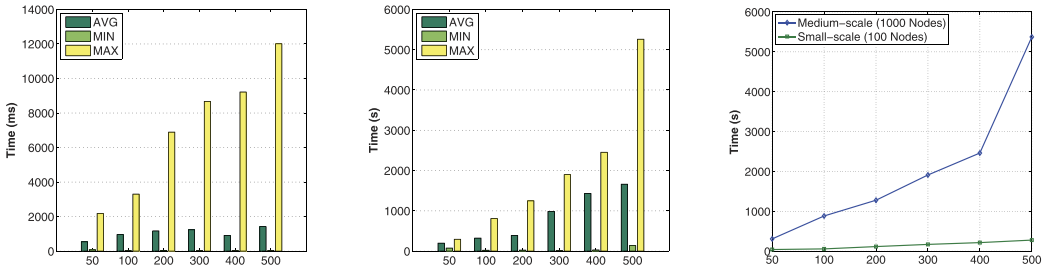
3. In Fig. 8c, the expert seeker enters information regarding the RFS (simplified form for brevity). Upon submission, form elements are translated into XML and SOAP messages. This is done on the client side using XForm technology and a browser plugin for XForm processing. Additional communication tools might be used (if available) such as Skype, however, without the ability to perform complex interactions such as delegations or duplication of requests (4-eyes principle).
4. The expert can review a request (see Fig. 8d) and decide whether to process a request or delegate the request to some other peer. Delegation loops or cycles are prevented by the *RFS Delegation Management* (see Fig. 5). Delegation rules (see also *Rule Management*) ensure that RFSs are not delegated back to the originally delegating HPS.

6 Experiments

In Fig. 9, we show the essential steps of the ExpertHITS ranking algorithm including data sources used to calculate the weighted interaction graph. A query interface enables expert seekers to specify queries based on preferences. As mentioned before, preferences include demanded set of hierarchically defined skills (*Skill Matching*). User profiles are evaluated to find the potential candidate experts. The ExpertHITS calculation is performed *online* based on the weighted, trust-based interaction graph.

6.1 Setup

In our experiments, we focus on the performance of ExpertHITS as well as the influence of trust and ratings on hub/authority scores. In this paper, we do not deal with performance issues due to network delay or end-to-end



(a) Small-scale. (b) Medium-scale. (c) Total processing time.

Fig. 10. Concurrent request processing time ExpertHITS.

characteristics of the entire system. Here, we focus on ExpertHITS calculation time under different conditions.

System configuration. In all our experiments we used a computer with Intel Core2 Duo CPU 2.50 GHz and 4 GB RAM hardware running Java 1.6 and an OSGi Java container for hosting services. A query service has been implemented on top of the HPS Framework [2]. We use trust mining and metric calculation capabilities available as services to construct graphs based on user relations and trust [8]. The ExpertHITS algorithm has been implemented on top of a Java-based graph modeling toolkit.[3]

Data generation. The approach we take is to generate artificial interaction data imitating real collaboration environments. For this purpose, we adopt the *preferential attachment* method [18] which provides a realistic model for science collaboration scenarios. Specifically, a collaboration network is modeled as an undirected graph $G = (N, E)$ comprising a set of nodes N and edges E establishing connections between nodes. The probability of establishing a new connection to a given node is proportional to its degree distribution. Using this basic network structure, we generate interactions (delegations and ratings) associated with edges. Assuming a scale free network with power law distribution, hubs play a central role, thereby generating a large amount of delegations. This behavior is taken into account when generating artificial interactions by estimating that 80 percent of delegations are initiated by about 20 percent of network users (i.e., immitating hubs).

6.2 Results

Complexity is a crucial factor in order to support personalization of queries. The complexity for computing ExpertHITS is $\mathcal{O}(|N| * it)$, $|N|$ representing the number of nodes in the graph and it the number of iterations until the algorithm converges. We analyze different network scales comprising actors and interactions that have already been matched with a specific query context.

- **Small**: 100 nodes, 400 edges ($\approx$60 ms).
- **Medium**: 1,000 nodes, 4,000 edges ($\approx$600 ms).
- **Large**: 10,000 nodes, 40,000 edges ($\approx$12,100 ms).

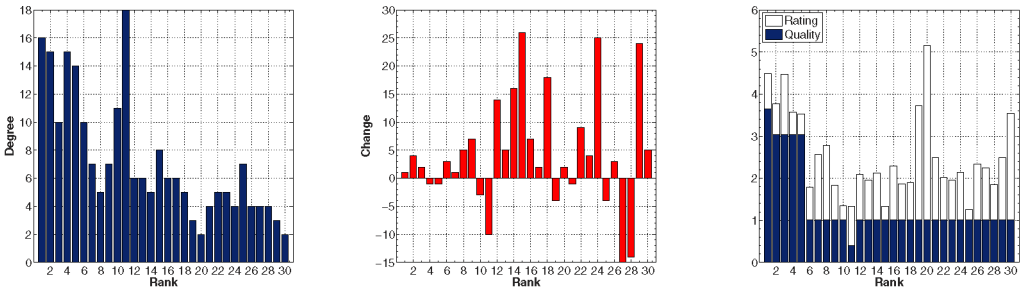
ExpertHITS can be computed in a sufficient amount of time scaling up to large networks (i.e., 10,000 nodes). Notice, these networks represent already filtered subgraphs based on the query context. We believe that this assumption is realistic considering the targeted Expert Web consisting of professionals. The system must be able to handle multiple requests simultaneously. We analyze the performance of ExpertHITS under different load conditions. At this stage, we focus on small-scale (100 nodes) and medium-scale (1,000 nodes) networks. Figs. 10a and 10b show the results given 50-500 concurrent requests to calculate ExpertHITS. A queue holds instances of the constructed network. A thread pool instantiates worker threads to calculate personalized ranking scores based on query preferences.

Small-scale networks can be processed in a real-time manner requiring in our experiments in the worst case (MAX values) up to 12 seconds. On average, 17 seconds can be expected under different load conditions (50-500 concurrent requests). The results of medium-scale networks are shown in Fig. 10b and compared with small-scale networks in Fig. 10c. Computing ExpertHITS in such networks takes up to several minutes when serving concurrent requests (i.e., on average, 390 s at a load of 200 requests). Load conditions in the range between 300-500 concurrent executions of the algorithms results on average in response times between 15-25 minutes.

Given our initial online help and support example, we believe it is sufficient to compute ExpertHITS in this magnitude because illustrated processes in software engineering do not demand for hard computational (time) constraints. Scalability and reduced processing time can be achieved by using multiple servers and load balancing mechanisms. These mechanisms are subject to our future work and performance evaluation.

To test the effectiveness of ExpertHITS, we performed experiments to study the impact of ratings and trust on expert rankings. In Fig. 11, 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 (horizontal axis). Fig. 11a shows the degree of network nodes and Fig. 11b ranking changes obtained by comparing ranking results using the HITS algorithm without taking trust or ratings into account. Specifically, $pos(u)_{HITS} - pos(u)_{ExpertHITS}$ returns the absolute ranking change of u in a given result set. In Fig. 11c, we show the average rating of each ranked node; average rating of node u received from its neighboring nodes divided by the expected rating. We define *quality* as the aggregated trust weights. Quality is calculated as $\sum_{v \in \text{knows}(u)} \sum_{z \in \text{inlink}(v)} w_{zv}$. Quality measures the overall trust in node v. In Fig. 11c, we see that all nodes within the top segment received high ratings given a high degree of links which is the desired property of ExpertHITS. Some nodes are demoted (negative

3. JUNG: http://jung.sourceforge.net.



(a) Node degree. (b) Ranking change. (c) Ratings and quality.

Fig. 11. Impact of ExpertHITS on rankings.

ranking change) since the node (e.g., see 11) has received low ratings even though the node has a high degree of links. On the other hand, 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). The different levels of quality can be explained by the impact of node degree on quality (see rank between 6 to 30). High link degree to trusted authorities increases the quality of hubs. Overall, ExpertHITS exhibits the demanded properties of promoting well-connected and rated hubs, thereby guaranteeing the discovery of reliable entry points to the Expert Web.

7 Related Work

The notion of service orientation is not only applicable to web services. Service orientation in human collaboration is becoming increasingly important. Major software vendors have been working on standards addressing the lack of human interaction support in service-oriented systems. WS-HT [17] and Bpel4People [4] were released to address the emergent need for human interactions in business processes [19]. These standards specify languages to model human interactions, the lifecycle of human tasks, and generic role models. Role-based access models [17] are used to model responsibilities and potential task assignees in processes. While Bpel4People-based applications focus on top-down modeling of business processes, *mixed service-oriented systems* [3] target flexible interactions and compositions of Human-Provided and software-based services. Open service-oriented systems are specifically relevant for future *crowdsourcing* applications [5]. While existing platforms (e.g., Amazon's Mechanical Turk [20]) only support simple interaction models (tasks are assigned to individuals), social network principles support more advanced techniques such as formation and adaptive coordination.

Task-based platforms on the web allow users to share their expertise [21]; or users offer their expertise by helping other users in forums or answer communities [22], [23]. By analyzing email conversations [24], the authors studied graph-based algorithms such as HITS [12] and PageRank [25] to estimate the expertise of users. In [26], an email analysis in enterprises, defining information flow metrics in the social interaction graph was presented. The work by [27] followed a graph-based approach and applied HITS as well as PageRank in online communities (i.e., a Java question and answer forum). While the above cited works attempted to model the importance of users based on interactions; they do not consider that interactions typically take place in different *contexts*. Approaches for calculating personalized PageRank scores [15], [28] were introduced to enable topic-sensitive search on the web. In contrast, we presented a model where expertise analysis is performed considering context information. We proposed an algorithm that can be computed online, while most other approaches demand for offline calculation due to computational complexity.

Recently, trust in social environments and service-oriented systems has become a very important research area. SOA-based infrastructures are typically distributed comprising a large number of available services and huge amounts of interaction logs. Trust in SOA has to be managed in an automatic manner. A trust management framework for service-oriented environments has been presented in [29], [30], and [31], however, without considering particular application scenarios with human actors in SOA. While various theoretically sound models have been developed in the last years, fundamental research questions, such as the technical grounding in SOA and the complexity of trust-aware context-sensitive data management are still widely unaddressed. Although several models define trust on interactions and behavior, existing work lacks case studies about the application of these models in SOA.

8 Conclusion and Future Work

Unlike traditional models found in process-centric environments, we proposed the combination of preplanned process steps *and* ad hoc activities to solve emergent problems in distributed collaboration environments. Our approach is based on the Human-Provided Services concept enabling knowledge workers to offer their skills and expertise in service-oriented systems. Expert discovery is greatly influenced by (behavioral) trust and reputation mechanisms. We demonstrated a novel approach for estimating expert reputation based on link structure and trust relations. Trust information is periodically updated to capture dynamically changing interaction preferences and trust relations. We have shown that ExpertHITS can be computed in an online manner, thereby enabling full personalization at runtime. Existing approaches in personalized expertise mining algorithm typically perform offline interaction analysis. Our empirical evaluations have shown that ExpertHITS exhibits the desired properties; trust and rating weights influence hub- and authority scores. These properties

ensure that our algorithm discovers experts which are well connected to other experts.

Although we have focused on the application of ExpertHITS in human-centric and social collaborations, we believe that the underlying trust-based interaction model can be applied to coordination problems in distributed systems in general. In our future work, we will study *network effects* of two-sided markets [32] in mixed service-oriented systems. Also, we plan to make the system available for public use.

Acknowledgments

Part of this work was supported by the European Union through the FP7 project COIN (FP7-216256).

References

[1] OASIS, "Business Process Execution Language for Web Services, Version 2.0," 2007.
[2] D. Schall, H.-L. Truong, and S. Dustdar, "Unifying Human and Software Services in Web-Scale Collaborations," *IEEE Internet Computing,* vol. 12, no. 3, pp. 62-68, May/June 2008.
[3] D. Schall, "Human Interactions in Mixed Systems—Architecture, Protocols, and Algorithms," PhD dissertation, Vienna Univ. of Technology, 2009.
[4] A. Agrawal et al., "WS-BPEL Extension for People (BPEL4People), Version 1.0," 2007.
[5] D. Brabham, "Crowdsourcing as a Model for Problem Solving: An Introduction and Cases," *Convergence,* vol. 14, no. 1, pp. 75-90, 2008.
[6] D. Artz and Y. Gil, "A Survey of Trust in Computer Science and the Semantic Web," *J. Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 5, no. 2, pp. 58-71, 2007.
[7] J. Golbeck, *Computing with Social Trust,* Human-Computer Interaction Series, first ed. Springer, Dec. 2008.
[8] F. Skopik, D. Schall, and S. Dustdar, "Modeling and Mining of Dynamic Trust in Complex Service-Oriented Systems," *Information Systems,* vol. 35, pp. 735-757, 2010.
[9] C.-N. Ziegler and J. Golbeck, "Investigating Interactions of Trust and Interest Similarity," *Decision Support Systems,* vol. 43, no. 2, pp. 460-475, 2007.
[10] J. Golbeck, "Trust and Nuanced Profile Similarity in Online Social Networks," *Trans. Web (TWEB),* vol. 3, no. 4, pp. 1-33, 2009.
[11] C. Dwyer, S.R. Hiltz, and K. Passerini, "Trust and Privacy Concern within Social Networking Sites: A Comparison of Facebook and Myspace," *Proc. 13th Am. Conf. Information Systems (AMCIS),* 2007.
[12] J.M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," *J. ACM,* vol. 46, no. 5, pp. 604-632, 1999.
[13] ACM Classification, http://acm.org/about/class/1998, 1998.
[14] I. Becerra-Fernandez, "Searching for Experts on the Web: A Review of Contemporary Expertise Locator Systems," *Trans. ACM Internet Technology,* vol. 6, no. 4, pp. 333-355, 2006.
[15] T.H. Haveliwala, "Topic-Sensitive Pagerank," *Proc. 11th Int'l Conf. World Wide Web (WWW '02),* pp. 517-526, 2002.
[16] D.J. Watts, *Six Degrees: The Science of a Connected Age.* WW Norton & Company, 2003.
[17] A. Agrawal et al., "Web Services Human Task (WS-HumanTask), Version 1.0," 2007.
[18] R. Albert and A.-L. Barabási, "Statistical Mechanics of Complex Networks," *Rev. Modern Physics,* vol. 74, pp. 47-97, June 2002.
[19] F. Leymann, "Workflow-Based Coordination and Cooperation in a Service World," *Proc. On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE,* pp. 2-16, 2006.
[20] Amazon.com, "Amazon Mechanical Turk," http://www.mturk.com, 2010.
[21] J. Yang, L. Adamic, and M. Ackerman, "Competing to Share Expertise: The Taskcn Knowledge Sharing Community," *Proc. Int'l Conf. Weblogs and Social Media,* 2008.
[22] P. Jurczyk and E. Agichtein, "Discovering Authorities in Question Answer Communities by Using Link Analysis," *Proc. 16th ACM Conf. Information and Knowledge Management (CIKM '07),* pp. 919-922, 2007.
[23] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne, "Finding High-Quality Content in Social Media," *Proc. Int'l Conf. Web Search and Web Data Mining (WSDM '08),* pp. 183-194, 2008.
[24] B. Dom, I. Eiron, A. Cozzi, and Y. Zhang, "Graph-Based Ranking Algorithms for E-Mail Expertise Analysis," *Proc. Eighth ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD '03),* pp. 42-48, 2003.
[25] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," technical report, Stanford Digital Library Technologies Project, 1998.
[26] T. Karagiannis and M. Vojnovic, "Email Information Flow in Large-Scale Enterprises," technical report, Microsoft Research, 2008.
[27] J. Zhang, M.S. Ackerman, and L. Adamic, "Expertise Networks in Online Communities: Structure and Algorithms," *Proc. 16th Int'l Conf. World Wide Web (WWW '07),* pp. 221-230, 2007.
[28] G. Jeh and J. Widom, "Scaling Personalized Web Search," *Proc. 12th Int'l Conf. World Wide Web (WWW '03),* pp. 271-279, 2003.
[29] D. Kovac and D. Trcek, "Qualitative Trust Modeling in SOA," *J. Systems Architecture,* vol. 55, no. 4, pp. 255-263, 2009.
[30] W. Conner, A. Iyengar, T. Mikalsen, I. Rouvellou, and K. Nahrstedt, "A Trust Management Framework for Service-Oriented Environments," *Proc. 18th Int'l Conf. World Wide Web (WWW '09),* 2009.
[31] Z. Malik and A. Bouguettaya, "Reputation Bootstrapping for Trust Establishment among Web Services," *IEEE Internet Computing,* vol. 13, no. 1, pp. 40-47, Jan./Feb. 2009.
[32] R. Kumar, Y. Lifshits, and A. Tomkins, "Evolution of Two-Sided Markets," *Proc. Third ACM Int'l Conf. Web Search and Data Mining (WSDM '10),* pp. 311-320, 2010.

Daniel Schall received the PhD degree in February 2009 with a thesis on "Human Interactions in Mixed Systems—Architecture, Protocols, and Algorithms" while working as a research assistant at the Distributed Systems Group, Vienna University of Technology. Currently, he is working as a postdoctoral researcher at the Vienna University of Technology. Prior to joining the Distributed Systems Group, he worked at Siemens Corporate Research in Princeton, New Jersey.

Florian Skopik received the PhD degree in June 2010 with a thesis on "Dynamic Trust in Mixed Service-Oriented Systems" while working as a research assistant at the Distributed Systems Group, Vienna University of Technology. Currently, he is working as a postdoctoral researcher at the Vienna University of Technology. His research interests include models, algorithms, and architectures that support the management of trust in mixed service-oriented environments.

Schahram Dustdar is working as a full professor of computer science (informatics) with a focus on Internet technologies heading the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology (TU Wien), where he is the director of the Vita Lab. He is the chair of the IFIP Working Group 6.4 on Internet applications engineering and a founding member of the Scientific Academy of Service Technology.