DISSERTATION

Human Interactions in Mixed Systems - Architecture, Protocols, and Algorithms

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften

unter der Leitung von

Univ.-Prof. Dr. Schahram Dustdar
Institut für Informationssysteme
Arbeitsbereich für Verteilte Systeme
Technische Universität Wien

eingereicht an der

Technischen Universität Wien
Fakultät für Informatik

von

Daniel Schall

d.schall@infosys.tuwien.ac.at

Wien, Jänner 2009

Kurzfassung

Die zugrundeliegende Arbeit umfasst die Anwendung des Web Services Paradigmas in Kollaborationssystemen. Verteilte Systeme können mittels Web Services verbunden und erweitert werden. Die Anwendung einer Service-orientierten Architektur (SOA) ermöglicht es, Services zu finden, ohne die Verfügbarkeit aller möglichen Instanzen eines Services zu jedem Zeitpunkt zu kennen.

Jede Kollaboration erfordert in der Regel Interaktionen zwischen Menschen und Software Services. Diese Interaktionen werden von dynamischen und komplexen Wechselwirkungen geprägt. In heutigen Service-orientierten Architekturen ist das Zusammenspiel von Interaktionen zwischen Menschen und Services nur bedingt unterstützt und grösstenteils limitiert auf vordefinierte Geschäftsprozesse.

Die vorgestellte Arbeit beschäftigt sich mit Modellen zur Erweiterung von SOA mittels "Human-Provided Services" (HPS), welche als Dienstleistungen von Menschen in Service-orientierten Kollaborationssystemen verfügbar gemacht werden können. Die Anwendbarkeit der HPS Softwarearchitektur ist nicht nur auf Geschäftsprozesse limitiert, sondern erlaubt es, flexible Szenarien ohne vordefiniertem Interaktionsmodell zu realisieren.

Abstract

Web-based collaboration platforms evolve into service-oriented architectures by promoting composite and user-enriched services. In such platforms, the collaborations typically include both human and software services, thus creating highly dynamic and complex interactions. However, in existing platforms users cannot specify different interaction interfaces as services that can be reused in various collaborations. We argue that people need more ways to indicate their availability and desire to participate in collaborations.

Furthermore, open service-oriented environments require a flexible yet reusable collaboration model because compositions comprise interactions between people and a number of software services. The presented work introduces Human-Provided Services (HPS), which can be included in ad-hoc and process-centric collaborations. The HPS framework fosters the user-driven integration of human capabilities into service-oriented infrastructures, thereby promoting reusability and flexibility of interaction flows. By using the framework, people can manage their interactions and provide services in dynamic collaborations.

Acknowledgements

First and foremost, I would like to thank my adviser Prof. Schahram Dustdar for the excellent supervision and mentoring during all phases of my research. I have been very lucky to have an adviser who gave me the liberty to explore different ideas and directions, and at the same time the guidance I needed to complete my dissertation.

I would also like to thank Prof. Frank Leymann for valuable comments and suggestions that helped me to improve this work. He helped me to correct notation in formal specifications and algorithms.

I would like to acknowledge Christoph Dorn, Robert Gombotz, and Hong-Linh Truong for fruitful discussions on related topics.

Most importantly, this would have not been possible without the continuous support of my family. This dissertation is dedicated to my family.

Finally, I am grateful for the constant financial support from the EU FP6 project inContext (IST-034718), funding parts of the research discussed in this dissertation.

Daniel Schall
January 18, 2009
Vienna, Austria

For my family

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Meaning
α	The predefined PageRank damping factor. A value between 0.8 and 0.9.
$\vec{PR}$	The PageRank vector for parameter α.
$\vec{p}$	The teleportation distribution vector called *personalization vector*.
m	A metric or measured value in human interaction networks, $m \in M_R$.
w_m	Denotes the weight of a metric, $w_m \in W_{M_R}$.
$w_{v,u}$	Weight of a link connecting v to u.
ws_v	The sum of outgoing link weights of node v.
IIL	Interaction intensity level.
c	Denotes a context tag.
c_f	The frequency of a context tag.
γ	A factor in the range $0 < \gamma < 1$ used in additive smoothing.
$SE(u;c)$	Context-dependent skill- and expertise-based importance ranking.
$\vec{p_m}$	The teleportation distribution vector for metric m.
$DSA(u;C')$	Context-aware DSARank based on the set C' of context tags.
w_c	Weight for context-dependent DSARank.
$i(g)$	Subgraph intensity.
$w_{(u,tag)}$	Weight of a tag applied to resource.
$\omega(tag)$	The frequency of a tag applied to resource.
ϕ	Variable depicting the correlation coefficient within the range $[0, 1]$.
$pref(u)$	User-define preference vector used in recommendations for HPS design.
T	Denotes the set of human tasks, $ht \in T$.
τ	Time interval (expiration time) after which ht fails.
T^Q	The set of all possible task states, $q \in T^Q$.
$H(c)$	The set of tasks in context c, $H(c) = \{ht_i \| ht_i \in T, c \in C\}$.
s	Depicts an HPS, $s \in S$.
$HT(s)$	The set of tasks processed by s.
$S(c)$	The set of services associated with a context c.

Glossary

Term	Definition
Human interaction	Any interpretable action or event that is used to establish a link between actors.
Mixed system	A system comprised of human actors and software services interacting in regular and irregular ways.
Activity	A procedure performed by human actors in collaborations.
Activity declaration	A description of the activity.
Activity instance	The activity carried out by human actors.
Action	Interaction between actors conveying activity-related information.
Control Action	Coordinative or informative actions to capture activity-change events.
Human-Provided Services	Human capabilities represented as Web services-based interaction interfaces.
Ad-hoc flows	Collaboration and interaction flows not following predefined model.
State-aware interactions	Shared awareness between actors of the progress of activities.
Process-centric	Modeled control and data flow in workflow-based systems.
Opportunistic compositions	Loosely structured collaborations comprising interactions between humans and between humans and software services.

Chapter 1

Introduction

Web services have paved the way for a new blend of collaboration systems. Services let us design modular information systems in distributed environments (Alonso et al. 2003, Papazoglou et al. 2007). Service interfaces are described in a standardized manner by using, for example, the Web Services Description Language (WSDL). Users can create collaborative features by (re)using and composing Web services. 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), (Andrews et al. 2003). Services are increasingly found in open collaboration platforms that are available on the Web. Users and developers have the ability to use Web services in various applications because services offer well-defined, programmable, interfaces. In process-centric collaboration, we typically follow a top-down approach and define workflows by using *activities* to specify work items in the workflow, *control flows* to define the structure of activities and their relations, and *data flows* to assemble and disassemble input or output data (Leymann 2006). Before creating a process model, the business analyst or process designer must fully understand each step in the process. This phase is called the design or modeling phase. A process model is executed (enacted) by instantiating a set of services that realize the modeled flow. Such models' *reusability* is generally high, because we can apply process models several times. However, *flexibility* is limited because whenever unexpected changes occur, which have not been considered in the modeling phase, the flow has to be remodeled. Such changes may cause exceptions, disrupting the normal execution of the process. Thus, it is important to support *adaptivity* in collaborations. An important role towards adaptive processes plays the ability to support "adhocness" — enabling humans to create "unforeseen" activities at run-time. Ad-hoc flows are more flexible but less reusable, because many aspects depend on the actual players (humans) involved in such flows. While the process-centric collaboration approach follows a top-down methodology in modeling flows, ad-hoc flows in flexible collaborations *emerge* at run-time. A run-time environment *constraints* the execution of flows. Such constraints are, for example, the availability of resources, services, and people.

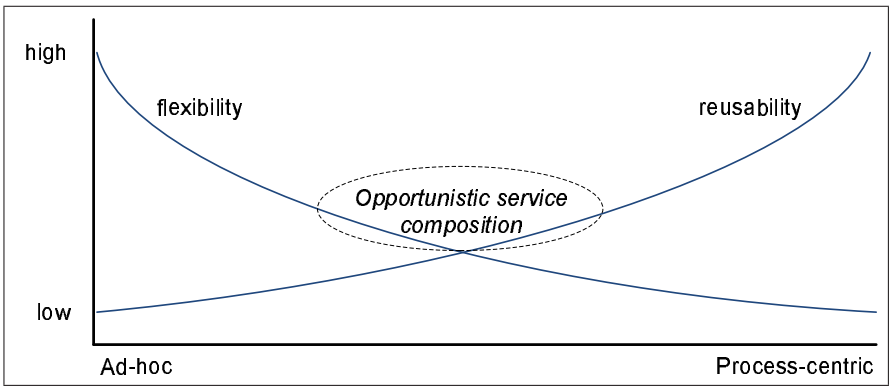


Figure 1.1: Flexibility versus reusability in collaboration. Opportunistic service composition represents loosely structured collaborations.

In this work, we define a collaboration spectrum ranging from ad-hoc to process-centric interaction models (Dustdar 2004,Schall et al. 2008b). Ad-hoc collaborations, for example, are situations in which people must act in a flexible manner.

We show this range in Fig. 1.1: **ad-hoc**, activities are created at run-time (a bottom-up system perspective); **process-centric**, activities and flows are created at design-time (top-down perspective). The area — depicted as *opportunistic service composition* — represents loosely structured collaborations. The degree of control in compositions, for example, compositions of software services, can be understood as the trade-off between fexibility and reusability: we lose high reusability of well-modeled processes; but gain flexibility and robustness at run-time. The philosophy of this thesis is to view compositions of humans and software services as *complex systems*. In complex systems, "the whole is more than the sum of its parts", Aristotle. Open collaboration environments require flexible compositions comprised of interactions between human and software services. Thus, a human is typically *part-of* such compositions. The presented work introduces Human-Provided Services (HPS), which can be used in ad-hoc and process-centric collaborations. The HPS framework enables humans to integrate their capabilities as Web services in SOA.

1.1 Motivating Examples

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 companies specific platform; it is now possible to utilize the knowledge of many people participating in interactions on the Web. The shift towards the Web 2.0 allows people to write blogs about their activities, share knowledge in forums, write Wiki pages, and utilize social-services to stay in touch with other people.

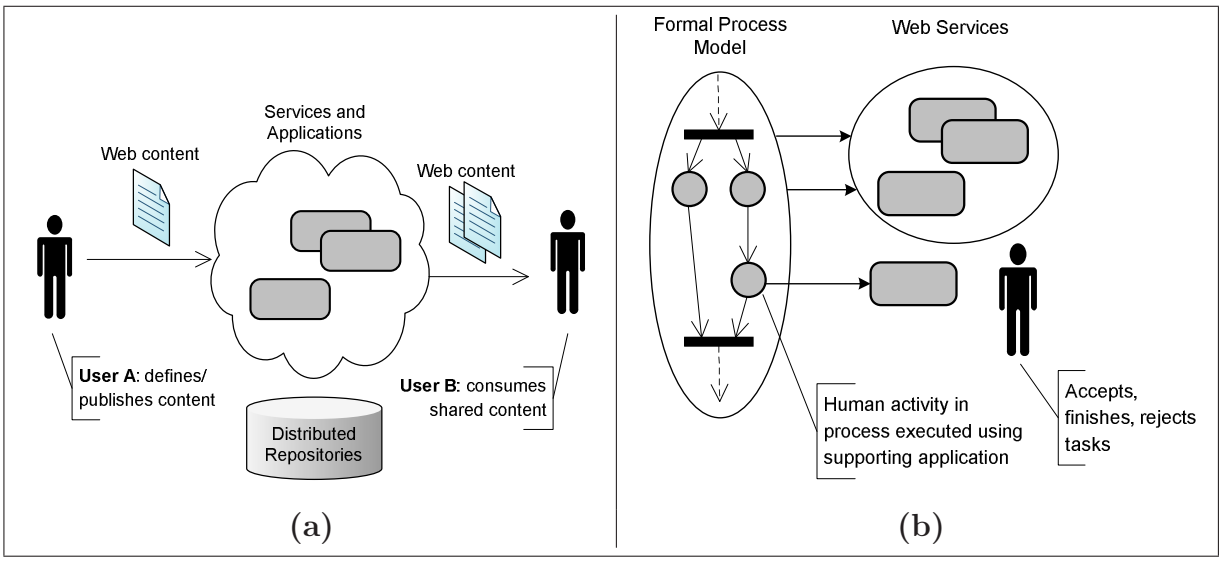


Figure 1.2: (a) Ad-hoc and (b) process-centric collaboration models.

The Web has moved from being a "read-only" repository of Web pages to a *Web of services*, which can be composed and enriched. Moreover, properties of users are no longer tied to specific platforms and Web pages; they will be globally available (Ramakrishnan and Tomkins 2007).

Consider Fig. 1.2 (a), User A publishes Web content by using open service-based applications. Other Web users can then consume, aggregate, or filter the content. However, users can't apply the same procedure in other collaborations (for example, to share content including documents, videos, and photos) because the depicted scenario relies mainly on ad-hoc interactions. In addition, the collaboration is not structured as there's no interaction link between the users. This makes it difficult – if not impossible – to manage interactions spanning multiple users and services.

Figure 1.2 (b) shows a process-centric collaboration scenario involving human actors (depicted as human activity in the process model). An example of such interactions are human activities in BPEL processes. Such activities are defined as BPEL4People (B4P) activity (Agrawal et al. 2007). However, the applicability of such models in open collaboration environments may be limited because we can't model *emerging interactions* between humans and software services in advance. Consider the following motivating scenarios detailing collaborations involving both human and software services.

Ad-hoc collaborations: in Fig. 1.2 (a), User A records a video and posts it on the Web. However, current platforms don't let consumers (User B) actively find available users who can contribute to collaborations by producing the desired content on demand. In particular, users should be able to find any person who can deliver the desired content using whichever platform (service) to host the Web content. This use case depends on the activity to be performed by humans and the involved services; but not on the platform that is used to share or host the content.

User-defined processes: continuing the previous use case, the collaboration might involve many different services and people. For example, a (software) service should automatically check the input User A receives (for example, for file format compatibility) and convert it into a suitable format — if needed. The requester can then check whether the provided contribution needs to be revised or re-recorded. Here interactions include human as well as software services. However, most systems do not address reusability aspects of loosely structured processes in such collaboration scenarios. User-defined processes allow people to manage and control their interactions in a flexible manner.

Human interactions in formalized processes: it becomes increasingly important to enable interactions between business processes and human actors (Fig. 1.2 (b)). However, people are increasingly on the move and use different devices to access various information. So, we must consider mobility aspects, such as the location and limited processing power of a user's mobile devices, and we must adjust interactions according to the user's context. In particular, software services should be able to find and select the right person. However, most systems do not support the lookup of humans based on dynamically changing properties such as expertise and skills of users.

1.2 Problem Formulation

Recently, Web-based collaboration platforms have started to evolve into Web services-based architectures. In such platforms, collaborations include both humans and software services. The challenge of composing these new type of services — user-driven contributions as services and software services — is that interactions are highly dynamic and context-dependent. However, a fundamental issue is that existing collaboration platforms do not support humans in specifying their capabilities as services. User-defined services can be used as interfaces to interact with people. Furthermore, humans need different ways to denote their availability and desire to participate in collaborations. The problem is that current systems lack the notion of *human capabilities* in SOA. The challenge is to support the user in designing and providing services. Such services are called HPS and can be discovered like any software-based services. However, humans must be able to offer HPSs and mange their interactions in pervasive environments. Moreover, in open and dynamic collaboration environment, typically a very large number of people collaborate and interact by using different collaboration tools and platforms. It is important to determine *expertise* and *skill* level of users. Somebody seeking help or advice on how to solve a specific problem needs to be able to find the right expert.

However, the expertise and importance of users changes depending on performed tasks, interactions with other users, as users gain know-how by collaborating with other experts, and based on the information users receive from other people. An expert recommender algorithm must consider the expert's *interest* in a certain area. For example, a scientist may have done research in a certain field; however, the scientist might change his/her principle research domain over time and therefore no longer be the right expert to contact.

Thus, the interest and activity level of a person in a specific field must be considered. We believe that ranking models should not only rely on profiles or skill information that need to be maintained and manually updated by users. It is unlikely that a single skill- or expertise-ontology is sufficient to capture the concepts and requirements of various collaboration domains. Tagging mechanisms can be used to classify information and to derive the context of interactions. Golder and Huberman (2006), for example, investigated tagging models in collaborations. Tags provide a) input to derive skills and user-interests; and b) the context of activities and interactions. The challenge is to devise a ranking model that is able to capture these dynamic, context-dependent properties.

1.3 Contributions

We introduce *Human-Provided Services* because current systems don't sufficiently address the challenges and problems presented in the motivating use cases. In this work, we present the applicability of HPS in open collaboration environments. We introduce a framework to support the integration of human capabilities and interactions in SOA. The HPS framework lets people supply services based on their skills and expertise. HPSs are interaction interfaces towards humans, letting users define various HPSs for different activities. Thus, HPSs are used to define the ability of users to participate in ad-hoc as well as process-centric collaborations.

Users can manage interactions, which might potentially span various platforms and services. People benefit from HPS because they can reuse different services in various collaborations, thus fostering the reusability of human capabilities. Moreover, HPSs can increase flexibility in collaborations because problems can be solved by HPSs that software services alone can't solve (see also (Naor 1996)).

At the implementation level, the framework utilizes Web services technologies such as WSDL to describe HPSs. From the user's point of view, HPSs are *activities* and *actions* that users can design in form of services. Based on HPS, we discuss the application of our ranking approach and the role of tagging mechanism to compute personalized ranking scores. We propose link-analysis methods to determine the importance of users in human interaction networks. Specifically, we adopt and significantly extend the *PageRank model*, as introduce by Page et al. (1998), for human interaction analysis. The benefit of PageRank is that the global properties of an interaction graph are taken into account when establishing the importance of users. This method is a variant of eigenvector centrality. We devise a new model, which we call *DSARank*, to capture interaction dynamics; thereby ensuring that ranking scores are distributed based on the activity-level of individuals. In addition, we propose link-analysis in subgraph partitions of the human interaction network to obtain context-sensitive rankings. We provide empirical results by evaluating our ranking model in real-life interaction networks. On the one hand, human interaction graphs based on email interactions; and on the other hand phone conversations in cellular networks.

Publications

- Activity and Interaction Modeling: Publications in the area of activity-centric collaboration and related interaction models appeared in *Human Interactions in Dynamic Environments through Mobile Web Services* (Schall et al. 2007) and *VieCAR - Enabling Self-adaptive Collaboration Services* (Schall et al. 2008). The concept of activities and actions using SOA has led to the definition of Human-Provided Services. Initial use cases of HPS were published in a book chapter *Context-aware Mobile Computing* (Schall, Dorn, and Dustdar 2007a). Furthermore, we gave a brief introduction of HPS and the relation to activity-centric collaboration in (Schall, Gombotz, and Dustdar 2006). We discuss activities and interaction models in Chap. 3.

- Human-Provided Services: The most important manuscripts discussing the HPS framework as well as HPS interaction models were published at

  1. International Conference on Web Services (*ICWS 2007*) (Schall, Gombotz, Dorn, and Dustdar 2007): focusing on the notion of activities and the mapping of such activities to interactions with HPS. In the paper we focused on mobile collaboration scenarios and context-aware discovery of HPSs. The work was performed based on results of *Web Services on Embedded Devices* (Schall, Aiello, and Dustdar 2006) and *Wireless Internet Applications* (Schall, Dorn, and Dustdar 2007b), which we present in Sec. 7.4.
  2. IEEE Conference on Enterprise Computing, E-Commerce and E-Services (*EEE 2008*) (Schall, Truong, and Dustdar 2008a): we presented the HPS framework and a detailed description of HPS related XML collections. Chapter 5 discusses the framework and introduces HPS related metrics.
  3. IEEE Internet Computing (special issue on Web-scale workflows) (Schall, Truong, and Dustdar 2008b): presenting the HPS framework as well as selected use cases showing how HPSs can be utilized in Web-scale collaborations (see Sec. 7.3).

- Importance Ranking and Recommendations: We presented a ranking approach for activities, context and services at the EUROMICRO Conference on Software Engineering and Advanced Applications (*SEAA 2008*) (Schall, Dorn, Dustdar, and Dadduzio 2008). Furthermore, the publication contained a description of VieCAR (*Vienna Collaborative Activity and Resource Management Framework*), which is an OSGi-based middleware closely related to the HPS middleware implementation, see (Schall et al. 2008, Section 2). However, the details regarding the ranking model employed in VieCAR are not presented in this thesis. The HPS ranking model and related metrics (as presented in this thesis) have been submitted to an international journal; submission entitled *Global Importance Ranking based on Local Interaction Intensities and Context*. However, no outcome of the review process was available with the completion of this thesis. A brief introduction of ranking models and interaction mining was given in inContext reports (Tilly, Yu, Schall, and Peray 2007) and (Tilly, Yu, Schall, and Peray 2008).

- Tagging and Message Annotation: An initial tagging model that associates task-information as annotations with, for example, email messages, was presented in *Pattern-based Collaboration in Ad-Hoc Teams Through Message Annotation* (Schall, Gombotz, and Dustdar 2007). The work provided the basic ideas for the interaction-based recommendation algorithms to support the design of HPSs (Sec. 5.4.1).

Other publications focusing on context, e.g., (Dorn, Schall, and Dustdar 2006), were not directly used in this thesis but are related to the problem of context-aware HPS discovery.

1.4 Structure of Thesis

In Chap. 2 we discuss related work in task-based collaboration platforms, SOA and Web services, and ranking approaches. Chapter 3 introduces basic interaction models in human collaboration and presents our activity-based interaction model. Activity-centric collaboration is the starting point for HPS. We discuss HPS interaction models emphasizing the role of human interactions in SOA.

In Chap. 4, we present our interaction analysis approach, called *DSARank*, focusing on the estimation of user-importance, skills and expertise level in human collaborations. However, our ranking model is well suited for importance ranking of Web services in general.

In the following, we provide on overview of the HPS framework in Chap. 5. An important aspect in enabling HPSs is to support users in designing HPSs, for example, the design of various SOA artifacts. We present a user-centric approach helping users to design HPSs. We provide an architecture overview and discuss the main features of the framework. To support collaborations in open service-oriented environments, various metrics need to be defined to enable ranking, recommendations, and rewarding models. These models are based on, for example, human tasks.

Chapter 6 details the various software components and services implemented within the HPS framework. In addition, we highlight the implementation of the interaction analysis and ranking APIs. Chapter 7 presents the experiments performed in this work including the evaluation of interaction-analysis algorithms and a performance study of Web services in pervasive environments.

Finally, a conclusion of this work is given in Chap. 8 by discussing open issues and by providing an outlook on future research.

Chapter 2

Related Work

The work presented in this dissertation focuses on a methodology and tools allowing human interactions in SOA to be executed in a flexible manner. In complex systems, there are several types of architectural views (Levis 1999, Crawley et al. 2004):

- The functional architecture: a partially ordered list of functions that are needed to accomplish the system's requirements.
- The physical architecture: representation of physical resources and their interconnections.
- The technical architecture: an elaboration of the physical architecture that comprises a minimal set of rules governing the arrangement, interconnections, and interdependence of the elements, such that the system will achieve the requirements.
- The dynamic operational architecture: a description of how the elements operate and interact over time while achieving the goals.

We structure our discussion into related work in the area of task-centric computing, SOA, Web services, and process-centric collaboration. In Sec. 2.1, we discuss various approaches ranging from *human computation* in open collaboration environments, for example, platforms available on the Web, to *workflow* systems, which can be used to execute business processes. In our discussion, we emphasize systems based on Web services technology, focusing mainly on *functional and technical architectures*.

Second, we cover research in pervasive computing using mobile Web services (Sec. 2.2). One of the major challenges in supporting human interactions in SOA is to enable flexible interactions in pervasive environments, for example, information access using mobile devices. In this work, we present Web services toolkits and their performance characteristics.

Third, we present research in *complex systems* and the dynamic nature of human interactions (Sec. 2.4). Related research mainly falls into the *dynamic operational architecture*

view. *Human dynamics* have great significance for various metrics and algorithms presented in this thesis. In particular, our metrics attempt to capture the dynamic nature of HPS. The interaction analysis algorithm presented in this work is based on statistical ranking models, which have been applied with great success in search engines (see Sec. 2.3).

2.1 Task- and Service-oriented Systems

Collaboration systems use tasks to associate a state with a work item that has to be performed by a human actor. Tasks can be used in systems that model workflows by designing process models. On the other hand, tasks can be used in ad-hoc collaboration to establish state-awareness among people involved in the interaction (collaboration).

Recently, major software vendors have been working on standards addressing the lack of human interaction support in service-oriented systems. WS-HumanTask (WS-HT) (Amend et al. 2007) and B4P (Agrawal et al. 2007) were released to address the emergent need for human interactions in business processes. These standards specify languages to model human interactions, the lifecycle of humans tasks, and generic role models. In (Russell and Aalst 2007), the relation of B4P-related Web standards and resource patterns was discussed. Role-based access models (see Agrawal et al. (2007) and Mendling et al. (2008)) are used to model responsibilities and potential task assignees in processes. A concrete implementation of B4P as a service was introduced in (Thomas et al. 2007, Holmes et al. 2008). In the following, we provide a brief overview of B4P and WS-HT. A detailed knowledge of BPEL concepts and syntax is not needed.

Human Tasks in B4P: We discuss B4P from the technical (specification) point of view since B4P is becoming a broadly accepted standard to support human interactions in BPEL. Figure 2.1 shows the basic concepts in B4P processes. Many BPEL processes require `Human Interactions`, which is basically the definition of a set of `Human Tasks` that are assigned to people. The `BPEL Role Model` defines task as well as process-centric roles. For example, task specific roles such as `Initiator` or `Stakeholder`. The `Logical People Group` allows for the "late binding" of people involved in interactions. At run-time `Logical People Groups` are bound to, for example, organizational directories (Leymann 2006).

Compared to concepts in B4P, people using the HPS framework decide which service they provide, for example, in a specific collaboration context to manage their interactions by using HPS. B4P, for example, specifies how a process architect can involve people in formalized processes, or workflows. However, "emerging" aspects of the collaboration environment such as user-defined services are not covered in WS-HT or B4P-related specifications.

As mentioned earlier, it is important to support ad-hoc flows in collaborative systems. *Caramba*, a system enabling the coordination of distributed teams using activities was introduced by Dustdar (2004). In Caramba, activities are used to establish state-awareness

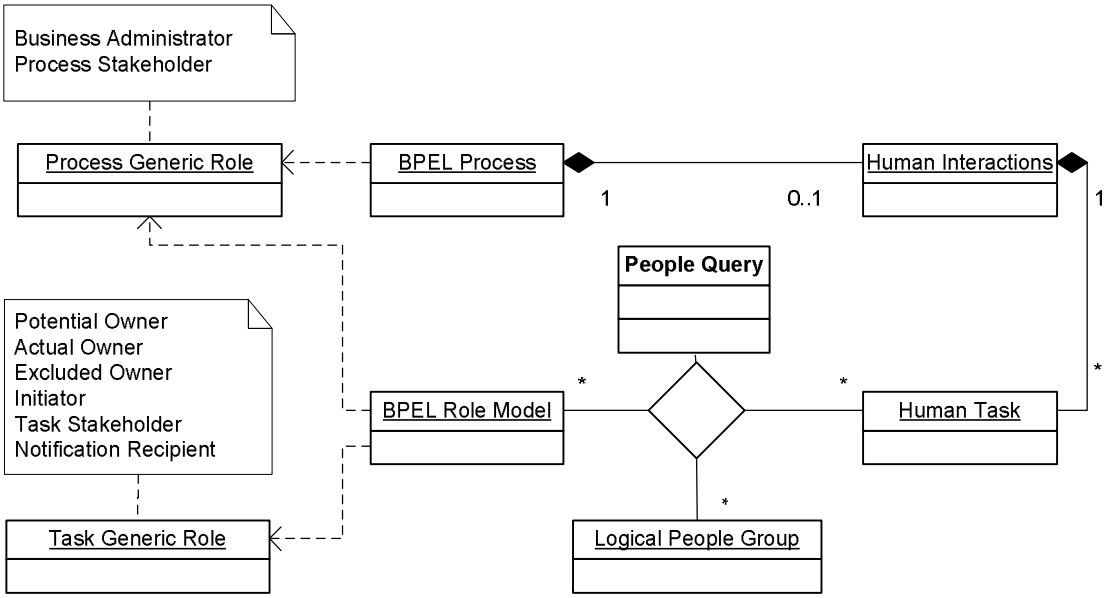


Figure 2.1: B4P task and role model.

in a process. Adams et al. (2006) introduced *worklets* grounded in *activity theory* representing self-contained subprocesses. Garlan et al. (2004) proposed adaptive, task-based platforms to cope with changing environmental conditions. Similarly, Moody et al. (2006) introduced business activity patterns to design flexible applications. Human tasks metrics in workflow management system have been discussed in (Kumar et al. 2002). A formal approach to modeling and measuring inconsistencies and deviations, generalized for human-centered systems, was presented in (Cugola et al. 1996).

Interface Generation: The presented HPS framework utilizes open standards such as WSDL[1], to describe HPS interfaces, and XForms[2] to automatically generate user interfaces. GUI generation and mappings for WSDL was presented in (Kassoff, Kato, and Mohsin 2003, Song and Lee 2007). Also forms generators such as IBM's XML Forms Generator[3] (link followed on January 2009) are available for the Eclipse environment. In this work, we not only focus on automatic GUI-generation based on WSDL descriptions, but also the mapping of human activities onto Web services. We support the automatic generation of different presentations formats (WSDL or XForms).

Human computation is motivated by the need to "outsource" certain steps in a computational process to humans (Gentry et al. 2005). Kosorukoff and Goldberg (2001) presented an application of human computation in genetic algorithms. A variant of human computation called games that matter was introduced by von Ahn (2006). Related to human computation are systems such as Yahoo! Answers[4] and Amazon Mechanical

[1] `http://www.w3.org/TR/wsdl`
[2] `http://www.w3.org/MarkUp/Forms`
[3] `http://www.alphaworks.ibm.com/tech/xfg`
[4] `http://answers.yahoo.com/`

Turk[5]. Both Yahoo! Answers and Amazon Mechanical Turk are Web-based, task-centric, platforms. Users can publish, claim, and process tasks. Su et al. (2007), for example, evaluated the task properties of a similar platform in cases where large amounts of data are reviewed by humans. While Yahoo! Answers is mainly used as a question/answer forum, Amazon Mechanical Turk enables businesses to access the manpower of many people using a Web services API.

In Table 2.1, we compare different collaboration paradigms. The first column shows features, or capabilities, which collaboration systems must in our opinion support to address the needs in large-scale collaborations and workflows involving human and software services. Task-based systems such as Yahoo! Answers are denoted by the category "human reviewed data".

2.2 Web Services in Pervasive Computing

Web services-based applications that can be hosted by mobile devices have recently earned much attention by academia and industry. Service-oriented infrastructures supporting mobile Web services — i.e., the registration, discovery, and invocation of services — must account for the challenges imposed by embedded devices. These devices have limited hardware resources in terms of CPU/battery power and memory. Additionally, mobile services may become unavailable due to interruptions of network connectivity. Mobile Web services open many interesting opportunities for HPSs. For example, services can be provided in a certain location area. Furthermore, requesters may specify constraints in the HPS discovery process such as *proximity* to find, for example, services that are nearby. Thus, the research performed in this work is related to human interactions in dynamic environments using mobile Web services. In particular, we focused on deployment, management mechanisms, and performance characteristics of mobile Web services.

Understanding performance limits is important as many Web services-based applications fulfill critical tasks in terms of retrieving or providing information. Aiello and Dustdar (2008), for example, presented a Web services stack with the aim of connecting heterogeneous devices at home. A mobile Java-based platform utilizing Web services has been introduced in (Rellermeyer and Alons 2007). Work in the area of interoperability problems and SOA, with special focus on legacy issues, can be found in (Jammes, Mensch, and Smit 2005). The work involved the implementation of a Web services stack on devices following a form of device profiling. From the technological point of view, Pham and Gehlen (2005) conducted a performance study of SOAP-based servers on mobile devices. The gSOAP toolkit has been introduced by Engelen (2003a). Furthermore, a performance study and comparison of various Web services toolkits including gSOAP, .NET-based implementations, Apache Axis, and xSOAP was presented in (Engelen 2003b). Tierno and Campo (2005) analyzed the limits of applications on a mobile Java platform. In particular, the

[5] `http://www.mturk.com/`

Feature	Human computation	Human reviewed data	Human process interactions	Expert finder systems
Human requesters	no	yes	no	yes
Modeling interactions in process	yes	yes	yes	no
Interaction-based collaborations	no	No explicit collaboration link	yes	no
Open collaboration environments	yes	yes	Enterprise level collaboration	Ontologies describing skills
Context-dependent discovery	no	Skills	Role models	Skills
Expertise ranking	no	yes	no	yes
User-defined interactions	no	no	Defined by process designer	no
User-defined services	no	no	no	no

Table 2.1: Related collaboration systems and the features they support.

authors discussed the ability to process multimedia data, for example, applications such as image processing, movement detection, and pattern recognition, on smart phones. In addition, various code optimization techniques were highlighted.

2.3 Metrics and Ranking

The definition and modeling of metrics associated with Web services (e.g., QoS metrics) are active research areas in the Web services community. The challenge is to find a common understanding and agreement on how to express service related metrics. The reason is that Web services are a "horizontal" technology to implement domain specific applications. Thus, it would be desirable to define a common set of metrics and model that can be extended for a particular domain. To this date, a variety of QoS models have been proposed to cover the most common QoS attributes of services such as response time and service availability. The OASIS working group developed a model for the management of Web service quality factors (Kim and Lee 2005). A method for analyzing Web service interactions and a categorization of performance and dependability attributes was presented in (Rosenberg et al. 2006). Maximilien and Singh (2004) presented a QoS ontology for autonomic Web services and a matching algorithm to find the most trusted service. Sheng et al. (2004) proposed personalized service composition considering various constraints such as context; for example, location.

2.3.1 Ranking Web Services

Ranking and selection of the "right" service is highly important in SOA. A large number of service providers may publish their services on the Web, which can be (dynamically) found and invoked by service consumers. In a manner similar to search engines for Web pages, ranking algorithm for services should suggest the most relevant service to perform certain "activities". Rankings can be computed by aggregating functional and non-functional properties (NFPs).

Aggregated scores based on, for example, QoS metrics can be calculated using *scoring of preferences* (Dujmovic 2007), for example, see (Andreozzi et al. 2006). Ranking of services using mining techniques has recently gained attention because the quality of a particular service is not only determined based on information such as current load of a service, but also past invocations. However, the complete invocation graph (e.g., interactions between composite services) may not always be available in large-scale distributed systems. A short overview of different ranking approaches in networks of Web services was presented in (Gekas 2006). Constantin et al. (2006) proposed a distributed ranking algorithm to determine the importance of Web services based on interaction logs.

2.3.2 Tagging and Search

The focus of our research is to determine the expertise-level of users in Web-based collaborations. In the context of the Web, (Becerra-Fernandez 2006) described a system and ontologies expressing skills of experts. Furthermore, (Aleman-Meza et al. 2007) proposed the combination of diverse semantic information to derive skill profiles and expertise of users. However, these works did not discuss how to obtain information regarding users' expertise in an automated manner.

Recently, models for collaborative tagging have been presented (Cattuto, Loreto, and Pietronero 2007) and personalized recommendations based on tagging models (Byde, Wan, and Cayzer 2007). In (Jäschke et al. 2007), the authors presented an evaluation of tag-recommendations in folksonomies using collaborative filtering methods and an algorithm called FolkRank. Heymann et al. (2008) showed an approach for tag-based association rule mining. In contrast, we propose *activity tagging* to determine the expertise and know-how of users. Based on tags, we propose collaborative filtering methods for service recommendations. Li et al. (2008), for example, showed that user-generated tags can be used to discover social interests shared by groups of users.

2.3.3 Global Importance Ranking

We believe that models and algorithms to determine the expertise of users are important in future service-oriented environments. The notion of service-orientation is not only applicable to Web services. Service-orientation in human collaboration is becoming increasingly important. For example, task-based platforms allow users to share their expertise (Yang, Adamic, and Ackerman 2008); or users offer their expertise by helping other users in forums.

By analyzing email conversations, Dom et al. (2003) studied graph-based algorithms such as HITS (Kleinberg 1999) and PageRank (Page et al. 1998, Brin and Page 1998) to estimate the expertise of users. Shetty and Adibi (2005) followed a graph-entropy model to measure the importance of users. Karagiannis and Vojnovic (2008) presented an email analysis in enterprises, defining information flow metrics in the social interaction graph.

Zhang et al. (2007) followed a graph-based approach and applied HITS as well as PageRank in online communities (i.e., a Java question and answer forum), naming their approach *ExpertiseRank*. More precisely, the Java form and discussion threads were used to capture human interactions. We cite some results below:

- The authors found that structural information in human interactions can be used to determine the expertise of users.
- Structural characteristics matter when social network-based algorithms are used for expertise ranking.

- PageRank did nearly as well as human raters in terms of estimating the expertise of users.

The analysis of social networks has a long history of research in social sciences. However, most studies usually focus on the structural characteristics of such networks. In a similar manner, *ExpertiseRank* mainly uses structural characteristics of the human interaction network to calculate the users' importance.

While the above cited works attempted to model the importance of users based on interactions and information flow; they ignore the fact that interactions typically take place in different *contexts*. In contrast, we propose a model where expertise analysis is performed in context-dependent subgraph partitions. Rodrigues et al. (2006) as well as Tang et al. (2007) studied subgraph extraction and visualization in co-author networks using the DBLP computer science bibliography. Furthermore, Conyon and Muldoon (2006) proposed PageRank in weighted bipartite graphs.

2.3.4 The PageRank Model

We provide a brief overview of PageRank and related literature because our ranking model relies on the probabilistic formulation of the "Random Surfer Model". PageRank (Page et al. 1998, Brin and Page 1998) is an eigenvector centrality-based model for ranking documents on the Web. The basic idea is that the importance of a Web page v depends on the citation links pointing to v. Recently, much research has been performed to better understand the theoretical foundations of the PageRank model, e.g., Bianchini et al. (2005) and Brinkmeier (2006). See the work of Berkhin (2005) for a state of the art review in this area.

Personalized PageRank: In PageRank, the "Random Surfer" follows the outlinks of a Web page with probability α, usually a value between 0.8 - 0.9 according to Page et al. (1998), or with probability $(1 - \alpha)$ "jumping" to a randomly selected Web page.

Richardson and Domingos (2002) introduced the *Intelligent Surfer Model*, arguing that users do not select Web pages at random. Haveliwala (2002) proposed a topic-sensitive ranking model by computing personalized PageRank vectors using different categories of Web pages. Topic-sensitive ranking scores can be aggregated into a composite score at query time. White and Smyth (2003) defined a variant of this model called PageRank with priors. Beyond topic-sensitivity, Jeh and Widom (2003) showed that personalized PageRank vectors can be decomposed to compute personalizations for individual Web pages. Recently, Fogaras et al. (2005) and Chakrabarti (2007) proposed Monte Carlo methods to compute fingerprints of personalization vectors.

Personalization in PageRank-based link analysis is an important tool because the importance of nodes in a graph (e.g., humans as part of the interaction network) can be customized based on context information. However, to our best knowledge there is no existing work in the area of personalized PageRank in human interaction analysis.

2.4 Complex Systems

It is important to understand mixed systems of human and software services as complex systems. Both, the technological and social aspects shape the operation constraints of a system (Kleinberg 2008). We believe that such operation constraints of complex systems lead to certain runtime behavior, which cannot be modeled in terms of "prewiring" of interactions based on role models, but rather by observing the natural evolution of cooperation in real systems. Fundamental issues in cooperative systems and (business) processes are *monitoring* of human tasks and *reputation* mechanism.

Vamos (1983), for example, formulated a control system perspective: "Starting from various technologies, needs, evolution trends, and realization possibilities, a new perspective of system architecture is evolving that feeds back its revolutionary effects, not only to technology but to a broad spectrum of human activity." This evolutionary perspective is important when designing and modeling systems. For example, users must be able to change and adapt their activities based on interests, available services, and expertise. The problem of composition is strongly related to organization and control. The key principles of autonomic computing (e.g., see the (IBM 2005) whitepaper for an overview) aim at supporting systems featuring `self-*` properties. In this work, we employ monitoring of human and service interactions to give recommendations for the design of HPSs and execution of human tasks.

However, expertise and human activity change depending on the environment and context. Thus, human dynamics must be considered when determining reliability, reputation, and expertise of users and HPSs in complex systems. It is therefore important not only to model human interactions in ad-hoc and process-centric systems, but also to understand how people are connected (Shi et al. 2008, Yang et al. 2008) and how information flows are influenced by structure (Rosvall and Sneppen 2006, Rosvall and Bergstrom 2008).

Human dynamics play an important role when modeling ranking algorithms capturing the dynamic nature of interactions. Barabási (2005) introduced models to capture the dynamics in human communications (see also Barabási (2007)). Human dynamics has been studied by observing various interaction and communication mechanisms. For example, Oliveira and Barabási (2005) analyzed Darwin's and Einstein's communication patterns of correspondence and showed that today's email exchanges follow the same scaling laws. Analytical models based on queuing theory and human activity have been proposed in (Vázquez et al. 2006). Onnela et al. (2007) introduced metrics to measure the dynamics in human communications in cellphone networks. Based on visitation patterns of a news portal, Dezsö et al. (2006) showed that these information access patterns follow a power-law distribution.

Chapter 3

Interaction Models

3.1 Abstract

We provide basic definitions of interaction models in human collaborations. A short introduction of common collaboration and interaction models is given to provide the context for more advanced interaction models in service-oriented environments. The term interaction makes an explicit statement regarding the number of actors involved, which must be equal or greater than two. Involvement in an interaction can mean active participation, or just being affected by, or possibly even only being aware of interactions.

Within the inContext project, we defined interactions as: A human interaction denotes any interpretable action or event that allows us to establish a link between actors based on which measures can be taken to understand and enhance collaborations. We present the HPS interaction model and discuss different degrees of formalism in human interactions. Furthermore, we introduce a task-execution model serving as input for recommendations of the right person to perform tasks in collaborations.

3.2 Common Interaction Models

- Synchronous communications: The interaction model in conventional telephone communications between users. Both users are involved in the interaction, regardless of the initiating user. Other combinations are possible such as multi-party calls or calls supporting rich media, for example, IP-based calls.

- Asynchronous communications: The most popular messaging tool is email enabling "offline" or asynchronous interactions between users. Considering a single interaction (i.e., message), we speak of one-to-many interactions. Instant messaging (IM) has similar characteristics in terms of a) message-cardinality (the number of recipients

of a particular message) and b) the addressing scheme of people using URIs. In contrast, messages are exchanged (or delivered) in near-synchronous mode.

3.2.1 Levels of Participation

The very basic categorization of participation levels is *active* and *passive* participation in interactions. Here, active participation means an observable action in a certain *interaction context*. As an example, a user sends a message regarding a certain topic to one or more users; more generally, initiates any kind of interaction in a certain context. Various factors may influence the context of an interaction, for example, in the scope of a project, a certain activity in a project, or even an activity in the scope of multiple projects. On the contrary, passive participation means that no observable action was recorded or logged by the system. In this research not only explicit collaboration between actors is considered, but also actions by individuals that effect other actors, and even such actions that raise or increase awareness between actors without active participation in a collaboration (Gombotz et al. 2006).

3.2.2 Roles

Role models may be defined for different interaction levels and for different contexts. On the one hand, roles in human interactions can be derived based on properties such as *initiator* of an interaction or active collaborators as *contributors* (e.g., in discussion threads or collaborative editing of documents). For example, the WS-HT specification defines a *Task Generic Role* model for B4P (see Fig. 2.1 and (Amend et al. 2007, Section 3.1)). In (Dorn et al. 2007), various views on teams including organizational structure, projects, and interactions are given. These views represent team-centric roles and their properties in collaborations. While roles can be modeled and represented explicitly, Dustdar and Hoffmann (2007) proposed *interaction mining* to detect roles in collaborations based on *patterns*. An example of pattern-based collaborations and context-dependent roles, for example, *observers*, has been presented in (Schall, Gombotz, and Dustdar 2007).

3.3 Activity-centric Collaboration

Activities in collaboration are commonly understood as steps in a workflow. Within the scope of the inContext project, we considered semi-structured collaboration scenarios without predefined control and data flow models. In contrast, formalized processes in workflow systems demand for precise control and data flow models. Therefore, such models represent well-understood processes of stable business activities.

The assumption we make in this work is that collaborations are comprised of activities and actions, which are performed by human actors. These actions are conducted in

a particular context (e.g., people working on projects), thereby resulting in interactions with other team members. Examples of such activities include "writing/sending emails", "making phone calls", and "creating/editing documents". We can think of these activities as *ad-hoc activities*, which emerge on demand rather than following a predefined work-plan. Being able to perform ad-hoc activities is a vital part in collaboration. It fosters creativity and awareness in teams, the ability to solve unforeseeable problems, and allows team members to adapt to changes by optimizing their actions. As the next step, we show use cases in the following section to illustrate how users can apply activities in dynamic collaboration scenarios.

3.3.1 Activity Use Cases

It is useful to show some use cases of activity-centric collaboration before going into technical details; how activities are modeled and the relation to HPSs. An activity can be performed many times resulting in *activity instances* being created. Each instance corresponds to an *activity declaration*, which defines at the very basic level the activity name, URI, and resources that are used to perform activities. A complex activity is composed of sub-activities, which are the basic elements of an activity structure. For example, an activity "finalize document" may be composed of two sub-activities, "submit for review" and "send instant message". These sub-activities are basic activities that can be executed using service-oriented architectures. Basic activities are defined as *activity types*. A simple example, "send instant message" may have a `To` property (the recipient URI) and a `Text` property (plain text associated with message). Throughout this work, activity declarations depict basic activities that may have an associated activity type. Figure 3.1 shows example use cases, which we will describe in detail in the following — discussions and models are based on (Schall, Gombotz, Dorn, and Dustdar 2007).

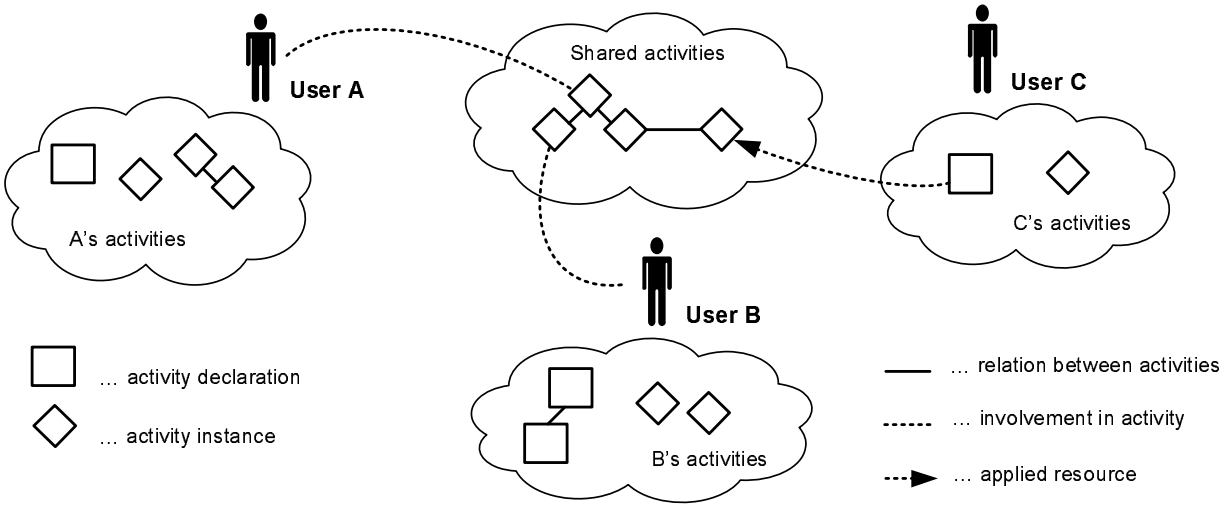


Figure 3.1: Use cases of activity-centric collaboration.

Each user may have a set of private or protected activities which are not visible to all users. Let us first discuss the role of User A and User B in Fig. 3.1. User A has three activity instances in his workspace (illustrated as a cloud); two activities are related to each other, for example, as parent-child activities. In addition, User A's workspace contains an activity declaration. As mentioned before, a declaration may correspond to one or more activity instance(s). User B works on two activity instances, which are not connected to each other. In B's workspace, we see activity declarations that are related to each other. For example, a complex activity can be refined by adding sub-activities as child elements. User A and User B can share selected activity instances by moving instances from their private workspace to a shared workspace (`shared activities`). Also, users can create new activity instances in a shared workspace, which are visible to all users that have permission to access the workspace.

Now, we illustrate two use cases to demonstrate how User C could integrate his capabilities as HPSs. Both cases presume an existing team which organizes work using activities — User A and User B in Fig. 3.1. We assume that User C offers an HPS. (A detailed knowledge of the HPS interaction model is not needed at this point).

Automatic discovery. Suppose the team comprised of User A and User B instantiates an activity named "evaluate open-source solution". This activity requires the knowledge of a user familiar with the specified open-source project. Let's assume that User C has not yet joined the team and that C is not available at the time the activity is instantiated. Furthermore, potential collaborators should have worked on similar evaluation activities in the past, for example, whether or not the open-source project can be used in an existing infrastructure. In this use case, User C has expertise in this particular type of activity and offers a "consulting service" to make his expertise available. Thus, C's capabilities could be integrated into an existing activity-based collaboration structure.

However, since C is not available at the time of instantiation, A and B use an HPS registry to indicate the need for this type of consulting service. Notice, multiple users with different levels of expertise and skills can provide the same type of service. Also, a user can choose which service he or she wants to provide.

Consequently, a "flag" — *discover and notify* — is set for the service in this activity (a publish/subscribe mechanism at the technical level). Once User C becomes available, the demanded HPS enters the online state and C's consulting service can be integrated into the aforementioned "evaluate open-source solution" activity. Thus, the execution of the activity can be continued. In (Schall et al. 2007), we presented a similar use case; additionally emphasizing the use of context information in the discovery process, for example, location context and proximity of HPSs.

An important fact in this use case is that User C need not be aware of the activity structure shared between User A and User B. In particular, User C may have declared an HPS as an activity which can be used in many different collaboration scenarios. Thus, C contributes his expertise in the scope of a specific activity without being aware of the "global" collaboration context. However, from C's point of view, interactions in this specific context are also represented as an activity instance. Such partial views on the collaboration

context can be observed in many real-life collaboration scenarios, for example, due to hierarchical structures, role models, and trust-based access control.

To finalize our discussion on automatic discovery, it is important to note that the presented use case shows human interactions in a *dynamic* and *open* environment. Users can contribute their expertise as services, which can be used for human interactions in the context of activity-centric collaboration. An example of such HPSs are "consulting services" that can be discovered and integrated into collaborations as the demand for such HPSs arises. *Reputation*, *trust*, and *reliability* are of major importance to realize this vision. The interesting perspective in this context is to study the dynamics and evolution in such environments, e.g., see (Lieberman et al. 2005).

User-driven integration. This use case is strongly motivated by *social-effects* in collaborations. While we previously assumed a scenario where users perform activities in the absence of detailed knowledge about the nature of the collaboration (activity) structure, here we motivate the need to establish a *social context* in activity-centric collaboration. Let us assume that A, B, and C share activity declarations. For example, A can view B's activity declarations and vice versa. Now, there is a need to process a *pending activity*. Like in the previous use case, a pending activity could be "evaluate open-source solution", which has been instantiated but is not assigned to a user or HPS. We regard pending activities as items in a *work queue* that need to be processed by the team. Notice, at this point the team is still comprised of A and B, while C is a potential collaborator to perform the activity marked as pending. In this given collaboration setting, we have the following options to integrate HPSs into the collaboration structure:

- Let us assume that User C is aware of the collaboration structure (Fig. 3.1). He decides to join the shared activity workspace — the ongoing collaboration between A and B — by using one of his *own* HPSs in the instantiated "evaluate open-source solution" activity.

  Indeed, C needs to have the permission to claim and process the activity. Apart from access rights that can be enforced through role models, reputation and expertise are key indicators to determine whether C should be able to claim a particular activity or not. When referring to explicit constraints (e.g., role-based access models), we speak of *functional properties*. *Non-functional properties*, on the other hand, are based on soft-constraints such as expertise. In other words, if a set of users is eligible to perform an activity, non-functional properties are used to decide which user is best suited for the activity at hand. The decision — user selection — is made either automatically by the system (e.g., an activity management service implementation) or administered by a user; for example, a super user responsible for the top-level activity and its child elements (e.g., User A in Fig. 3.1).

- Another strategy User C may choose is to decide to *adopt* an existing activity declaration. User C can select an existing activity from A or B, or decide to adopt an activity from the outside world. However, which strategy C will choose is in fact

more subtle than it may seem at the first glance. The question is whether A, B, and C are sufficiently (self-)*similar*.

As an example (based on simulations of evolving populations), Riolo et al. (2001) find that "cooperation can arise when agents donate to others who are sufficiently similar to themselves in some arbitrary characteristic". Self-similarity, in a cooperation and collaboration context, means that communities and teams usually share similar interests and features. Intuitively, users with similar interests (and expertise) tend to work on similar activities. Guimerà et al. (2003) studied self-similar community structure in human interaction networks (i.e., based on an email network). The authors find that this network self-organizes into a self-similar structure. Finding such structures is a community-detection problem. Newman (2006), for example, introduced *modularity* as a measure to find the optimal community structure.

Therefore, C will choose from A's or B's activity declarations based on his interest and similarity. The chosen activity declaration is then adopted and integrated into the existing, ongoing activity instance. Notice, an activity declaration can be depicted as an HPS. Such cases are beneficial when A and B have previously not considered to include existing capabilities (e.g., services) into a certain activity.

- An important requirement for flexible collaboration is the ability to create *new* activity declarations and HPS-mappings of such declarations. (The mapping onto HPSs will be discussed after we have concluded the use case discussion.) Such mappings indicate that C considers a particular HPS to be a suitable solution to perform a certain activity. The ability to declare new activities as HPSs allows a user to easily provide capabilities and opportunities to the team that may have not been considered before.

Both use cases will be the motivation for the design and implementation of a novel framework allowing humans to create and use HPSs in various collaboration scenarios. Next, we summarize important features of activity-centric collaboration and basic HPS concepts.

3.3.2 Summary of Basic Concepts

We provide a brief summary of previously introduce concepts before continuing our discussion on HPS interaction models. Our main focus is the use of activities in service-oriented architectures.

Managing collaboration structure. The important characteristic of activity-centric collaboration is that users can create and share ad-hoc (informal) processes by defining the steps needed to create the demanded collaboration output. Users can modify activity structures by editing activity steps or sequences. These structures can be defined as private or shared activities. Activities help to establish awareness among coworkers by letting users share status information (progress of activities) or refinements of ongoing activities.

Maintaining activity knowledge. Shared activities allow teams or communities to establish a common understanding of steps and/or useful resources in specific collaborations; thus fostering reuse of activity declarations and complex activities (i.e., a set of structured activities). Shared know-how defined as activity-structures increases the efficiency in collaborations and allows users to define best-practices. This is very much in the spirit of Web 2.0 platforms, which enable users to jointly edit and provide content. However, activities are used to structure collaborations and interactions; thus, they can be executed using service-oriented architectures and adapted to satisfy the requirements of the actual environment.

Activities in service-oriented architectures. In this work, we are mainly interested in the application of activities in service-oriented architectures. Specifically, activity declarations and corresponding type definitions enable HPSs by mapping (human) activities onto Web services.

Notice, Web services can be used in activities, for example, "send instant message" using an instant messaging service, or as standardized interfaces — standards such as WSDL, SOAP, WS-Addressing[1] — to offer human capabilities as HPSs. Not activities are provided by humans; only HPSs using Web services technology. HPS (interface) descriptions should not be confused with an API definition to access and manipulate a human activity, for example, an activity management service. HPS descriptions are based on declarations of activities. Thus, HPSs are not general purpose definitions of human capabilities.

Activities are shared between users to establish awareness in teams and to coordinate work among users. HPSs can be shared and (re)used by users to publish their capabilities as services; for example, on the public Web. An HPS is defined by associating a user's profile to an HPS description, which we call a *personal service*. In this work, we do not define precise ontologies to depict, or categorize, different HPSs. We employ a light-weight *tagging* approach allowing users to associate metadata to HPSs. Such tags are used in the discovery of services.

3.4 HPS Interaction Model

The use of Web services technology to enable human interactions in open environments offers many interesting opportunities such as discovery of HPSs in dynamic collaborations. However, most Web services-related standards and toolkits have been designed to satisfy the requirements of enterprise-grade applications. Such applications demand for reliability, security, and the consistent (transactional) execution of business processes.

The goal of this work differs with respect to such application scenarios. We focus on ad-hoc collaboration scenarios, which typically require flexible execution of human activities, and opportunistic compositions of human and software services. Therefore, we propose a light-weight activity model comprised of the essential concepts enabling humans to manage

[1] `http://schemas.xmlsoap.org/ws/2004/08/addressing`

their interactions in service-oriented environments. Before doing so, we need to elaborate on the different types of interactions in mixed systems.

3.4.1 Types of Interactions

In our previous use case discussions we emphasized the role of activity-centric collaboration in SOA. Here we focus on the different types of interactions maintaining the collaboration setting as introduced in Sec. 3.3.1. In particular, we assume a scenario as illustrated in our **automatic discovery** use case, i.e., a user provides an HPS without being aware in which activity structure the HPS is used. As depicted in Fig. 3.2, we extended the collaboration scenario by introducing a set of human and software services (the cloud below the shared activity instance). Furthermore, User A and User B have toolboxes, which they can use to establish a mapping between activities and services. In other words, a service is used to work on activity instances at run-time. Indeed, a service is usually defined by Web services professionals using an XML-based description language (e.g., WSDL). Thus, an activity declaration also encompasses an (user) interface representation so users can interact with services. One can think of such interface representations as forms or widgets allowing users to specify the (input) parameters of a service request issued by the user and to view the corresponding response (output) of that request. The following discussion are based on (Schall, Gombotz, and Dustdar 2006).

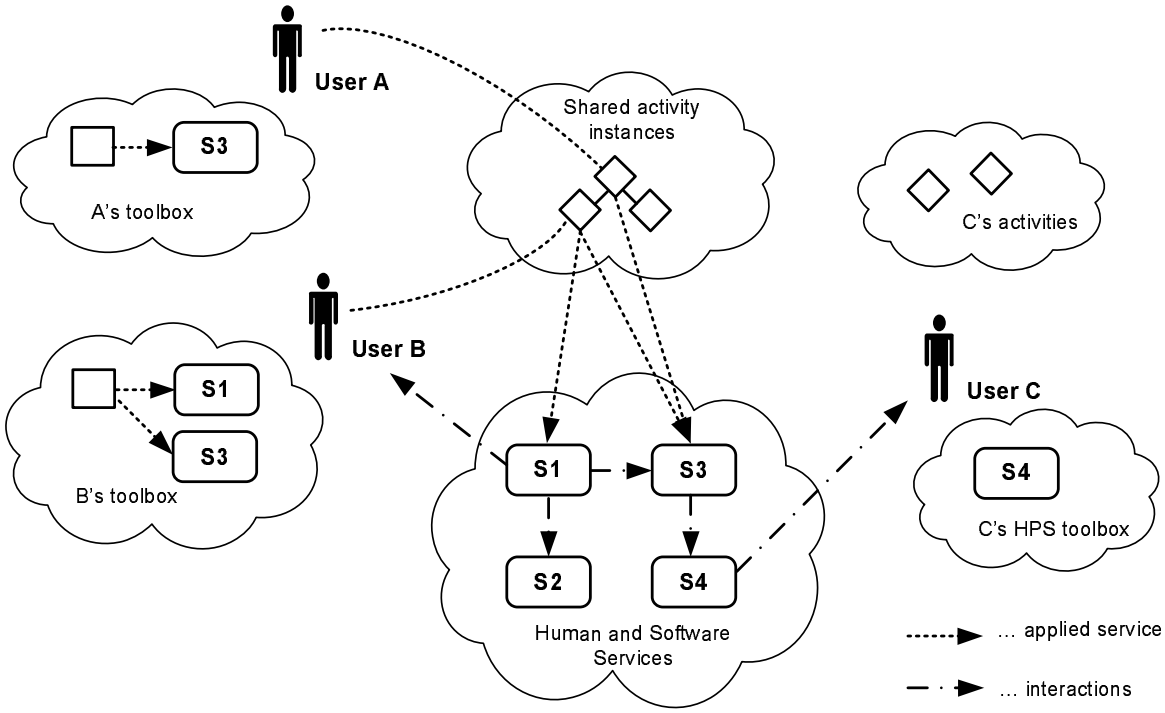


Figure 3.2: Types of interactions in mixed systems.

Let us now define the different types of human and software service interactions in mixed systems (with reference to Fig. 3.2).

Human interactions using software services. People can use Web services to work on their activities. User A, for example, uses S3 to work on specific activities. Prior to working on a certain type of activity, User A associates S3 with an activity declaration (illustrated as A's toolbox). Similarly, User B can create his own activity definitions for S1 and S3. Such compositions are an activity-based mashup of services defined by the user. Mashups allow users to combine data and application logic from different (existing) Web applications or services. The focus of these compositions is to create solutions — for example, B uses his toolbox to create a complex activity composed of S1 and S3 — in ad-hoc and semi-structured collaborations. However, mashups are not only used to create *local* applications, for example, applications used within organizations; mashups have already been applied to create *global* services. For example, research scientists scattered around the globe weaved data together from different sources to show the geographic distribution of ant species by using map services (Butler 2006). From the technological point of view, Maximilien et al. (2008) defined mashups as applications allowing users to solve situational problems.

In our work, mashups are loosely structured compositions of services that are controlled by interactions between users and services. As an example, User A works on the top-most activity instance, which is connected to B's activity (a child-activity). Both A and B work on shared activity instances using a set of services — applied services are S1 and S3.

Service interactions initiated towards humans. Such scenarios include notifications and information "pushed" towards the user. In Fig. 3.2, S1 notifies User B about events with respect to the shared (activity-based) collaboration structure.

As an example scenario, an intelligent meeting scheduling service has been implemented within the inContext project. The scheduling service is composed of a set of collaboration services. The selection of participants, suggested physical location of the meeting, and relevant documents are recommended by a relevance ranking service; thereby helping users to plan and organize meetings. Based on the performed activities, deadlines, or other context-information such as involvement of users and their roles in interactions, notifications are sent by using communication services, for example, email, instant messaging, or SMS. The selection of the most suitable communication service is influenced by the user's current context and preferences. For example, users may be busy or only available in urgent matters. Interaction rules are used to express user preferences as event-condition-action patterns.

Interactions between software services. Such interaction scenarios are found in compositions of software services. For example, S1 is a composed service using S2 and S3 to realize its features. For example, see (Alonso et al. 2003). The interesting aspect in this interaction scenario is that S3 depends on S4 (S3 requests input from S4; depicted as the arrow going from S3 to S4). However, S1 may not be aware of this dependency. Thus, it is important to account for this dependency when negotiation of the expected QoS-level (service-level-agreement) between S1 and S3 takes place.

Human interactions as part of (software) service compositions. Many business processes require human input as part of their regular execution. We briefly highlighted such interaction scenarios and the role of B4P in Sec. 2. In B4P, tasks are used to interface with people.

When the BPEL engine reaches an (people) activity, a `People Query` (for example, as previously depicted in Fig. 2.1) is used to retrieve a set of people, which are selected from an organizational directory to work on tasks (Leymann 2006, Section 7). The result of those tasks is passed back to the BPEL engine and the business process continues its execution. Typically, a "global" view is assumed and therefore interactions between various human and software services are modeled at design time using, for example, B4P standards.

Human interactions using HPSs. The novel concept presented in this work is based on the idea that humans can specify their capabilities as Web services. The notion of activity-centric collaboration attempts to tie into the compositional nature of the Web by providing the fundamental concepts to structure and compose activities in a hierarchical manner. User C, for example, created the service S4 (see C's HPS toolbox), which can be used by other (software) services. However, as mentioned before, C may not be aware of the fact that the HPS S4 is used in a composition of a set of services. Here User C's activity instances are depicted in a private activity workspace. User A and User B have created mappings to software services S1 and S3. Also, S4 can be used by A and B to work on their shared activity instances enabling a "consulting service" to be integrated into existing collaboration structures.

While such compositions of activities and services are *user-driven*, business processes — as mentioned in the previous interaction scenario — perhaps require a detailed definition of tasks, notifications, and roles in advance. For example, in B4P a process designer usually creates a model so that BPEL engines can interface with humans.

3.4.2 Conceptual Model

Previously, we discussed various interaction scenarios in a service-oriented collaboration environment. We motivated the need for activities and the role of activities in SOA. Activities are used for different purposes. On the one hand, people use activities to structure collaborations in a flexible manner. On the other hand, activities enable users to define HPSs. We now turn to the basic activity model comprised of the key concepts enabling the use of HPS in various interaction scenarios. As stated before, interactions may take place between humans (using HPSs) or between (software) services and humans. Both scenarios are enabled by using Web services technology.

However, complex flows spanning compositions of multiple HPSs and software services are not the focus of this work. Thus, the presented activity model depicts the most important elements to support basic interaction scenarios. A detailed description of various resources, for example, HPS WSDL mappings, will follow at a later point when we define the HPS-design process.

Here we discuss the activity-centric collaboration models. In Fig. 3.3, we show the basic model allowing users to create HPSs. Notice, the diamond-shaped UML symbol is a relation to link three concepts. These relations are called *n-ary relations*. In UML class diagrams, diamond-shaped symbols are not used to depict activity instances. The description of the model concepts is given in the following.

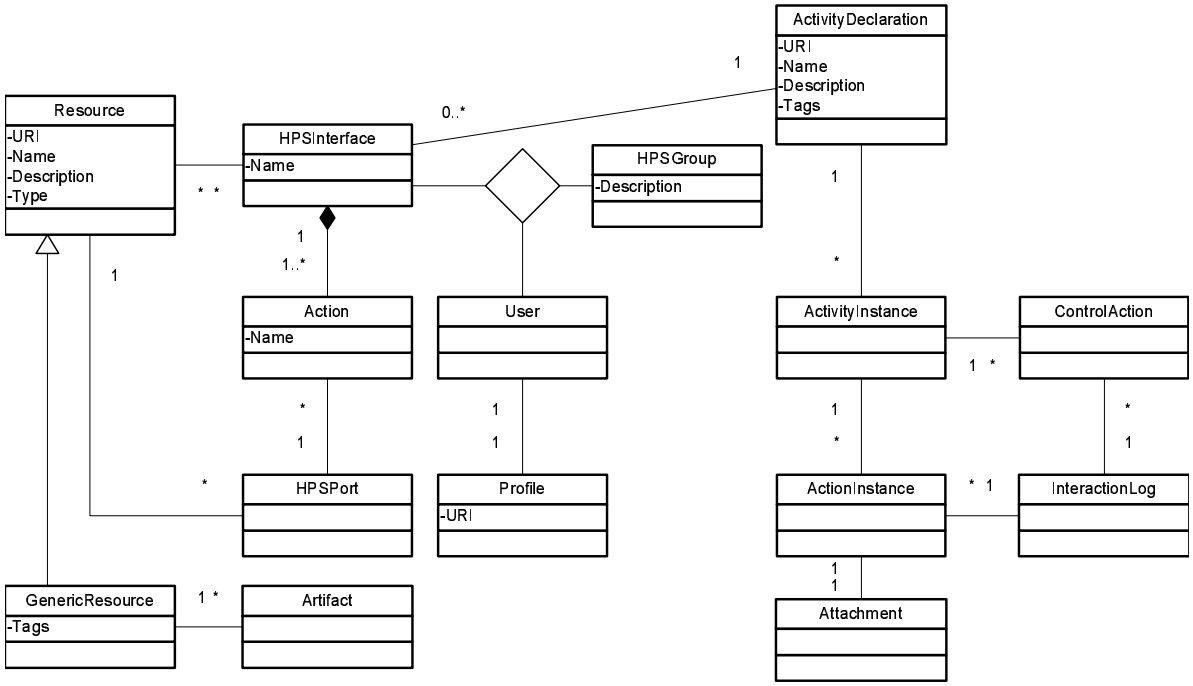


Figure 3.3: Overview of HPS activity model.

- 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 same tag is perhaps used to depict different activities. Some tags might simply be misplaced or not meaningful. Thus, at a later point we will discuss methods to deal with misplaced tags.

- The `HPSInterface` 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" software services. Essentially, 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`. As mentioned before, an HPS *requester* can be a human seeking expert opinion or a composed software service requiring human input. Since many users may provide the same type of service, 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, storing user preferences, and so forth.

- 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.

  Let us briefly discuss why type definitions, or typed messages, are useful. We can use HPS in ad-hoc collaborations by exchanging messages depicted as XML documents. Such ad-hoc interactions are in a manner similar to email-based collaborations. A user can compose a message, attach documents, and send those messages to other users. An email message contains header information, which is parsed and interpreted by an email service; however, the message body is usually composed as plain-text content. It is difficult to route and interpret such messages in a service-oriented environment because the actual content of a message does not have syntactic meaning for software services. Thus, email-based interactions — in the context of HPS — are used to trigger, for example, one-way notifications (e.g., email messages automatically generated by a service).

- 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`.

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 (Dustdar 2004). In particular, 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`. Specifically, each `ActionInstance` is defined by an `Action`. An `Attachment` is something generic to associate XML documents, for example, XML messages that are exchanged between HPSs, 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. In other words, `ControlActions` include *coordination*, *communication*, and *execution* actions that are associated with instances 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.

The next model, shown in Fig. 3.4, depicts an excerpt of the inContext activity model. It has been presented in a similar form in (Schall et al. 2008, Figure 3) and (Dorn et al. 2008, Figure 5). Our use cases (Sec. 3.3.1) described collaboration scenarios where people can organize their work as structured activities. In particular, activities can be hierarchically arranged to reflect "fine-grained" steps in collaborations.

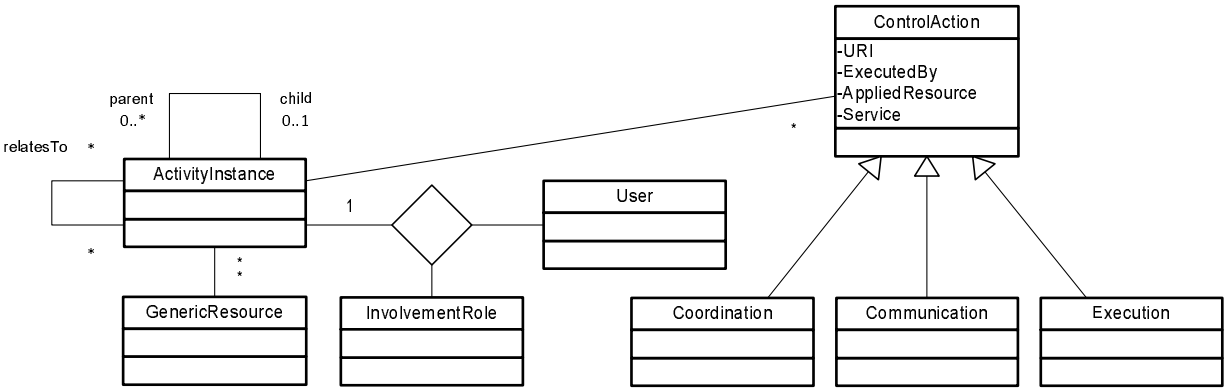


Figure 3.4: Excerpt of hierarchical activity model.

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, 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.

The basic HPS activity model (Fig. 3.3) 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`. The `relatedTo` property 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`. Here we apply the concept of an `ControlAction`, as introduced in (Dustdar 2004), to capture activity-change events, interactions between members, and work carried out. Actions can trigger events describing the progress of activities.

3.4.3 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 3.5 shows task-related concepts and their relation to previously introduced concepts.

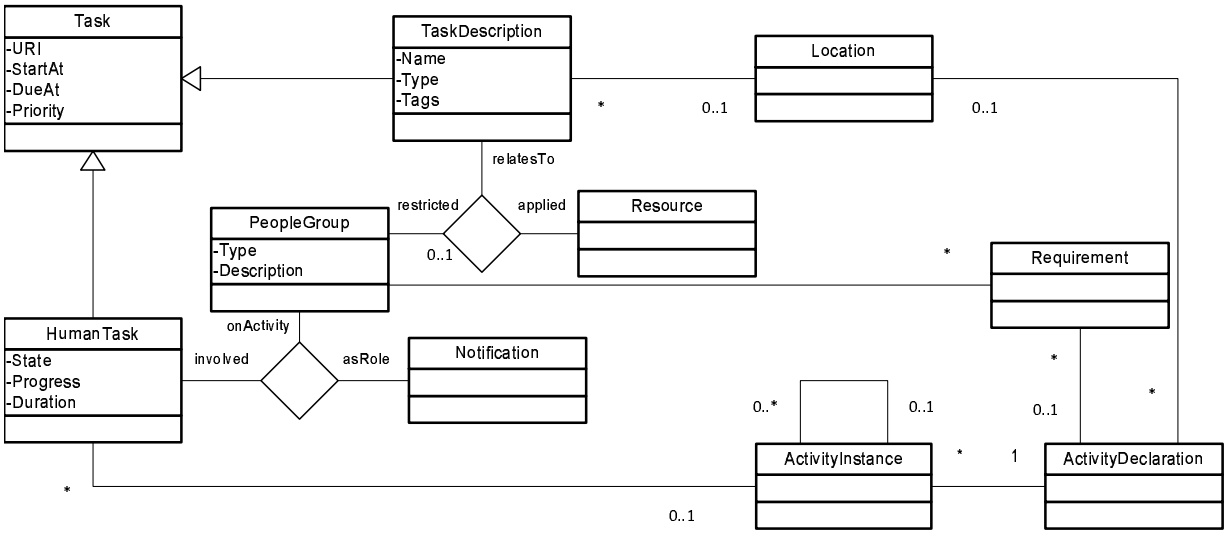


Figure 3.5: Overview task model.

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-HT specification. Moreover, functional properties can be associated with `ActivityDeclarations`, depicted as `Requirement` in Fig. 3.5; 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. 3.5. 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.

The next step is to introduce a *task execution model* defining the possible task states. The task execution model is depicted by Fig. 3.6. It is relevant for both cases, announcements of task and the control of activity executions. Based on this execution model, we will derive task-related metrics capturing the dynamic nature of HPS-based collaborations — these metrics will be defined when we introduce the HPS framework.

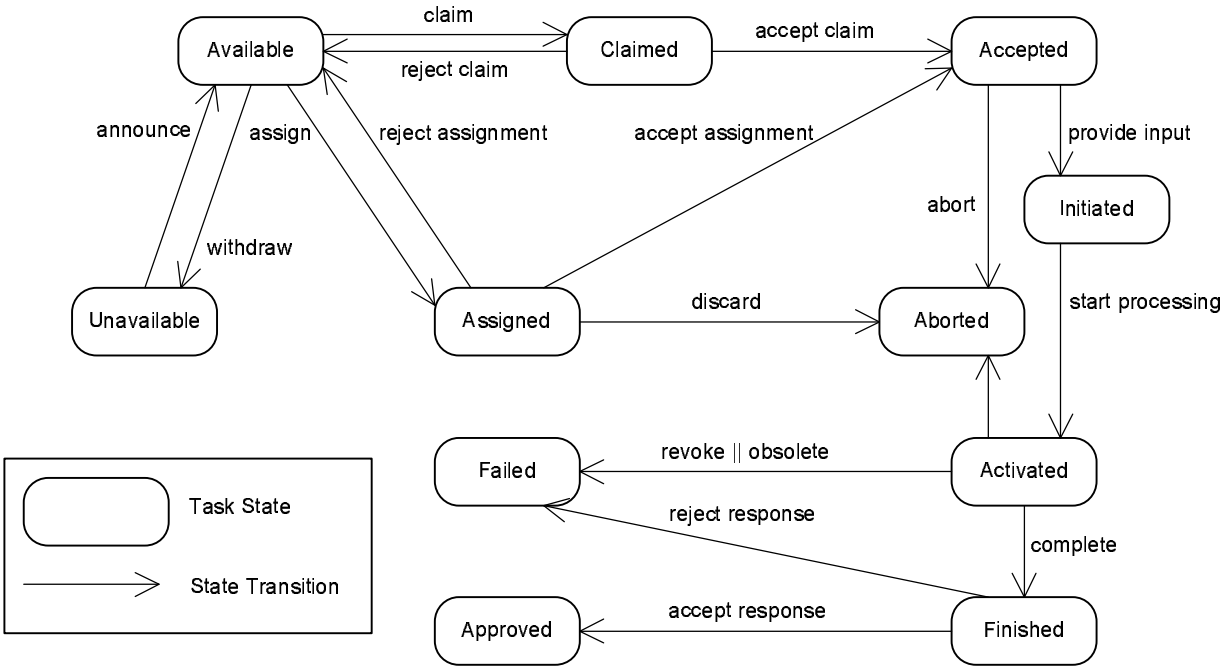


Figure 3.6: Task execution model.

- *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 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. As mentioned before, we will define metrics based on the presented model as well as a *task rewarding model*.

3.4.4 Calling an HPS

The next step in this chapter is to illustrate basic interactions in HPS-enabled environments. That is, using Web services to interact with humans. We use the basic activity model — depicted in Fig. 3.3 — to represent human activities as HPSs. In particular, an `ActivityInstance` represents an activity processed by humans.

In today's collaboration environments, we observe a shift towards "pervasive interactions". A system supporting HPSs must give users the flexibility to deploy user-defined services on a variety of devices. For example, a software stack based on Web services standards to enable HPSs may be deployed on devices, which are not always connected to fixed (wired) or mobile (wireless) networks. Moreover, due to the pervasiveness of services, we can also imagine that a software stack used for HPSs is deployed on — or hosted by — consumer hardware such as standard PCs. In many cases, such devices cannot be accessed from the public Web. To address these challenges, we introduce a) interaction (run-time) models suitable for HPSs and b) a middleware allowing various types of messages to be managed. In this section we show the run-time interaction model.

In Web service environments, a client sends a request to a service and receives a response from either the requested service, or some other service endpoint (see WS-Addressing). In many Web services-based applications, simple service interaction patterns are sufficient. For example, a requester invokes a Web service operation and receives the corresponding response in a synchronous manner. Asynchronous interactions are achieved by, for example, providing callback destinations. The support of asynchronous (interaction) patterns is mandatory in HPS.

In HPS, we need to define a human-centered terminology. If a user decides to offer a service, we speak of a *personal service*. The same service (type) can be provided by multiple users; available as a set of registered services. One can imagine a `Request` and corresponding `Response` as XML documents that are used in collaborations (e.g., input/output of HPSs). A request is a document providing the input information to process a human activity. Actions associate such documents with activities. Recall, these documents are

described as declarations which are detailed in Fig. 3.3. Similarly, a response is the output of an activity.

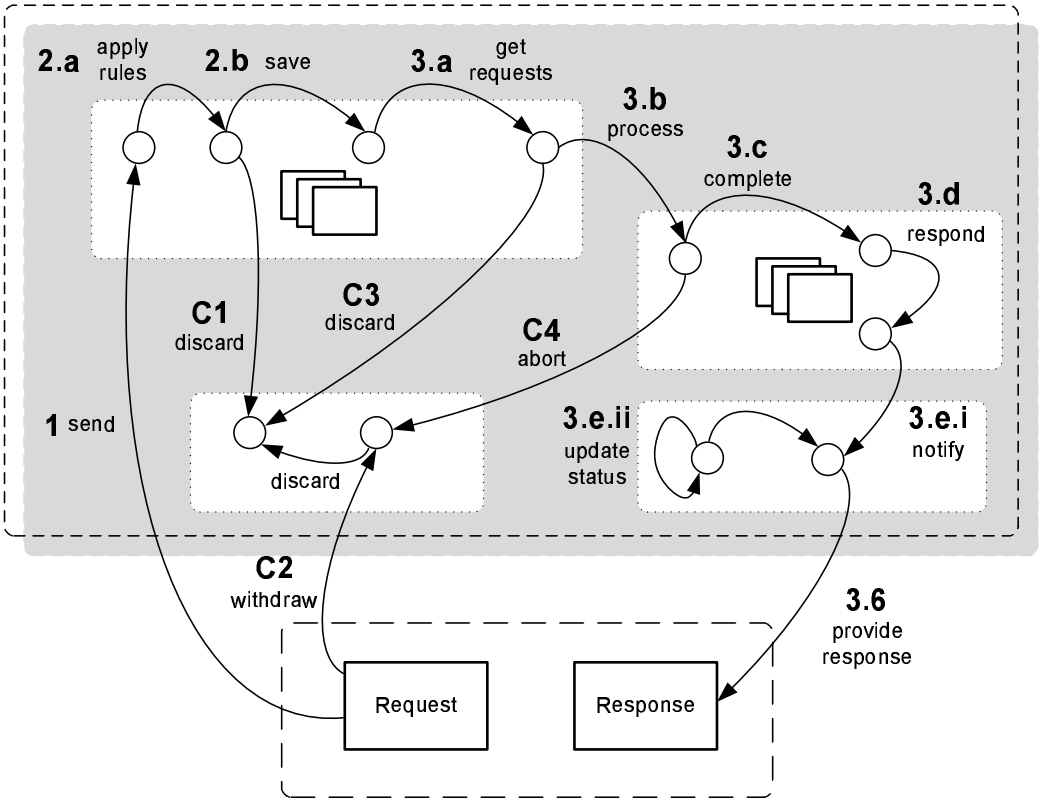


Figure 3.7: Example of HPS interaction as transactional model.

Figure 3.7 depicts the run-time model as a transactional diagram. The steps 1-3 show the normal execution of a request. Steps prefixed with a capital `C` denote the cancellation of a request.

1. A request is sent towards the HPS middleware platform, which offers an interface accepting different types of messages. Thereby, interactions between requesters and HPSs take place in an asynchronous manner by routing messages to the appropriate destination. Routing of messages is based on the geographic location of users, the preferred device, and the (logical) location of the deployed software stack enabling Web services-based interactions.

2. (a) Every request is then passed through a validation phase (`apply rules`) in which an authorization check is performed. The user can specify white/black lists, routing, and interaction rules. White/black lists are used to, for example, prevent certain users (or requesters in general) from interacting with HPSs.

   (b) Requests can be delivered to the corresponding HPS immediately, or through an offline interaction as illustrated in Figure 3.7 (`save`).

3. (a) To start working on a request, users retrieve requests by different means (i.e., push or pull based retrieval — `get requests`).
   (b) Users start working in requests using their preferred devices (`process`). Also, a hosted solution (i.e., a Web portal) is provided by the HPS framework enabling Web-based access to the user's "message inbox".
   (c) Then, a request is completed (`complete`) when the response (document) is ready for delivery.
   (d) The response is delivered to the middleware platform (`respond`). The requester has different options to obtain the response, which also depend on, for example, whether interaction control tasks are associated with a particular request.
   (e) i. If a task is used to share the state of a request (i.e., task state *finished*), status updates trigger notifications (`notify`) to inform the requester.
   ii. Without using control tasks, the exchange of message takes place in a manner similar to sharing a common view on a set of resources. The response is published to a collection of resources which are shared with the requester. The user is doing so by updating the status of a request (`update status`).
   (f) Finally, the requester can retrieve the response (*provide response*).

Cancellation of requests:

- `C1` Requests can be discarded automatically based on predefined rules and filters (e.g., white/black lists).
- `C2` Pending requests, which HPSs have not yet retrieved for processing, can be withdrawn by the requester.
- `C3` A user (offering an HPS) can inspect and discard a request.
- `C4` Both, the requester and user (HPS) can cancel a request being processed (`abort`).

3.4.5 Capturing Human and Service Interactions

A fundamental role in this thesis plays the analysis of human as well as service interactions. Mining of interactions is not only important in ad-hoc collaboration scenarios, but also to monitor deviations in formalized processes. The ProM framework, for example, is a process mining tool supporting the analysis of event logs in workflow management system, e.g., see (van Dongen et al. 2005), (van Dongen and van der Aalst 2005). Using process mining tools, we better understand if changes in the process model are necessary (Günther et al. 2008). Dustdar et al. (2005) introduced process mining in the context of ad-hoc flows in dynamic environments. Here we focus on logs captured in interactions between humans in SOA and email-based communications. Typically, a single source (i.e., service) cannot capture all relevant logs and collaboration data. In this work, interaction logs are obtained from email and instant messaging (IM) based interactions; and collaboration services such as a document portal.

3.4.5.1 Email Mining

Although services play are increasingly important in collaboration, email will remain one of the core communication means in the foreseeable future. Thus, we extract human interactions from email repositories. Emails contain the following significant information: (i) *sender and receivers*, the participants in interactions; (ii) *subject*, enabling the extraction of contextual information, for example, in which context a particular email message was exchanged; (iii) *reply-to*, to identify message threads; and (iv) *email attachments*, the applied collaboration resources and artifacts.

3.4.5.2 Raw Logs from Human-Service Interactions

Figure 3.8 shows a fragment of a logged human-service interaction obtained from a shared-workspace service.

```
guid 11:20:17 server incontext wp5/forms/allitems.aspx userID Mozilla/4.0
guid 11:20:23 server incontext wp5/forms/upload.aspx userID Mozilla/4.0
guid 11:21:30 server incontext wp5/t5.3 impl/design_12.doc userID Mozilla/4.0
guid 11:21:30 server incontext wp5/forms/allitems.aspx userID Mozilla/4.0
```

Figure 3.8: Example log from user interactions captured from share-workspace service.

Each line in the log file contains: (i) a unique id (`guid`), (ii) a timestamp, (iii) the server identifier, (iv) the top-level site, (v) the resource on the server, (vi) the user id, and additional information from the requesting client such as browser type. The given example shows a *document upload action*, denoted by the `upload` identifier. Based on these raw service logs, we can automatically establish correlations between upload actions, documents that are used in collaborations, and involved members, for example, people that are notified via email about the availability of a specific document in a shared document workspace (assuming that email messages and service logs can be correlated).

3.4.5.3 Activity-Event Logs

An activity management service is a representative source for notifications and activity-change events. Together with email and raw service logs, these sources cover a significant spectrum of coordinative and communicative work aspects in most teams (Dorn et al. 2008). Listing 3.1 shows an example of an activity-change event.

```
<ControlAction xmlns="http://in-context.eu/action" xsi:type="Coordination"
  URI="Coordination#6564" DescribesActivity="Activity#222">
<From>http://in-context.eu/User/User1</From>
  <CoordinationType>Delegation</CoordinationType>
  <To>http://in-context.eu/User/User2</To>
</ControlAction>
```

Listing 3.1: Example action event log captured from activity service.

Based on human and service interaction logs, we can establish a graph. Figure 3.9 shows an example instance of a bipartite interaction graph that is based on user interactions, activities, and logs of above mentioned events. (See also Guimerà et al. (2007) for bipartite networks.) Diamond-shaped nodes represent activity instances. The line-thickness of each edge is based on the action count.

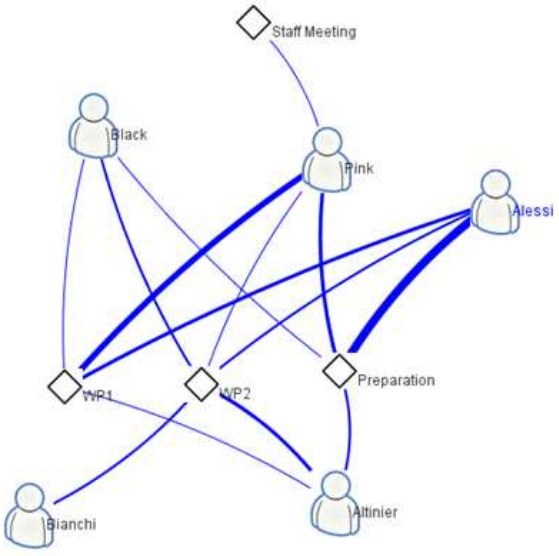


Figure 3.9: Bipartite graph capturing activities, user involvement, and control actions.

The important property of this graph representation is that interactions are always *context-dependent*. An alternative way of representing context-dependent interaction graphs is to tag *interaction links* with context identifiers. As mentioned before, context information can be extracted and combined from various sources to establish the interaction links between users.

3.4.6 Summary of Interaction Models

To summarize the presented interaction models, there are different degrees of formalism in HPS-based collaborations. The formalism needed to describe interactions — in terms of creating explicit process models to depict collaboration flows — is (i) a function of the *requirements* and (ii) is *constraint* by the *degree of complexity*. Complexity is related to the amount of information needed to describe the system (Kolmogorov 1983). Here the collaboration spectrum ranges from ad-hoc (informal) to formalized process models (Dustdar 2004, Schall et al. 2008b). In Table 3.1, we overview the degrees of formalism (Schall, Truong, and Dustdar 2008a).

Ad-hoc	Interactions take place in an ad-hoc manner if collaboration flows are not modeled in advance. As an example, interactions between two actors — a requester of an HPS and the user offering the service — take place by exchanging a set of XML documents and messages.
State-awareness	Tasks are used control the status and lifecycle of interactions. Certain constraints such as start-time (the latest point in time when a user has to start the execution of an activity) or deadlines (latest completion time of a given activity) can be associated with interactions.
Process-centric	A process is usually established to automate reoccurring activities; enforcing a certain flow of interactions. In collaboration systems, a process describes steps in a workflow composed of a set of activities to coordinate people and software services.

Table 3.1: Overview degrees of formalism to control interactions and collaborations.

For example, if the requirement is to create a model for stable, well-defined, business processes; we need to apply a process-centric systems perspective. However, if we assume an *open-world* of human and software services, we need to be aware of the constrains in terms of complexity related to the information needed to describe the mixed system. At this stage, the focus of our work is to enable HPS-based interaction scenarios with the focus on ad-hoc and state-aware collaborations. For users, the only requirement is to define *human activities* (at design time), which can be mapped onto specific HPSs. At run-time, a request to perform certain activities is realized by using Web services technology. In contrast to existing workflow systems, tasks or formalized process models are *optional* depending on the desired application or composition scenario.

Chapter 4

Global Importance Ranking

4.1 Abstract

In HPS, humans can interact with one another by using Web services. Moreover, compositions of services require in many cases the interplay of human and software services. Regardless of the interaction scenario, the *HPS requester* or *expert seeker* must be able to find the right person based on a set of skills and the expertise level of a user. We hypothesize that link-analysis in human interaction networks is a suitable method to determine the expertise of users.

We introduce a *link intensity*-based ranking model to recommend relevant users in collaborations. In open and dynamic environments on the Web, it is important to determine expertise and skills of users in an automated manner. Additionally, the ranking model must consider properties such as availability, activity level, and expected informedness of users. In this section, we present *DSARank* to estimate the relative importance of users based on the concept of eigenvector centrality in networks. We test the ranking model in real human interaction networks including email conversations and telephone communications in mobile phone networks. The results show that DSARank is better suited to calculate the importance of users in collaboration networks than traditional degree-based methods.

4.2 Preliminaries

Social scientists and physicists have been interested in the structural properties of networks for many years. A large number of *centrality metrics* have been proposed to measure the structural importance of nodes in networks. In social sciences, for example, such centrality metrics are used to study the role and importance of different actors in social networks (Wasserman and Faust 1994). However, it has been recognized that networks evolve over time (Barabasi and Albert 1999, Watts 1999). A lot of recent research in complex systems

views networks as dynamical systems comprising nodes that follow their own rules of behavior, and the edges between nodes represent dynamically coupled information flows (Newman, Barabasi, and Watts 2006, Chapter 1 (Networks as dynamical systems)). Before discussing our importance ranking model, we need to fix some terminology related to interaction models.

4.2.1 Basic Terminology

The term *interaction model* has different semantics depending on the context of the discussion.

Technical models describe the set of rules governing the arrangement and interconnections of elements. Interaction rules in a technical sense are, for example, message exchange patterns such as request/response — if the requester initiates a message, the provider responds with a message or fault.

Dynamical systems exhibit rules or models of cooperation — how we expect interactions to take place. As an example, Hamilton's rule, "I will jump into the river to save two brothers or eight cousins," is an interaction rule to depict the probability that cooperation is favored between related actors (Nowak 2006).

In this chapter we focus on interaction models in dynamical systems to answer questions such as — who is the best informed user in a network?

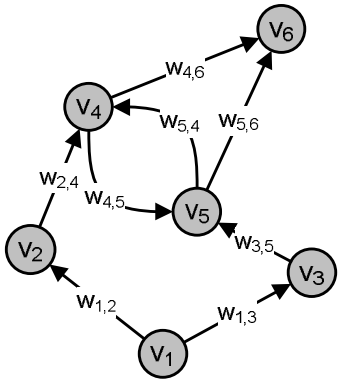
Let us start with the definition of some basic concepts. The interaction graph is defined as $\mathcal{G} = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$ the set of vertices and $E = \{e_1, e_2, \ldots, e_n\}$ the set of directed edges between vertices. Given an instance of $\mathcal{G}$, we create a row-oriented adjacency matrix of that interaction graph

$$A_{i,j} = \begin{cases} 1 & \text{, if } i \text{ is connected to } j \\ 0 & \text{, otherwise} \end{cases} \tag{4.1}$$

Typically, an edge has a weight, which can be calculated by, for example, counting the number of interactions. Suppose that W is a weighted adjacency matrix whose weights are positive entries $w_{i,j} \geq 0$ satisfying

- $w_{i,j} > 0$, if a directed edge connects i to j
- $w_{i,j} = 0$, if there is no edge between i and j

In Fig. 4.1 we show a) an example interaction graph and b) the corresponding weighted adjacency matrix W.



$$W = \begin{bmatrix} 0 & w_{1,2} & w_{1,3} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{2,4} & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{3,5} & 0 \\ 0 & 0 & 0 & 0 & w_{45} & w_{4,6} \\ 0 & 0 & 0 & w_{5,4} & 0 & w_{5,6} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.1: Directed interaction graph (left) and matrix representation (right).

4.2.2 Notation

In the following, we will present graph-based ranking models that are inspired by PageRank concepts. While our approach is not limited to interactions based on human collaboration networks — for example, our ranking model can be used to analyze interactions in service networks — we focus mainly on human-based interactions. In this case, $\mathcal{G}$ is composed of vertices occupied by the set $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ of users. Throughout this work, we depict a weighted edge as the interaction link ℓ. Depending on the context of the discussion, we define whether $\ell(u)$ is an out- or incoming interaction link. Furthermore, by following the PageRank model, we create a transpose matrix (that is, a column-oriented matrix) based on the row-oriented description of W. Let us define the column-stochastic transition matrix I

$$I_{j,i} = \text{probability of transitioning from node } i \text{ to node } j. \tag{4.2}$$

Column-stochastic means that the sum of all column elements equals 1. Table 4.1 gives an overview of operators and descriptions of basic metrics in interaction graphs. An edge $e \in E$ points from v to u, if $e \in \texttt{outlinks}(v) \cap \texttt{inlinks}(u)$. In other words, ℓ denotes an interaction link between users, which may comprise many interactions that are aggregated into a single link.

Operator	Description
$\texttt{inlinks}(u)$	Denotes the set of u's inbound interaction links. For example, $v \in \texttt{inlinks}(u)$ has *initiated* an interaction towards u. The set cardinality $\lvert\texttt{inlinks}(u)\rvert$ is given as $\texttt{indegree}(u)$.
$\texttt{outlinks}(u)$	All interactions initiated by u towards other users are denoted by the set $\texttt{outlinks}(u)$ and similarly, $\texttt{outdegree}(u)$ is the number of links in the set.

Table 4.1: Basic graph metrics and operators.

4.3 Human Interaction Networks

We study link-based ranking models using two types of interaction networks. On the one hand, we experiment with different ranking algorithms based on an interaction network, which we establish based on cell phone communications between users in a mobile phone network. The cardinality in human interactions is one-to-one because the used dataset does not comprise conference calls among users.

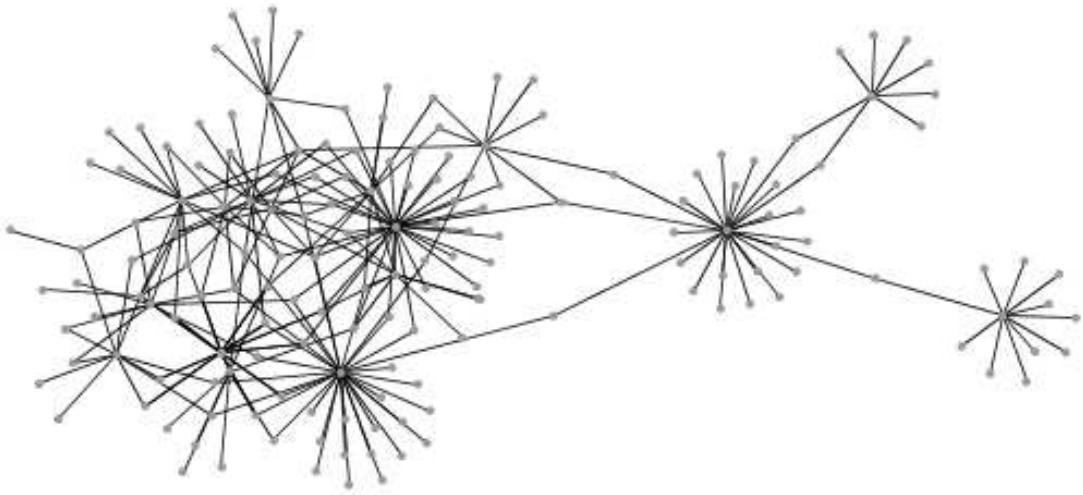


Figure 4.2: Example interaction graph of point-to-point human communications in mobile phone network.

The clustering coefficient is a measure to determine whether a graph is a small-world network (Watts and Strogatz 1998). The coefficient indicates whether the graph, or a neighborhood of the graph, is well connected or not. The shown network exhibits a low clustering coefficient of 0.2 due to peripheral users, which are not well connected.

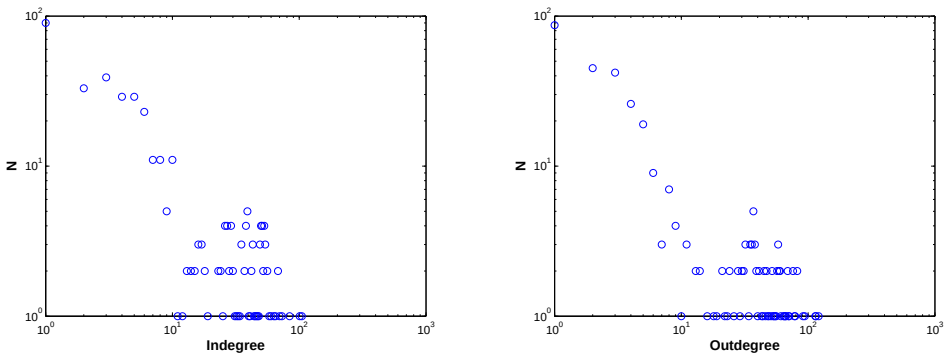


Figure 4.3: Degree distributions of human communications in cellular network: (a) indegree and (b) outdegree distribution of telephone calls in mobile phone network. Vertical axis **N** (counterclockwise rotation) shows the cumulative number of users.

Analyzing human interactions in mobile phone networks is well suited for our ranking approach as we can measure characteristics such as *intensities* of interactions between two users considering underlying properties such as the duration of calls. We use the "Reality Mining" dataset[1], which was made available by the MIT Media Lab, see (Eagle and Pentland 2006).

Figure 4.2 shows an example graph captured in April 2005. The dataset comprises communication, proximity, and location information captured at MIT for the academic year 2004-2005. The public database image comprises both *participants* of the study and *non-participants*. Non-participants are users who did not have any logging software installed on their cell phones, but whose interactions were captured as incoming/outgoing calls on participants' devices. We perform filtering and select users if their degree of incoming links is greater than 1 (if a user interacts with at least two different users). Partial observations of interactions limited to the small set of participants explains the low clustering coefficient in Fig. 4.2. The structure in terms of in- and outdegree distributions is shown in Fig. 4.3.

The second dataset is an email interaction network. Email messages exchanged between people serve as input to establish the interaction graph. Each interaction is a directed link between sender and receiver of a message. The cardinality of email-based interactions is naturally one-to-many as multiple recipients can be specified.

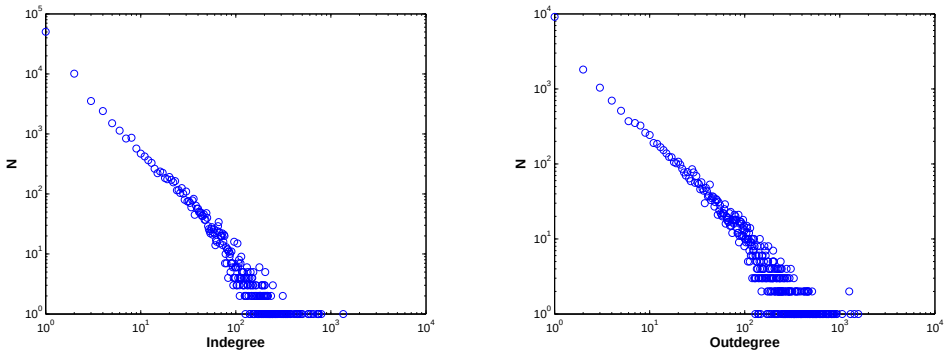


Figure 4.4: (a) Indegree and (b) outdegree of human interactions in email conversations. Vertical axis **N** (counterclockwise rotation) shows the cumulative number of users.

Figure 4.4 shows in/outdegree distributions of an email interaction graph. Both distributions have a power low tail as it can be observed in many structures and dynamics on the Web, for example, the structure in human task networks (Yang et al. 2008), the Web graph (Pandurangan et al. 2006), or the dynamics in human interactions (Barabási 2005). Figure 4.4 describes the Enron email interaction network[2].

[1] `http://reality.media.mit.edu/`

[2] `http://bailando.sims.berkeley.edu/enron_email.html`

4.4 Link Analysis using PageRank

We propose to determine the importance of users by using link analysis techniques. Let us first review the PageRank model and an iterative algorithm to compute PageRank scores. Then, we discuss the application of PageRank in human interaction networks.

4.4.1 PageRank Primer

Using PageRank, u's importance is influenced by nodes $v \in \texttt{inlinks}(u)$ that are in some manner linked to u. For example, in the Web graph, a link is a hyperlink between Web pages, whereas in social networks a link might represent a connection in the network. As an example, the *knows* property in a FOAF[3] profile can be used to create links between people.

The idea of PageRank is best explained in its original context. A user browsing the Web (the "Random Surfer") navigates through a set of Web pages by selecting one of the links on a given page. In other words, the random surfer follows a link with probability α, usually a value between 0.8 - 0.9 according to Page et al. (1998), or with probability $(1-\alpha)$ teleports to a randomly selected Web page. PageRank in directed graphs is defined as follows:

$$PR(u) = \alpha \sum_{v \in \texttt{inlinks}(u)} \frac{PR(v)}{\texttt{outdegree}(v)} + (1-\alpha)p(u) \tag{4.3}$$

Symbol	Meaning
α	The predefined PageRank damping factor (usually a value between 0.8 and 0.9).
$\vec{PR}$	The PageRank vector for parameter α.
$\vec{p}$	The teleportation distribution vector called *personalization vector*.

Table 4.2: PageRank and related symbols.

The vector $\vec{PR}$ can be calculated using an iterative algorithm, for example, the Jacobi iteration as demonstrated in Algorithm 4.1. A simple measure to test whether the algorithm has converged — $\vec{PR}^{(k)}$ holding the importance scores obtained in iteration k — is to look at the error ϵ so that

$$max||\vec{PR}^{(k)} - \vec{PR}^{(k-1)}|| \leq \epsilon$$

However, in practice, a fixed number of iterations is used and measures such as Kendall's τ (a rank correlation coefficient, e.g., see (Berkhin 2005)) to determine whether the ranking scores obtained in iteration k and $(k-1)$ will change the position of u within $\vec{PR}$.

[3] `http://xmlns.com/foaf/spec/`

Algorithm 4.1 Iterative method to compute PageRank scores in interaction graphs.

input: A human interaction graph.
input: Convergence criteria: error ϵ smaller than desired precision.
output: Ranking scores available in vector $\vec{PR}$.
for each user $u \in \mathcal{U}$ **do**
  $PR(u) = (1 - \alpha)p(u)$
end for
while not converged **do**
  for each user $u \in \mathcal{U}$ **do**
    for each user $v \in$ `inlinks`(u) **do**
      $w \leftarrow getEdgeWeight(v, u)$
      $PR(u) \leftarrow wPR(v)$
    end for
    $PR(u) \leftarrow \alpha PR(u) + (1 - \alpha)p(u)$
  end for
end while

4.4.2 PageRank in Human Interaction Networks

There are different views on how we can recast PageRank for expertise analysis in human interaction networks:

1. The first view is closely related to the random surfer model. Discussion forums, for example, are popular platforms when users require help or advise. Typically, the user — or *expert seeker* — creates a description of the problem by posting a question in the forum. Other users reply to the question with either a) a description of a potential solution of the given problem or b) a reference to an existing solution description (e.g., recommending existing postings in the same or some other forum). Hence, the expert seeker decides to navigate through the set of postings or may choose a random posting until he finds a helpful solution or aborts his endeavor. Indeed, the very idea of PageRank is to use citation links as sources for reputation.

2. On the other hand, we can interpret each link between people as a channel to propagate information in a network (Conyon and Muldoon 2006). The strength of a link limits the flow of information between v and u. For example, v may notify one of its neighbors u about some news or forward information to a randomly chosen person.

We hypothesize that PageRank is a suitable model for user-importance ranking. To test this proposal, we calculate PageRank vectors $\vec{PR}_{\Delta t}$ of all users in a time window Δt. Before doing so, we also provide the characteristics of the mobile phone dataset over time, starting with Figure 4.5 (a) showing the fraction of users active in a given month (period 2004 - 2005). Each link in the interaction graph is established based on phone calls.

The ranking results, given the model in Equation 4.3, are obtained using Algorithm 4.1 using the following parameters

- Uniform teleportation distribution: $p(u) = 1$
- Degree based link weights: $w_{v,u} = 1/outdegree(v)$

In Figure 4.5 (b) we show the difference between the set of users in two consecutive months as the Jaccard index, which is often used to measure the distance of two sets. It is defined as $\texttt{jaccard}(U_1, U_2) = |U_1 \backslash U_2|/|U_1 \cup U_2|$. The Jaccard index is a useful (set) metric because a high index indicates that many changes happened in the network; in terms of joining or leaving users. In such cases, we expect $\vec{PR}_{t_1}$ and $\vec{PR}_{t_2}$ to be less correlated. (The index of the interaction network is shown in Figure 4.5 (b)).

In the same figure, we show the correlation coefficient of PageRank vectors in two consecutive months. We calculate the Pearson correlation coefficient, which is defined within the interval $[-1, 1]$, and perform a simple mapping of the interval $[0, 1]$ so we can show both coefficients on the same scale. We see in Figure 4.5 (b) that the vectors $\vec{PR}_{t_1}$ and $\vec{PR}_{t_2}$ are highly correlated.

In period 8 (February 2005) we see a spike in the interaction network's Jaccard index, resulting in lower correlation of ranking scores. As mentioned before, this is an expected observation since many joining or leaving users introduce a new collaboration (interaction) setting. For example, "important" people joining the observable network. Overall, we believe that the PageRank model — applied to human interaction networks — captures well the importance of users since ranking scores are in general highly correlated in a relatively stable interaction network.

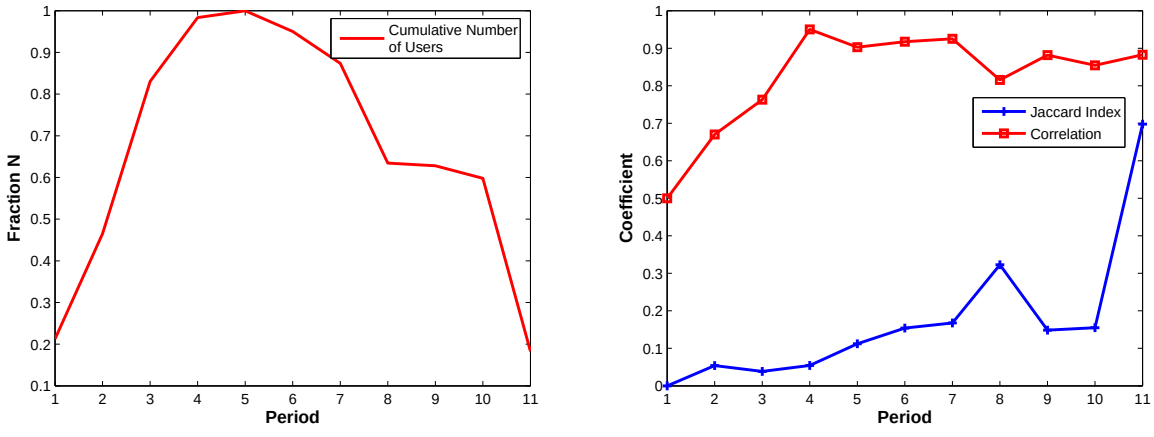


Figure 4.5: (a) The cumulative number of users over time in mobile phone network. (b) Jaccard distance index of network and correlation of PageRank scores over the entire period 2004 - 2005 using a 1 month time-window to update ranking scores. We index each month starting 2004-07 as period 1 and ending at 2005-05 denoted as period 11.

4.5 Towards DSARank

PageRank has been hugely popular in ranking Web pages on the Internet. To date, the company Google provides one of the most successful search engines. Over the past years intense research has been done to better understand the PageRank model and optimizations in computing the rank of Web pages in a large graph of billions of pages. Berkhin (2005), for example, provided a comprehensive survey on PageRank.

However, while we believe that PageRank fits also well to determine the *importance* of users in human interaction networks, our goal of ranking users is motivated by a concrete application scenario (that is, HPS). Specifically, our aim is to devise a ranking model that is able to recommend users who can perform tasks or, more generally, participants in collaborations.

In human interaction networks, we must consider the workload of users in terms of the set of incoming tasks or requests, and also the *intensity* and importance of interactions. Therefore, the goal of our work is different when compared to social network analysis, which attempts to understand the structure of human interaction networks and the role of human actors. For example, in social network analysis, one objective is to determine the *prestige* of users. In this work, however, importance of a user also means to find the most relevant user or expert who can assist in solving a problem.

While in the Web graph hyperlinks usually represent a binary choice, page u is connected to page v, if there is a hyperlink between them, in human interaction networks we can measure the importance of links by considering the intensity of interactions. Also, additional metadata associated with links such as *tags* help us to determine the context of an interaction. We argue that a ranking model with the objective of finding collaborators based on expertise must account for the network *dynamics* in interactions. Hence, we propose a model that regards those users who actively contribute to interactions in a specific collaboration network as important peers. Our model has the following ingredients:

- **Collaboration dynamics.** We extend the PageRank model by introducing various human-centered metrics to account for the dynamics in human collaboration networks. Such metrics include *availability* and *activity level* of a user. Our model attempts to balance between the importance and the activity level (*interaction intensity*) of users.

- **Skill, activity, and action aware recommendations.** Here we go a step further and propose additional parameters for personalized ranking. Interactions in collaborations are always performed in a certain context — that is, in the scope of a certain activity. Recall from previous discussions, we have established the following concepts:

  - An *activity* describes in which kind of work users are engaged. To give examples, a scientist may declare his/her field of research as activities such as “graph algorithms” or “Web service discovery”.

- *Actions* are usually performed in the scope of certain activities, for example, creating documents or research papers in a certain research domain.
- In this work, the notion of human *skills* is equivalent to expertise of a user and can be declared by the user as activities. However, the actual skill level is obtained based on observations of actions (e.g., interaction logs) serving as evidence.

  As an example, the expertise (skill level) of a scientist in a given field depends on the output in form of published paper — actions in a given research activity.

In the remainder of this work, we refer to our approach — accounting for the above mentioned properties — as *DSARank*. DSARank stands for *Dynamic Skill- and Activity-aware* PageRank. Before introducing DSARank, we define a set of human-centered metrics capturing the dynamic nature of human collaboration.

4.5.1 Intensity Metrics

In this section we define availability and intensity metrics to measure dynamics in human interaction networks. We introduce these metrics in the context of point-to-point human communications (e.g., phone conversations), but we can apply similar metrics to measure dynamics in, for example, messaging-based interactions.

Definition 4.5.1 (Availability) *Let us define a user's availability*[4] *as the entire duration user u interacts with other users, i.e., $\{v|v \in \mathcal{U}, v \in A(u)\}$, $A(u)$ defining the set of those users adjacent to u. We define t_{call} as the duration of a specific phone call. Here, $call \in (u, v)$ is commutative (a call can be an incoming or outgoing call). Given the captured telephone logs, we calculate u's estimated availability as*

$$\texttt{availability}(u) = \sum_{v \in A(u)} \sum_{call \in (u,v)} t_{call}(u, v) \tag{4.4}$$

Definition 4.5.2 (Link Intensity) *Let us define the intensity of an interaction link ℓ as*

$$i(\ell) = \left[\prod_{call \in \ell} t_{call} \right]^{1/|\ell|} + \kappa \tag{4.5}$$

The single link intensity is the geometric mean of $|\ell|$ calls plus a small smoothing factor κ defined as

$$\frac{\text{Number of missed calls}}{\text{Total number of calls} \in \ell \text{ with } t_{call} > 0} \tag{4.6}$$

[4] Availability is defined based on already captured interactions.

In general, the geometric mean is used when values in a set of numbers influence each other. For example, when growth rate or relative change in a series of numbers is analyzed. In other words, the single link intensity is the *average product* of a set of calls between people to exchange information. Every information flow between people, perhaps gossip or work-related news, depends on a set of calls that mutually influence each other. (At the moment, we do not consider the context of interactions.)

Definition 4.5.3 (Interaction Intensity) *For a specific user, we define interaction intensity as follows*

$$i(\ell;u) = i(\ell) * |\ell| \left[\sum_{\ell \in \texttt{links}(u)} i(\ell) \right]^{-1} \tag{4.7}$$

The set $\texttt{links}(u)$ contains directed interaction links. For out intensities i_{out}, we demand links to be outgoing links, and similarly for in intensities i_{in}, links must be incoming links.

Definition 4.5.4 (Interaction Intensity Level) *Based on the definition of $i(\ell)$ and $i(\ell;u)$ we define the interaction intensity level IIL as*

$$IIL(u) = \left[\beta^2 \left(\sum_{\ell \in \texttt{outlinks}(u)} i_{out}(\ell;u) \right)^2 + (2-\beta)^2 \left(\sum_{\ell \in \texttt{inlinks}(u)} i_{in}(\ell;u) \right)^2 \right]^{(1/2)} \tag{4.8}$$

The factor $\beta \in [0,2]$ allows IIL to be biased towards i_{in} or i_{out}, where 1 means no bias, i.e., equal importance for in-/out intensities. Biasing IIL is only valid for *all* users.

Definition 4.5.5 (IIL Imbalance) *We define the imbalance $imb(IIL) \in [-1,1]$ as*

$$imb(IIL) = \begin{cases} \dfrac{\sum_{\ell \in \texttt{inlinks}(u)} i_{in}(\ell;u) - \sum_{\ell \in \texttt{outlinks}(u)} i_{out}(\ell;u)}{\sum_{\ell \in \texttt{links}(u)} i(\ell;u)} & , \textit{ if } \sum_{\ell \in \texttt{links}(u)} i(\ell;u) > 0 \\ \infty & , \textit{ otherwise} \end{cases} \tag{4.9}$$

Specifically in messaging-oriented communications, $imb(IIL)$ is a useful measure to determine extreme values:

- *Passive* involvement of users in interactions such as observers yields $imb(IIL) = 1$
- *Active* involvement, but in the extreme case, all interactions could be outgoing and none of the interactions is replied by the other users (no incoming interactions), thus $imb(IIL) = -1$

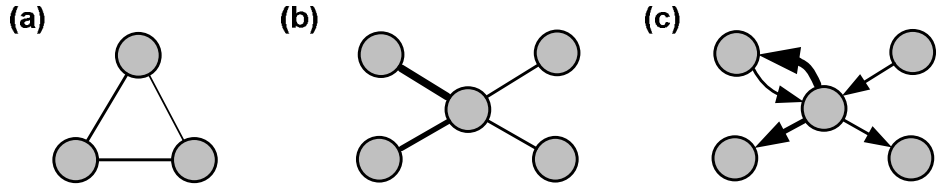


Figure 4.6: Schematic illustration intensity metrics: (a) depicts a subgraph in the network (triangle), (b) undirected, weighted links of nodes in the network, and (c) directed links between nodes.

Here we give some background related to our intensity-based approach. *Motifs* are "small" subgraphs that appear a number of times in complex networks. These simple building blocks (Milo et al. 2002) can be found in biological, technological, and sociological networks (Milo et al. 2004) and can be of very different sizes and connectivities. For example, the authors in (Adamic et al. 2008, Yang et al. 2008) analyzed motifs in online communities (question/answer forums).
Onnela et al. (2005) proposed motif detection in weighted complex networks considering motif intensity and coherence. Figure 4.6 (a) shows a simple subgraph (triangle) and its paths. Each link in the path may have different weights. In (Onnela et al. 2005), motif intensities were studied to detect the *strength of interactions*. Furthermore, in (Onnela et al. 2007), the authors studied the homogeneity of weights in subgraphs, which they defined as *subgraph coherence*. To study coherence of *weights*, the ratio of the geometric to the arithmetic mean can be defined.
We also use the geometric mean to calculate the strength of interactions. As mentioned in (Onnela et al. 2005), it is important to consider the coupling between network structure and interaction strength. However, in our work, even an *attempt* to interact with some other actor in the network (e.g., a user attempted but failed to contact another users) is appended as smoothing factor κ to the link intensity $i(\ell)$. This is done because certain communication means (e.g., the physical network) are unreliable in terms of establishing the communication channel between actors. We calculate the strength of interactions as undirected links, Fig. 4.6 (b), and calculate the *homogeneity of interaction strengths* (see interaction intensity $i(\ell; u)$). For example, to calculate if the strength of a specific link is much higher, or lower, than the average strength. In particular, we use the ratio of $i(\ell)$ — the strength of a specific link — an the arithmetic mean of the strength of all links $\ell \in \texttt{links}(u)$. Finally, we create the directed, weighted graph Fig. 4.6 (c).

4.5.2 Intensity-based DSARank

In this section we introduce DSARank, a model to capture the dynamic nature of collaborations. More precisely, in the DSARank model we assume that expert seekers interact with *well-informed* users depending on the intensity of interactions (that is, IIL-dependent)

and also with users that are typically highly available. We believe that DSARank is better suited to recommend users (experts) in human collaboration. Our model does not only rely on the structure of the network, for example $\texttt{indegree}(u)$ and $\texttt{outdegree}(u)$ of a particular user u, but also captures the dynamics in collaborations. Also, it is important to consider the *context* of interactions. Context allows us to refine rankings of users based on skill information and expected expertise level of users. DSARank has the following important properties:

Non-uniform personalization vectors. Previously, we used $p(u) = 1$. However, a user does not randomly establish interactions (or collaborations) with other users. In particular, the expert seeker chooses to request an opinion or input from users who are potentially those people influencing or controlling the flow of information.

User preferences. Typically, there is a trade-off between various personalization metrics. For example, interaction metrics include availability and interaction intensity. We should be able to favor one metric over the other. We can then decide which of those metrics should mainly influence rankings, for example, of the set of users recommended for collaboration.

Symbol	Meaning
m	A metric or measured value in human interaction networks. For example, a metric depicts availability or intensity of interaction links. Metrics can be obtained by observing the interaction in a specific context, or the (whole) interaction network in general. The set $M_R = \{m_1, m_2, \ldots, m_n\}$ defines the various interaction-based ranking metrics.
w_m	Denotes the weight of a metric. For example, one may define preferences for different metrics. If weights are not explicitly specified, we regard each metric to be equally important and use the following definition of weights: $W_{M_R} = \{w \mid w_m = 1/\lvert M_R\rvert, \forall m \in M_R\}$. Also, the sum of metric weights must be equal to 1.

Table 4.3: Metrics and weights in interaction networks.

As mentioned earlier, I is a column-stochastic matrix. Thus, the sum of outgoing edge weights of a particular node v must be equal to 1. The sum of weights is given as

$$ws_v = \sum_{z \in \texttt{outlinks}(v)} w_{v,z} \tag{4.10}$$

Next, we define the formula for the basic DSARank

$$DSA(u) = \alpha \sum_{v \in \texttt{inlinks}(u)} \left(\frac{w_{v,u}}{ws_v} \right) DSA(v) + (1 - \alpha) \sum_{w_m \in W_{M_R}} w_m p_m(u) \tag{4.11}$$

with $||\vec{p}|| = 1$

Remark 4.5.6 (Normalization) *The personalization vector $\vec{p}$ has to represent a valid probability distribution (i.e., $||\vec{p}|| = 1$). Therefore, we need to normalize availability and IIL. Let $p_1(u)$ denote the personalization for availability and $p_2(u)$ for IIL:*

$$p_1(u) = \frac{\texttt{availability}(u)}{\sum_{v \in \mathcal{U}} \texttt{availability}(v)} \qquad p_2(u) = \frac{IIL(u)}{\sum_{v \in \mathcal{U}} IIL(v)} \tag{4.12}$$

In the following, we go a step further and introduce personalization not only based on availability and IIL, but also additional skill-metrics to rank users based on *interaction contexts*.

4.6 Context-Aware DSARank

The next step is to introduce our approach to determine the most relevant user (expert) based on interaction contexts. In Fig. 4.7, we show the three essential steps: (1) capturing activity-/action information, and metadata associated with interactions, (2) perform ranking in subgraph partitions, and (3) aggregation of rankings based on the expert seeker's preferences.

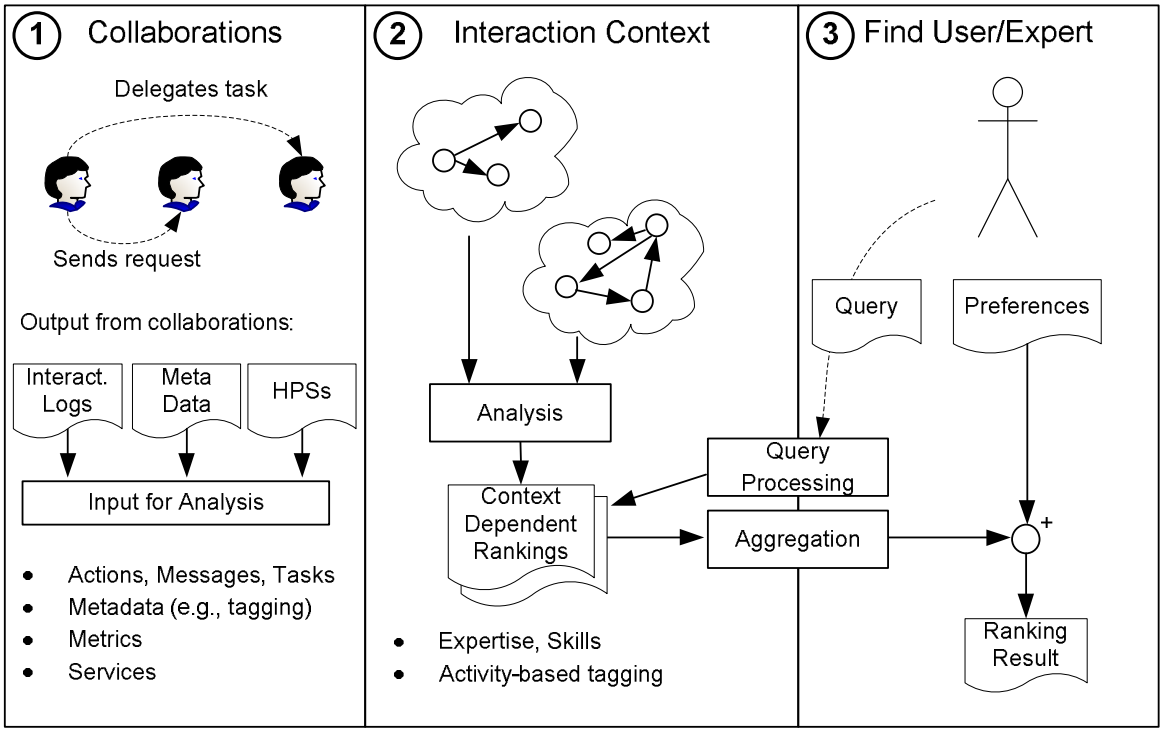


Figure 4.7: Schematic illustration of context-aware DSARank.

Interactions between two users may have different scopes. Therefore, we determine a user's expertise by considering context because interactions take place at different levels of importance (i.e., the weight of a link) and intensities. However, there might be a "coupling" between contexts because a link may convey information relevant for multiple context topics.

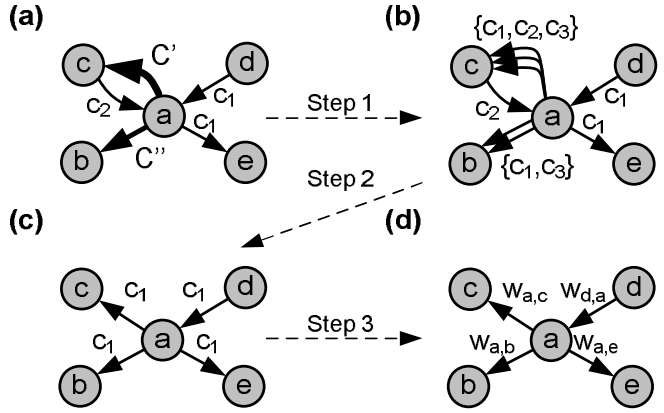
- **Step 1.** Capturing interactions in human collaboration networks and associated metadata including tags and activity information, which can be provided by users, to define the *interaction context*. As discussed in Sec. 4.5.1, we derive metrics to calculate the dynamics and user involvement given the entire flow of interactions in the network.
- **Step 2.** Perform ranking based on the interaction context by decomposing the interaction network into subgraphs. Decomposing the network can be performed using applied tags, user profiles, or content analysis of messages (e.g., emails, forums, discussion groups, etc.). The subnetworks (i.e., graphs) serve as input to calculate `Context Dependent Rankings`. This step is computed offline.
- **Step 3.** The expert seeker formulates a query, for example, specifying a set of skills or keywords, which is evaluated by a `Query Processing` module. The next step is to utilize the precomputed rankings to perform `Aggregation` of ranking scores of those users matching the query. This is done by taking the expert seeker's `Preferences` into account.

4.6.1 Interaction Context

We discuss context in a general setting and define *context tags* as any information that can be used to determine the interaction context.

Definition 4.6.1 (Interaction Context) *We define the context $C = \{c_1, c_2, \ldots, c_n\}$ of an interaction as the set of context tags. An interaction takes place in a certain context $C' \subseteq C$, if a link ℓ is annotated with $c \in C'$. The context between $u, v \in \mathcal{U}$ is determined by the scope of a link $\ell(C')$. For users in a particular context c, we can create a subgraph g based on subset of links $\{\ell(c)\}$. The set of users $U(c) \subseteq \mathcal{U}$ interact in a context c.*

As an example for context tags, users tag email messages if these messages are related to a certain project or activity. We assume that different types of context tags are applied to interaction links with a certain frequency c_f. To account for misplaced or missing tags, we perform additive smoothing which is a simple but effective method to calculate probabilities of tags. Usually a smoothing factor γ ranging from $0 < \gamma < 1$ is used. The smoothened tag occurrence probability is $P(c_f; \gamma) = (c_f + \gamma)/(\sum_{c \in C'} c_f + \gamma)$, where C' denotes all context tags used in a link ℓ between u and v.



$$W^T = \begin{bmatrix} 0 & 0 & 0 & w_{d,a} & 0 \\ w_{a,b} & 0 & 0 & 0 & 0 \\ w_{a,c} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ w_{a,e} & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.8: Example of tagged graph and corresponding (weighted) subgraph.

In Fig. 4.8 we show (a) an example interaction graph with tagged links, (b) the graph with decomposed links based on tags, (c) the context-dependent subgraph (for example c_1), and (d) the weighted subgraph. Also, we show the weighted transpose matrix W^T of the subgraph. As shown in Fig. 4.8 (a) and (b), each interaction link ℓ is tagged with a certain frequency c_f. We interpret context tags applied to interaction links as follows: User v interacts with user u in different contexts C'. However, some tags appear with higher frequency than other tags. Thus, we assume that v regards some interactions with u as important in a specific context — the relevance of a link in a context c is proportional to the tag's frequency. Therefore, we use smoothened tag frequencies as weights to depict the relevance of an interaction link ℓ for c. For example, the chance of receiving an item of information or that u will be contacted in a specific context c.

Given the set of context dependent links $\{\ell c\} \in g$, we estimate the expertise of users interacting in c as

$$SE(u;c) = \sum_{v \in \texttt{inlinks}(u)} \left(\frac{w_{v,u}}{ws_v} \right) SE(v;c) \tag{4.13}$$

Compared to $PR(u)$, $SE(u)$ is computed in a similar manner; however, in $SE(u)$ we do not use personalization vectors (i.e., $\vec{p}$) because $SE(u)$ will be used in a context-dependent subgraph g, which is already itself a personalization. Notice, depending on the interaction network, $SE(u)$ can also be interpreted as the *expected informedness* of a user.

Definition 4.6.2 (Context-Sensitive IIL) *Previously, we discussed IIL without considering interaction context. However, for many collaboration and ranking scenarios, it is important to find users that are highly involved in specific projects, activities — the context in general — because a user's expertise and expected informedness or not equal for all contexts in which the user is participating. For a given context c, we calculate $IIL(u;c)$ based on the set of context dependent links $\{\ell(c)\}$ in g.*

4.6.2 Context-Sensitive DSARank

As mentioned before, $SE(u)$ is the context-dependent skill- and expertise level of a user. We could use $SE(u)$ as a weighted preference in the ranking process by including $SE(u)$ when computing DSARank as defined in Equation 4.11. However, this is an impractical solution because we cannot precompute the vector $\vec{DSA}$ for every possible combination of demanded skills (i.e., depending on the expert seekers preferences). On the other hand, we must consider the *context* of interactions to recommend the right expert. The following equality helps to solve this problem:

Theorem 4.6.3 (Linearity) *(Haveliwala 2002)(Jeh and Widom 2003) For any personalization vectors $\vec{p}_1, \vec{p}_2$ and weights w_1, w_2 with $w_1 + w_2 = 1$, the following equality holds:*

$$\vec{PR}(w_1\vec{p}_1 + w_2\vec{p}_2) = w_1\vec{PR}(\vec{p}_1) + w_2\vec{PR}(\vec{p}_2) \tag{4.14}$$

The above equality states that personalized PageRank vectors $\vec{PR}(\sum_{w_m \in W_{M_R}} \vec{p_m})$ can be composed as the weighted sum of PageRank vectors. Thus, we can restate the definition of DSARank, w_c depicting the weight for a particular context c, as

$$DSA(u; C') = \sum_{c \in C'} w_c DSA \left[\sum_{w_m \in W_{M_R}} w_m p_m(u) \right] \tag{4.15}$$

It is clear that not all users participate in context C'. For a specific context c, we set

$$p(u) \equiv \begin{cases} p_m(u) & \text{, if } u \in U(c) \\ 0 & \text{, otherwise} \end{cases} \tag{4.16}$$

4.6.3 Summary of Ranking Model

In this section we proposed our context-aware ranking model termed DSARank. DSARank is a link-analysis algorithm which is (i) computed offline in context-dependent *subgraph partitions* and (ii) used online to compose personalized ranking results based on the expert seeker's preferences, for example, the demanded set of skills. Table 4.4 provides a brief summary of fundamental concepts and symbols.

Symbol	Meaning
$w_{v,u}$	Weight of a link connecting v to u.
ws_v	The sum of outgoing link weights of node v.
IIL	Interaction intensity level.
c	Denotes a context tag.
c_f	The frequency of a context tag.
γ	A factor in the range $0 < \gamma < 1$ used in additive smoothing; to smooth a distribution representing context tag occurrence probabilities.
$SE(u; c)$	Subgraph, or context dependent, skill- and expertise based importance "particle" — the smallest constituent part of an assembly of SE-based personalization. For example, an expert seeker's query may specify the demanded set of skills.
$\vec{p_m}$	The teleportation distribution vector for metric m.
$DSA(u; C')$	Context-aware DSARank which is customized based on the set C' of context tags.
w_c	Weight for context-dependent DSARank.
$i(g)$	Subgraph intensity.

Table 4.4: DSARank and related symbols.

4.7 Advanced Context-based Metrics

In this section we focus on advanced metrics to measure subgraph properties and context-dependent link characteristics. Subgraph properties are useful when we want to compare, for example, the intensities of user interactions in different contexts. Here we will define subgraph intensity $i(g)$ to analyze in which context users interact with the highest intensity.

In addition to subgraph properties, we will define context-dependent link metrics. These metrics are mainly used to filter and visualize properties of interaction graphs; however, they can also be used in more general ranking functions. In many cases interaction graphs comprise a large number of users. Thus, users and their interactions (links between users) need to be filtered; especially in visualizations of large graphs. Metrics such as link *pointedness*, *coverage*, and *affinity* help us to see the relationship between people (from a link analysis point of view) and importance of interactions in a filtered set of top-ranked users.

For context-dependent IIL, we calculate the interaction intensity of all users in g as subgraph intensity using the following definition:

Definition 4.7.1 (Subgraph Intensity) *Let us define the subgraph intensity $i(g)$ as*

$$i(g) = \frac{1}{|U(c)||C|} \sum_{c \in C} \sum_{u \in U(c)} IIL(u; c) \tag{4.17}$$

In one-to-many communications (e.g., email), we may have many recipients in a single interaction. Thus, interactions in messaging-oriented systems result in attaching one particular message to multiple interaction links. The following metrics are defined within a range of $[0, 1]$.

Definition 4.7.2 (Link Coverage) *Given the link ℓ, we define link coverage as a metric for the total recipient* `degree` *of messages msg attached to ℓ:*

$$\texttt{coverage}(\ell; u) = \left[\sum_{msg \in \ell} \frac{1}{\texttt{degree}(msg)} \right]^{-1} \tag{4.18}$$

In other words, `coverage` is a metric describing the total number of people who received a particular message in context C'. Notice, each message may have multiple context tags attached to it. If `coverage` is the "dispersion" of a link, then $\texttt{pointedness}(\ell; u) = \texttt{coverage}(\ell)^{-1}$ indicates whether messages in a given link address rather a few individuals or many people (i.e., the recipient `degree`).

Definition 4.7.3 (Link Affinity) *For a link between (v, u), we define link affinity as follows*

$$\texttt{affinity}(v, u; \ell) = 1 - \sum_{msg \in \ell} \begin{cases} \frac{\texttt{degree}(msg)-1}{\texttt{degree}(msg)} & , \; u \notin \texttt{recipients}(msg) \\ 0 & , \; otherwise \end{cases} \tag{4.19}$$

Definition 4.7.4 (Interaction Coverage) *Is defined in an interaction context as the relative number of people addressed by, or involved in interactions initiated by, a given user v. Those who are active in the same context are defined by the set $U_a(C')$ and users with passive roles in interactions by $U_p(C')$.*

$$IC(v; C') = \frac{|\{\ell(v; C'\}|}{|U_a(C')|} \tag{4.20}$$

Definition 4.7.5 (Interaction Precision) *We define precision as the relative number of users addressed by v, given the number of users involved in interactions, which have never been active in C'.*

$$IP(v; C') = 1 - \begin{cases} 0 & , \; if \; \{u \in \texttt{outlinks}(v)\} \cap U_p(C') = \emptyset \\ 1 & , \; if \; IC(v; C') = 0 \\ \frac{|\{u \in \texttt{outlinks}(v)\} \cap U_p(C')|}{|\{u \in \texttt{outlinks}(v)\}|} & , \; otherwise \end{cases} \tag{4.21}$$

Definition 4.7.6 (Context PC) *The ratio of precision and interaction coverage is given as*

$$PC(v; C') = \begin{cases} IP/IC & , \textit{ if } IC \neq 0 \\ 0 & , \textit{ otherwise} \end{cases} \tag{4.22}$$

PC is a metric indicating whether v interacts with the "right" people in C' and how many of those users which are potentially interested are included in conversations in a specific context.

Chapter 5

HPS Framework

5.1 Abstract

We envision collaboration scenarios where people define services based on their skills and expertise. The expert seeker (requester) can interact with experts by using HPS. To enable human participation through HPS, users must be able to utilize tools to design and model their participations. These tools must be simple yet powerful enough to deal with complexities in the service-design process. To date, most effort has been spent on tools for Web service professionals and developers which, however, cannot be used by users without programming skills. We present methods and tools supporting the user in the design of HPSs in SOA-based environments. We analyze the complexity and challenges of the design process and present our solution. In the design of HPS, we focus on two main aspects of human interactions in SOA: (i) an approach for designing service interfaces embodying human activities as actions offered by Web (HPS) users; (ii) a tagging model for activities and services to recommend (HPS interfaces) in the design process. We discuss the mapping of human activities onto Web services. We introduce a framework supporting HPSs in different types of interactions. We present the architecture and its core components: the *Middleware Layer* providing features for managing data collections and XML artifacts, *API Layer* comprising services for forms generation and XSD transformations, the *Runtime* enabling basic activity and user management features as well as support for human and service interactions using Web services.

In open and dynamic collaboration environments on the Web, it is important to define metrics capturing the characteristics of HPS. Such metrics need to be established based on *human tasks*, *interactions*, and *expertise* level of users. In this chapter, we present a task rewarding model, which rewards users based on performance indicators such as reliability and processing time of tasks. The rewarding model also accounts for global task properties including risk and trend of processing time in a set of related tasks.

5.2 HPS Revisited

This chapter provides an integrated view on previously introduced concepts and the technical realization of HPS interaction models. It is useful to revisit some of the most fundamental concepts — as introduced in Chap. 3 — before we elaborate on the technical architecture.

Human interactions using HPSs. The novelty of the HPS concept is that *human capabilities* are *modeled and enacted using Web services* technology. In other words, in HPS Web services are not used to implement certain (supporting) applications so that humans can deal with, for example, human tasks; instead, human capabilities are "exported" as Web service interfaces based on human activities (i.e., declarations of activities). Moreover, the notion of activity-centric collaboration ties into the compositional nature of the Web by providing the fundamental features to structure and compose activities in a hierarchical manner.

Human interactions as part of (software) service compositions. Many business processes require human input as part of their regular execution. In other cases, human input may only be required if the process cannot continue its execution due to exceptions that need to be analyzed by humans. B4P, for example, targets composition models of BPEL-based flows and human interactions — mainly in the context of business processes. When the BPEL engine reaches an (people) activity, a `People Query` is used to retrieve a set of people, which are selected from a registry or directory, to work on tasks. The result of those tasks is passed back to the BPEL engine and the process continues its execution.

In HPS, we target the *user-driven* design and interactions with *user-based* services. In business processes we typically require a detailed definition of tasks, notifications, and roles in advance. This is done by a business analyst or the process designer.

Topic	Introduced in	Feature in HPS Framework
Activity Use Cases	Sec. 3.3.1	Supporting the user- and community driven approach to design HPSs.
Conceptual HPS Model	Sec. 3.4.2	Collections of SOA artifacts that can be accessed through various means.
Task Model	Sec. 3.4.3	Task registry enabling the creation of (public) task announcements and human tasks to control the status of interactions.
Calling an HPS	Sec. 3.4.4	Dispatching and routing of interactions (that is, HPS requests and responses) using the HPS Access Layer.
DSARank	Sec. 4.6	All interactions are monitored and logged. Importance rankings are used in the design process and in the lookup of HPSs.

Table 5.1: Mapping between concepts and framework.

5.3 Outline of Approach

The first step in this section is to define our approach enabling HPSs — illustrated in Fig. 5.1. There are three essential steps: 1) *ability to define services*, 2) *user-centric service publishing and provisioning*, and 3) *discovery of HPSs* as well as *interactions with humans using Web services technology*. Our goal is to define a framework that integrates Web technologies and Web services enabling humans to publish services.

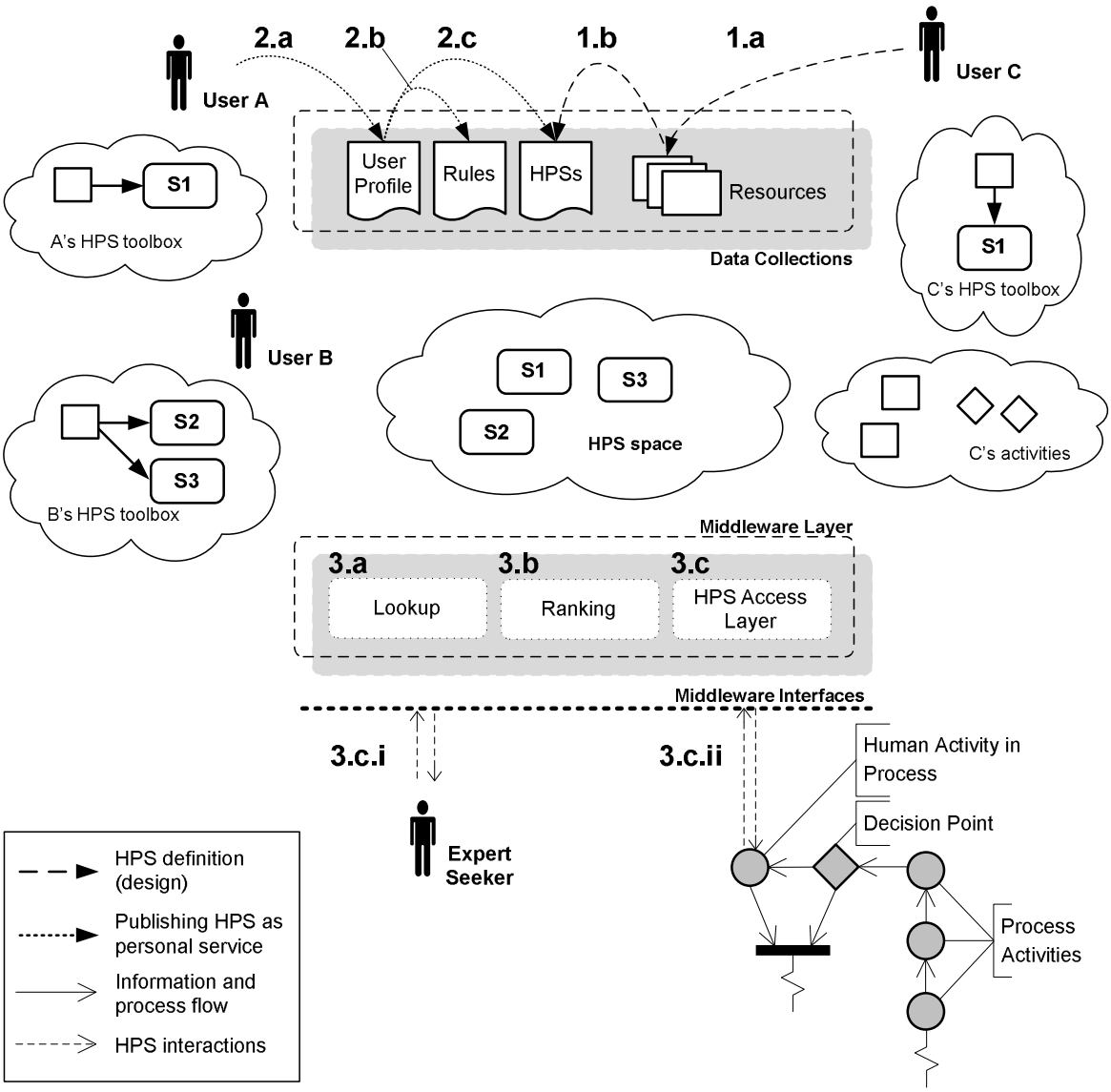


Figure 5.1: Overview and motivation of HPS framework. The framework enables a user-centric approach for the design and provisioning of HPSs.

1. *Ability to define services.* People should be able to define services and corresponding interaction interfaces. This can be done by reusing and/or modifying existing interfaces and resources; or by creating new interfaces. In Fig. 5.1, User C performs two essential steps:

   (a) User C uses his HPS toolbox to create new *type definitions*, for example, activity types and/or parts of complex data types. These definitions are stored in a collection of resources.

   (b) Based on complex activity structures, User C creates an HPS interface that is also stored in a collection of resources.

2. *User-centric service publishing and provisioning.*

   (a) In the first step, users specify their personal information as profiles. The structure and semantics of user profiles has been addressed in various standards and specifications, for example FOAF or vCard[1]. Similar profile information is used in HPS (in this work we will not focus on the details), but with some extended information to capture, for example, user competencies.

   (The next two steps are not in a particular order.)

   (b) The user can specify interaction rules to automatically route or filter requests. Notice, we use document-centric (Web service) interaction styles. Such XML-based documents are typically encapsulated in SOAP envelopes and exchanged as XML messages. In HPS, these messages represent *actions* and *resources* in collaboration that are associated with human activities or artifacts. Therefore, other users can be engaged in interactions by delegating requests or tasks to them. These rules are usually specified for specific HPSs.

   (c) User A creates a *personal service* to manage interactions in a given context; that is, in collaborations using HPS. This step basically associates a user profile with an HPS interface (i.e., the group of people providing specific services — previously, we introduced `HPSGroups` to depict these users).

   Then, personal services are deployed — this step is not explicitly shown in Fig. 5.1. We support different deployment options: a) hosted solution where users access their services and manage requests using a Web portal (no software stack is needed on the user's device) and b) deployment of a software stack on end-user devices, for example, mobile phones or PDAs. This software stack has XML parsing capabilities, a user front-end to manage activities and services, and a dynamic rendering engine to display a user-interface (GUI) based on XML documents.

3. *Discovery and interactions.* HPS aims at simplifying interactions with user-provided services by abstracting from service location and deployment. Requesters discover services and interact with the selected HPS through a middleware.

[1] vCard: `http://www.ietf.org/rfc/rfc2426.txt`

(a) Lookup: Requesters discover HPSs (finding HPSs by matching interface descriptions).

(b) Ranking: Services are ranked based on the requesters' preferences (ranking criteria). NFPs[2] are used in the ranking process. See, for example, (Ran 2003) for an introduction to Web services discovery using QoS properties in UDDI (Universal Description Discovery and Integration) registries and (Liu et al. 2004) for QoS computation and policing.

(c) HPS Access Layer: The HPS Access Layer (HAL) dispatches requests and performs security checks. In Fig. 5.1, we show two different HPS interaction scenarios.

   i. Interactions *between humans* using HPS.

   ii. Human input is demanded by a composed (software) service, for example, modeled as a step in a process flow.

We follow a top-down approach in this chapter. First, we discuss the design and (software) engineering perspective of HPS. However, the design of HPS has also a social component. For example, the design or provisioning of HPSs might be influenced by community structure, social interest and competition, and evolution of HPS-based communities. These communities are not "static" environments because users have the ability to contribute services, thereby creating an open marketplace of human-based services.

While we based our related-work discussion (i.e., Chap. 2) on four architectural views — (i) functional, (ii) physical, (iii) technical, and (iv) dynamic operational architecture, we can think of an additional view that is based on the social component that arises when systems of services are designed: (v) the social architecture. A social architecture can be thought of as a mix between user-centric services to engage in different Web-based interaction scenarios, community structure, social interest and evolution.

However, the design of HPS is not enough; we need to build a framework that manages data collections, interactions, and activities. Therefore, the next step is to introduce the architecture of the HPS framework. We detail the *Design* tools, *Runtime* services, *API Layer*, and *Middleware Layer*. In this chapter it is sufficient to show the conceptual framework. The implementation of all framework-based tools and services will be discussed in detail in Chap. 6 (implementation).

5.4 Supporting the Design of HPS

An HPS is exposed as a Web service interface that HPS requesters can use to interact with humans. From the user's point of view, services are represented as *activities* and *actions* the user can perform in SOA-based collaboration environments. To enable human participation

[2]NFP is an abbreviation for non-functional properties.

through HPS, users must be able to utilize tools to design and model their participations. These tools must be simple yet powerful enough to deal with complexities in the service-design process. To date, most effort has been spent on tools for Web service professionals and developers which, however, cannot be used by "novice" users. For example, users who do not have programming skills or expertise in XML-based markup languages.

Mashup editors, for instance Yahoo! Pipes[3], are examples of how simple tools can facilitate user participation and user-driven processes by gathering and aggregating different sources of knowledge. We argue that similar tools should be provided for the design of HPS, enabling users to create their personal services. In this section, we present methods and tools supporting the user in the design of HPS in SOA-based environments. We analyze the complexity and challenges of the design process and present our solution.

Our goal is to provide powerful yet simple tools for users to define and provide services. Such tools should automatically generate all the resources needed to allow users to fully participate in HPS-based interactions in SOA. Here we present an architecture and its implementation allowing humans to design services for various collaborations, with the following key contributions:

1. A methodology to integrate human interactions in SOA based on Web services technology.

2. Method to help users decide which services they should provide. Our proposed method is based on tagging and collaborative filtering of information based on social interest including user profile similarity and personalized expertise-based ranking.

3. Furthermore, we utilize Web services standards such as WSDL to depict HPSs. The HPS design approach enables the automatic transformation of human activities into low level service description. We implemented a set of tools and mappings to enable automatic transformations. Thus, a user is not required to learn (about) Web services standards, XML languages, or programming languages.

5.4.1 The Collaborative Design of HPS

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

Interface transformation and generation: Designing and providing a service should be as simple as writing a "blog entry". Mapping human activities onto Web services

[3]Yahoo! Pipes online at: `http://pipes.yahoo.com/`

is challenging. Users have to be supported in the design in an easy and intuitive manner by hiding underlying complex processes, which require steps such as automatic service interface generation and translation of service interfaces (e.g., WSDL) into GUI representations. Since Web services standards are used — at the technical level — to enable HPS, versatile collaborations can be supported including interactions between humans as well as the use of HPS in, for example, formalized processes.

Recommendations for the design of HPS: We argue that humans should be able to design and provide their capabilities as services. Many HPSs may be available and registered. These services may have different interface characteristics, for example actions and type-definitions, and may be used for very different collaborations. The actual meaning and usefulness of a particular service depends on the user's background, interest, and expertise. However, it is difficult to use ontologies or predefined taxonomies to classify different types of services because HPSs cannot be classified as "bags of services". For example, User A may insist that S1 belongs to a set of categories[4] $C1$ and User C may argue that S1 belongs to the set $C2$, which may or may not overlap to some degree with $C1$. The classification, or association of a service with one or more categories, emerges as users apply tags to services. Eventually, this "tagging process" should converge towards a stable set of defined categories that is agreed upon by most users. This typically happens in tagging systems on the Web. In our opinion it is more useful to apply tags to services for classification as opposed to the definition of rigid taxonomies. As validated in existing research results, for example see (Golder and Huberman 2006) for usage patterns of tagging, tags represent with good accuracy classifications of concepts.

5.4.1.1 Usage Patterns of Tagging in Mixed Systems

We propose tagging mechanism to help users in expressing their skills and the context of interactions in mixed systems. Tagging becomes increasingly important in today's collaborations because people can associate metadata to various artifacts including Web documents, links, messages, and so forth. Thus, people can perform search based on user-defined metadata. Generally speaking, tags are keywords and terms associated with information. Similarly, tags in the HPS framework are used to identify the context in which services and artifacts are used. A tag has different meanings depending on what is tagged and when "it" is tagged. In this work, we distinguish between the following types of tags:

- *Activities:* Activities can be tagged by users to indicate interests (who is interested in what). For example, a user may be interested in activities related to "Mobile Web services", "WSDL specification efforts", or "SOA runtime".
- *Services:* Service tags are applied to HPSs and software services. A user applies a tag to a service to indicate that a particular type of service can be used to perform an

[4]The term category has the same meaning as our previous definition and use of context tags that are depicted by the set C.

activity. Both the consumer and the provider of a service can assign tags. Depending on these roles (consumer or provider) these tags can be quite diverse because how a service is actually used is often different from how the provider intended the service to be used. To give examples of service tags, User A provides a "Review service" and applies tags such as "Review WSDL specification" whereas User B associates even more fine-grained metadata to the specific HPS by applying a tag "Review WSDL 2.0 specification".

- *Actions:* Action (instances) are tagged to denote the use of services from the point on when collaboration commences. For example, in the context of a particular action the tag "Write report". As mentioned earlier, these tags can be used to calculate the *intensities of interactions* in a specific context.

5.4.1.2 HPS Design Use Case

The next step is to illustrate (see Fig. 5.2) how the design is supported by utilizing tagged information and collaborative filtering methods. In Fig. 5.2 (b), dashed line patterns of boxes depict steps in which the user interacts with the system. Solid line patterns depict steps that are performed by the system without human intervention or input.

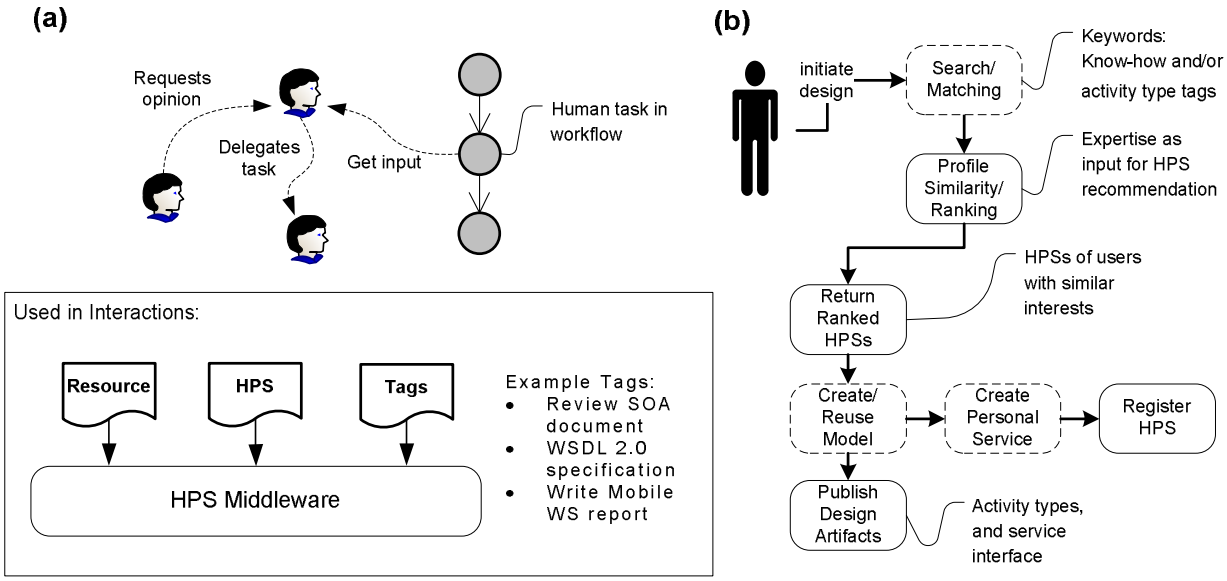


Figure 5.2: (a) Interaction scenario in human and service centric collaborations. Tags are applied to classify activities and services. (b) The design: people can reuse existing models or define new interfaces.

- *Search and matching:* The search for resources is performed by matching the user's query against existing HPSs. A matching function takes service metadata as input, either automatically extracted keywords or tagged information. Hence, tagged information is not only used during collaboration, but also at design time.

- *Similarity and ranking:* The next step is the ranking of HPSs which matched the demanded set of keywords. The output is a ranked list of suggested HPSs by comparing interest-similarity. Similarity is calculated based on the user's (i.e., the person that initiated the design process) activities and profile with interests of those users already providing a certain HPS.

- *Create or reuse model:* The user can create new models, publish related resources and type definitions, or reuse existing HPS definitions. The model defines human activities which are mapped onto Web services.

- *Create personal service:* The final step is to make the definition of the personal service available. (As already mentioned, a personal service is the association of a user's profile with an HPS.) An HPS registry is used to maintain this information.

5.4.1.3 Recommendation Algorithm

In the spirit of collaborative tagging systems, we perform ranking and recommendations based on tagged resources. Before defining an algorithm for recommendations, we need to define several concepts and functions:

- Tagged resources are represented as a triple (`user, resource,` {`tag`}).

- A user $query = \{word_1, word_2, \ldots, word_n\}$ contains a number of search terms that are used to match resources (i.e., an HPS description) assuming that $query \subseteq \{tag\}$ (the set $\{tag\}$ is applied to a particular resource) is satisfied.

- We use the *correlation coefficient* to measure similarity of profiles and activities. Let X and Y denote sets of word frequencies — a word is a string depicting a tag applied to resources such as user profiles, activities, or services; with $|X| = |Y|$. Furthermore, let $\overline{X}$ and $\overline{Y}$ denote the mean (or expected value) of word frequencies in X and Y respectively, N the number of distinct words that are used to calculate correlations and $stdev(\cdot)$ a function calculating the standard deviation of word frequencies. Then, the correlation coefficient is defined as

$$correl(X,Y) = \sum (X - \overline{X})(Y - \overline{Y})/(N * stdev(X)stdev(Y)) \tag{5.1}$$

  with $correl(X,Y) \in [-1,1]$. Based on the coefficient, we use the variable $\phi \in [0,1]$ to have strictly positive values for correlations, calculated as $\phi = \frac{correl(X,Y)+1}{2}$.

- We perform smoothing of tag frequencies by using additive smoothing. This technique is sometimes called Lidstone smoothing. Previously (Sec. 4.6.1 in Chap. 4), we used the same method to smoothen tag frequencies applied to interaction links. Recall, a smoothing factor γ is used, $0 < \gamma < 1$.

- Users can parameterize the ranking algorithm by assigning preferences. The user-defined preference vector is depicted as $pref(u)$, assigning a preference score to user u. The default value is $pref(u) = 1$.

Symbol	Meaning
$w_{(u,tag)}$	Weight of a tag applied to resource.
$\omega(tag)$	The frequency of a tag applied to resource.
ϕ	Variable depicting the correlation coefficient within the range $[0, 1]$.
$pref(u)$	User-define preference vector used in recommendations for HPS design.

Table 5.2: Symbols used in recommendation algorithm.

Algorithm 5.1 shows how to obtain ranking scores. We measure the similarity of the user's profile with activities of matching users that already offer a specific HPS (depicted as resource) and calculate ranking scores as the weighted sum of action tag weights.

Algorithm 5.1 Recommendation algorithm for HPS design.

```
1: input: A user query.
2: output: Recommended HPSs depicted as resources.
3: /* Get matching resources. */
4: R ← match(query)
5: /* Get users that apply resource as HPS. */
6: U ← getUsers(R)
7: for each user u ∈ U do
8:     /* Similarity is calculated based on v's profile and u's activity tags. */
9:     φ = similarity(u, v)
10:    if u ≠ v and φ > 0 then
11:        for each resource ∈ getResourceByUser(u) do
12:            for each tag ∈ resource do
13:                /* Get the frequency for a specific action tag used by u. */
14:                ω(tag) ← getFrequency(u, tag)
15:                /* Assign the weight using the smoothing factor γ. */
16:                w_(u,tag) ← getSmoothenedWeight(ω(tag))
17:                sum ← sum + w_(u,tag) * φ
18:            end for
19:            R(resource) ← R(resource) + sum * pref(u)
20:        end for
21:    end if
22: end for
23: return ranked list of resources
```

5.5 HPS Interface Transformation and Generation

The design process and methodology presented in this work has to be supported by a set of tools and models. We start with the definition of the process, which allows users to define services without having to understand Web services technologies. Several approaches (Kassoff et al. 2003, Song and Lee 2007) focus on automatic GUI generation based on WSDL descriptions. However, these works assume that the WSDL description of a service already exists and simply needs to be parsed and mapped into some GUI language/representation. We propose to create service descriptions *based on* human activity. Instead of mapping existing WSDLs into GUIs, we let users design interfaces that map into *WSDL and GUIs*. This is a challenging problem because neither a process to create mappings between human activity and WSDL nor the automatic generation of WSDL and GUIs — in terms of end-user design support — has been defined or addressed in previous research.

5.5.1 Design Process

We define a process allowing users to create an activity model serving as the input for automatic generation of service artifacts. Figure 5.3 shows the design process. Steps are depicted as box-shaped symbols. Boxes with dashed line patterns denote steps requiring human input. (In this case, only the first step requires human input.) Definitions and models (for example, models describing the mapping of human activities and WSDL elements) are depicted as box-shaped symbols that are slightly tilted. Furthermore, document symbols at the right side of Fig. 5.3 denote the output of the process. Tools and services used within this process generate WSDL descriptions as well as UI representations.

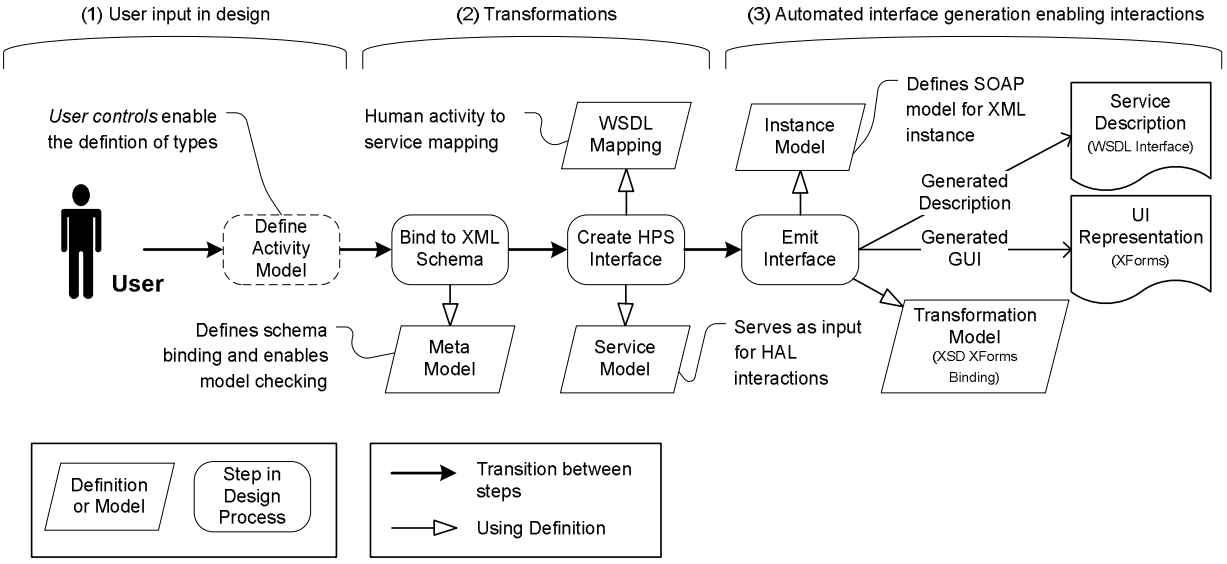


Figure 5.3: Conceptual approach and interface design.

1. **User input in design**

   - *Definite Activity Model:* users define their *Activity Model* (for example, activity types). User controls are simple GUI elements that are hosted in a Web portal provided by the HPS framework. These controls enable users to create complex structures. An example of such a control will be given at a later point.

2. **Transformations**

   - *Bind to XML Schema:* The next step is the automatic transformations of the user's input into XML artifacts. This requires the definition of a *Meta Model* defining the binding of the user's input (created using the control) to XML schema. Constraints and mappings expressed as meta models — defined as XML schemas — will be shown in the implementation chapter. However, detailed knowledge about these meta models is not needed at this point. At the technical level, an `XSD Transformer` is implemented to perform this step.
   - *Create HPS Interface: HPS Interfaces* are created by associating activity types with the *Service Model*, which defines the mapping of activity types (and human actions) to services definitions (WSDL). The mapping of an *HPS Interface* into WSDL is the binding of activity type definitions and actions to HAL. At run-time, HAL acts as a proxy service dispatching requests by performing security checks, routing, message transformations (if needed), and persistency management of messages (i.e., saving request/response messages in XML collections).

3. **Automatic interface generation enabling interactions**

   - *Emit Interface:* The final step comprises the automated generation of interfaces at run-time. An *Interface Emitter* generates: i) interfaces allowing software services to interact with HPSs by generating WSDLs. Thus, HPSs may be included in process by defining human interactions (e.g., B4P) in the process definition, which are enacted as HPS actions (interaction through HAL). (ii) GUIs are generated automatically by transforming WSDLs and XSDs into XML forms.

5.5.2 Interface Mappings

We continue our discussion of HPS interfaces by showing a concrete XML example of a WSDL description. These examples will be sufficient to illustrate the idea of mapping human activities and actions onto Web services. Of course, no user input is needed to create such mappings or to create WSDL descriptions out of human activities. These transformations are automatically performed by tools. A complete XML example of the discussed HPS WSDL is provided in Appendix B.

Let us start with some exemplary type definitions. Listing 5.1 shows `GenericResource`, `ReviewRequest` type definitions.

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

Listing 5.1: Review-activity types example.

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 *review* comprising resources, the actual request, and the reply, which is a complex XML data structure (abbreviated in this example).

Listing 5.2 shows an excerpt of the mapping of human activities into WSDL messages (review HPS). However, we only show the request denoted as `ReviewRequest`.

```
<wsdl:message name="GetReview">
  <wsdl:part name="part1" element="ReviewRequest" />
</wsdl:message>
<wsdl:message name="AckReviewRequest">
  <wsdl:part name="part1" element="AckReviewRequest" />
</wsdl:message>
```

Listing 5.2: HPS WSDL messages example.

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

Listing 5.3: HPS WSDL porttype and binding example.

Notice, `PortType` (i.e., the technical interface) in Listing 5.3 for all interactions is HAL. 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., `AckReviewRequest`).

5.6 Architecture of HPS Framework

In this section we discuss the architecture of the HPS framework. The framework comprises a set of tools, for example, enabling the design of HPS, and a middleware hosting various services such as HAL. In Fig. 5.4 we show the architecture and its main components.

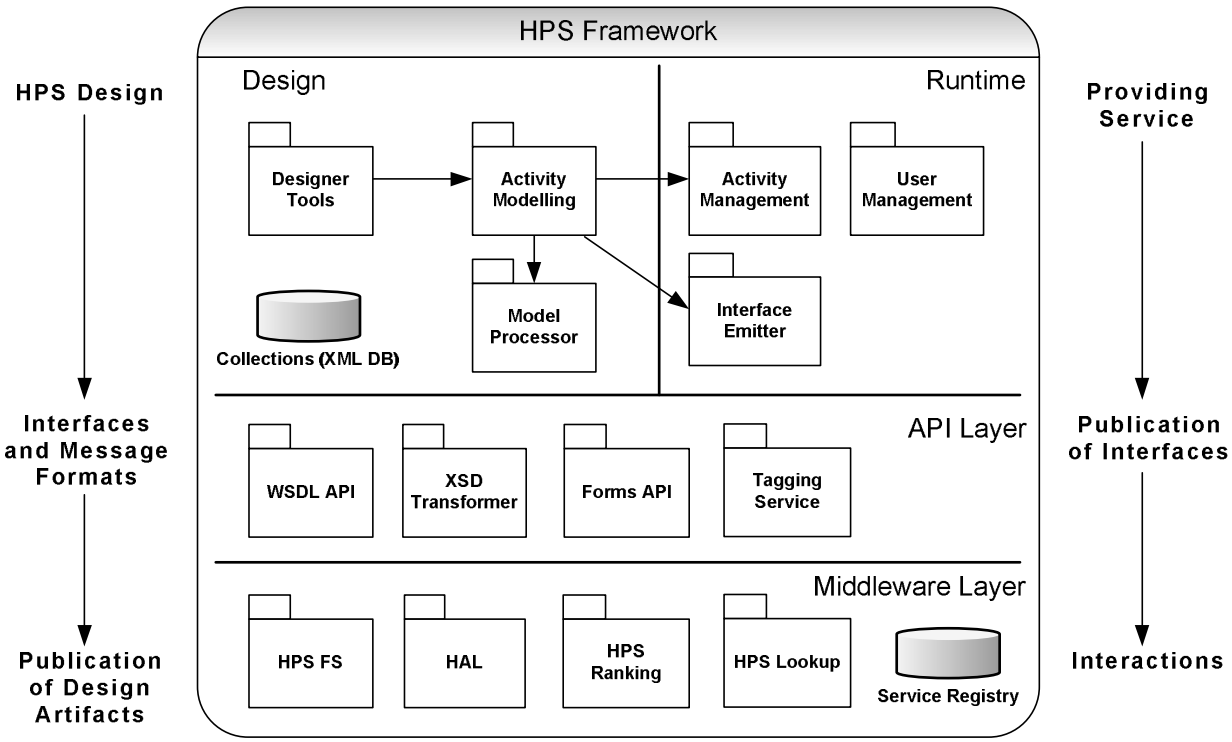


Figure 5.4: HPS framework and architecture.

5.6.1 Middleware Layer

The *HPS FS* — an XML based, distributed file system — manages user profiles, human tasks, service related information such as WSDL descriptions, personal services, and so forth. The HPS FS offers a set of APIs to manage XML artifacts and collections via the Atom Protocol Model[5] to retrieve and update HPS related information. We embed, for

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

example, HPS interfaces depicted as WSDL as elements in Atom-based XML documents (see Atom Syndication Format[6]). In other words, 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 include: (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. Also, Web service registries that are accessible via Atom-based protocols have been proposed in (Treiber and Dustdar 2007, Wu and Chang 2007) to manage metadata associated with software services.

HAL dispatches requests of various types including SOAP or JSON[7] requests and routing HPS interactions to the corresponding service. Thus, humans and software services (i.e., HPS requesters) are able to interact with HPSs by sending their requests towards the HPS middleware. In addition, HAL implements security features to prevent unauthorized access and allows requests to be routed according to user-defined rules.

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. (Chapter 4 is concerned with link-analysis algorithms for global importance ranking.) Ranking results and recommendations can be requested from a ranking service (not shown in Fig. 5.4).

The *HPS Lookup* supports various ways to discover HPSs. On the one hand, 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. On the other hand, a Web services-based API can be used to support typical lookup operations (e.g., get list of services). The middleware hosts a *Service Registry* that is used when the service lookup is performed.

5.6.2 API Layer

To support the design of HPS, the framework includes services and tools for *HPS Design* 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[8])

[6]Atom Syndication Format: `http://tools.ietf.org/html/rfc4287`
[7]JavaScript Object Notation: `http://www.json.org/`
[8]W3C Markup Forms: `http://www.w3.org/MarkUp/Forms/`

- *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)

5.6.3 Design Tools

HPS Design tools allow users to create service interfaces in a simple manner. These tools are hosted in a Web portal. Figure 5.4 illustrates the design flow on the left side (top down):

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

5.6.4 Services used at Runtime

The following services have been designed to enable HPS-based collaboration.

- The *User Management* service holds user-related data such as profiles and contact details.
- The *Interface Emitter* generates HPS interfaces depending on the interaction scenario; 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.
- The *Activity Management* service maintains activity declarations and activity instances.

5.6.5 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.

The basic protocol model specifies the use of the standard HTTP methods GET, POST, PUT, DELETE, and HEAD to retrieve, update, store, delete, or enumerate resources. Resources and collections include:

User Profile and Metrics. Profiles comprise user related information. There are many existing standards defining the structure and information of user profiles. Thus, it is not reasonable to focus on the definition of user profiles because FOAF or vCard can be used for this purpose. To briefly highlight the features of these standards: FOAF profiles are usually specified as XML/RDF documents and are often used in the context of the Semantic Web, for example (Aleman-Meza et al. 2007); vCard is a file format standard for electronic business cards. Therefore, users can specify basic information or simply import personal data that is already available in vCard profiles. Also, there are standardized ways to describe and exchange human resources-related data[9].

In this work, we make the distinction between *hard* and *soft-facts*. Hard-facts comprise information typically found in resumes such as education, employment history including organizational information and position held by the user, and professional activities. Soft-facts are represented as competencies. A competency comprises *weights* (skill level of a user), *classification* (description of area or link to taxonomy), and *evidence* (external sources acting as reference or recommendation). Soft-facts can be generated by the middleware based on users' activities to indicate expertise or skill level. As mentioned before, the structure and format of user profiles were not the main focus of this work. For completeness, we show the HPS profile model in Appendix C, Fig. C.2.

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.

[9]Human Resources Vocabularies: `www.hr-xml.org`

5.7 Metrics Characterizing HPSs

HPSs are a new type of services that are provided by human actors. Thus, we need to establish a new set of metrics that can be associated with HPSs. These metrics are able to describe human characteristics in a service-oriented economy. While we previously discussed the HPS framework by introducing the various services and tools from a "static" (deployment) point of view, here we show the dynamic architectural perspective. In particular, Fig. 5.5 shows the main principles of how HPS is used in dynamic environments.

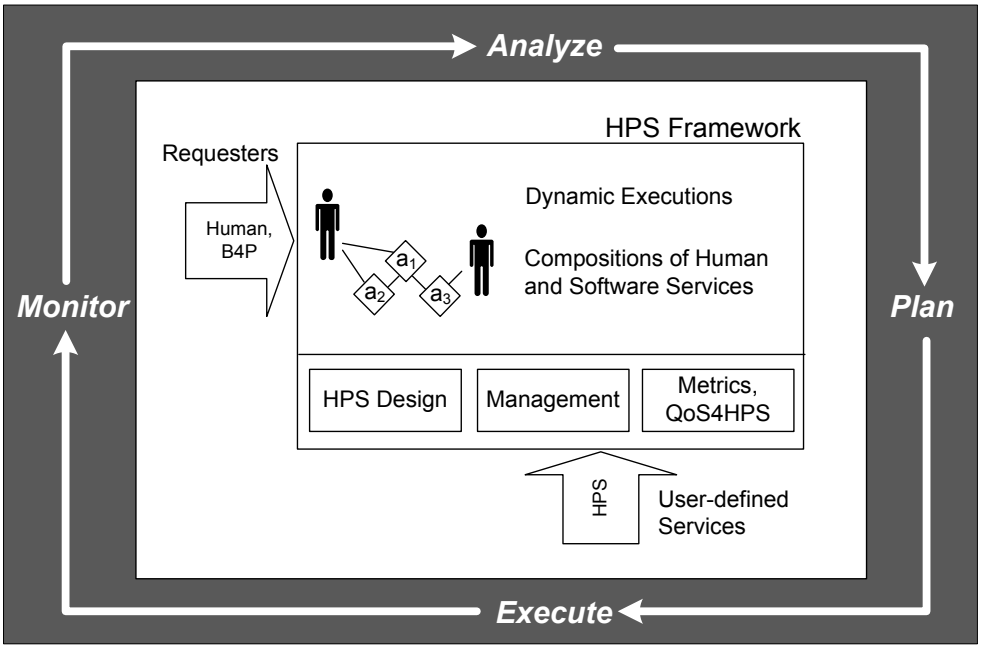


Figure 5.5: Conceptual overview of analysis and rewarding.

The HPS framework not only allows experts to define human services, but instead all people can define their contributions and capabilities as services. Such services can be invoked by B4P. Our approach is to map the HPS framework into an autonomic control feedback loop — the Monitor, Analyze, Plan, and Execute (*MAPE*) cycle; established in the context of autonomic computing. In service-oriented systems, compositions are the description of the foreseen flow of interactions between autonomous entities. We take the autonomic computing paradigm as a metaphor for the envisioned composition approach: An autonomic computing environment has the ability to manage itself and dynamically adapt to change in accordance with objectives and strategies (IBM 2005). Self-organizing environments can adapt based on the context (the runtime environment). Based on the observed context of the environment, different adaptation strategies can be applied to guide the interactions between entities, the parameters of those strategies, and actions to prevent inefficient use of resources and disruptions. The significance of these principles is

that the dynamics feed back to every aspect in the system's evolution. The formulation of compositions can no longer be thought of as a top-down design based on business or organizational requirements, but rather the specification of a system that is strongly influenced by human organization and activity. Despite of the availability of models for organizing humans and services, these compositions may fail to deliver the desired runtime behavior due to lack of control and context in complex environments. Our architecture exhibits the following key principles:

- **Monitoring** is needed to observe the actual run-time state of the system by logging interactions of heterogeneous entities including human and software services.
- **Analysis** is done to identify interaction patterns, relation and dependability of entities within the system and calculation of various metrics such as reputation. For example, envisioned patterns and metrics at the modeling or planing layer versus enacted in the real system.
- **Planning**, compositions of HPSs and software services are established by selecting the right elements constituting the set of interacting partners. Such selection or replaceability strategies can be established based on reputation mechanism. However, while analysis usually focuses on the anatomy of interactions in the real system, planning also feeds external requirements and constraints into the system, for example, *dependability* or *demanded-level* of expertise.
- **Execution** is performed as interactions between HPSs, software services, and compositions thereof. Such interactions may exhibit long-running flows and transactional behavior by incrementally updating information in compositions among peers. However, typically the exact execution of flows can not be modeled in advance, requiring adaptive behavior of processes and information flow (e.g., synchronization of entities).

5.7.1 Classification of Metrics

The classification of HPS metrics is shown in Figure 5.6 and comprises four main subclasses, which are *task, service, interaction,* and *user-related* metrics.

- *Task:* Metrics associated with tasks such as task processing statistics (e.g., number of *completed tasks*) and task metrics denoting how active a user is in a given field. For example, the number of *claimed/assigned tasks* in a particular domain can be used as input to derive user specific metrics such as *skill level* or area of interest.
- *Service:* Service related information including *availability*, *request/response ratio*, and *responsiveness*. Note in this context that HPSs might be offered to process different tasks or integrated into systems that do not require the specification of tasks.

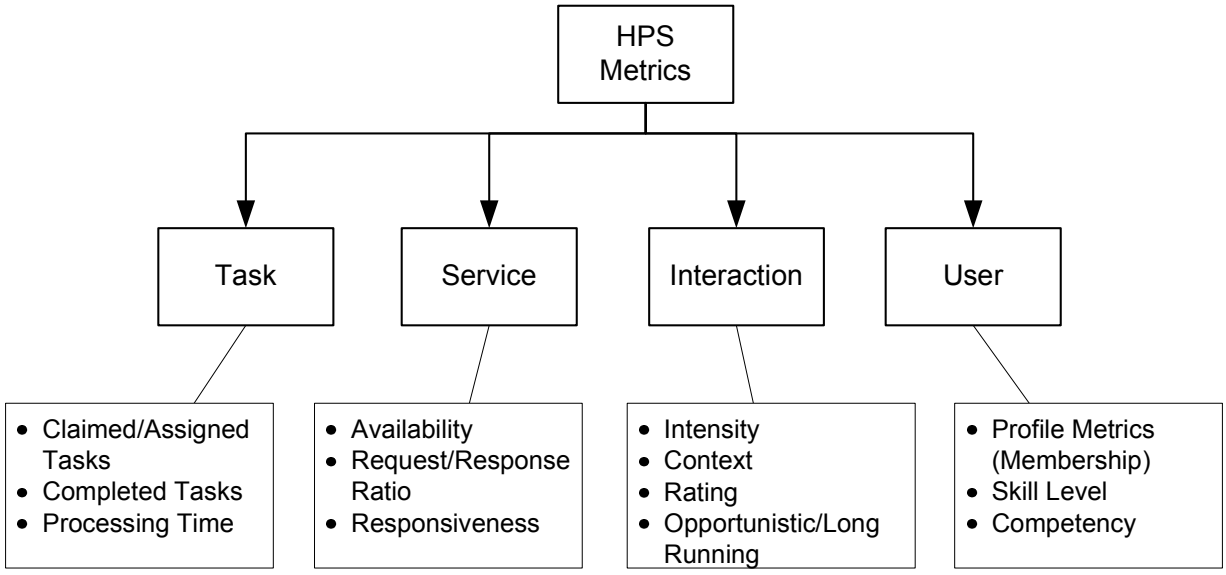


Figure 5.6: Classification of HPS metrics.

- *Interaction:* Interaction metrics are drawn from observations between at least two actors. As an example, *intensity* describes how many times a requester (expert seeker) has interacted with an HPS in a given period of time.

- *User:* User related metrics are partly obtained from information that is specified in the *User Profile* and partially derived from other metrics. For example, *membership* is a measure of involvement in different projects or professional organizations. On the other hand, *skill level* is a metric that can be derived from the user's activities in a certain area.

Various user related metrics can be obtained from the users' profiles. *Membership*, as depicted in Figure 5.6, can be used to establish a common context between users. For example, users might work on common projects or might know each other from past projects. This information can be utilized as evidences in the ranking procedure. More metrics can be calculated based on human task-/HPS related information and statistics. In addition, various metrics can be combined to establish compound metrics. However, at this stage we present these metrics independently without going into the details of composed metrics. Also, experiments and evaluations of metrics are based on individual classes of metrics.

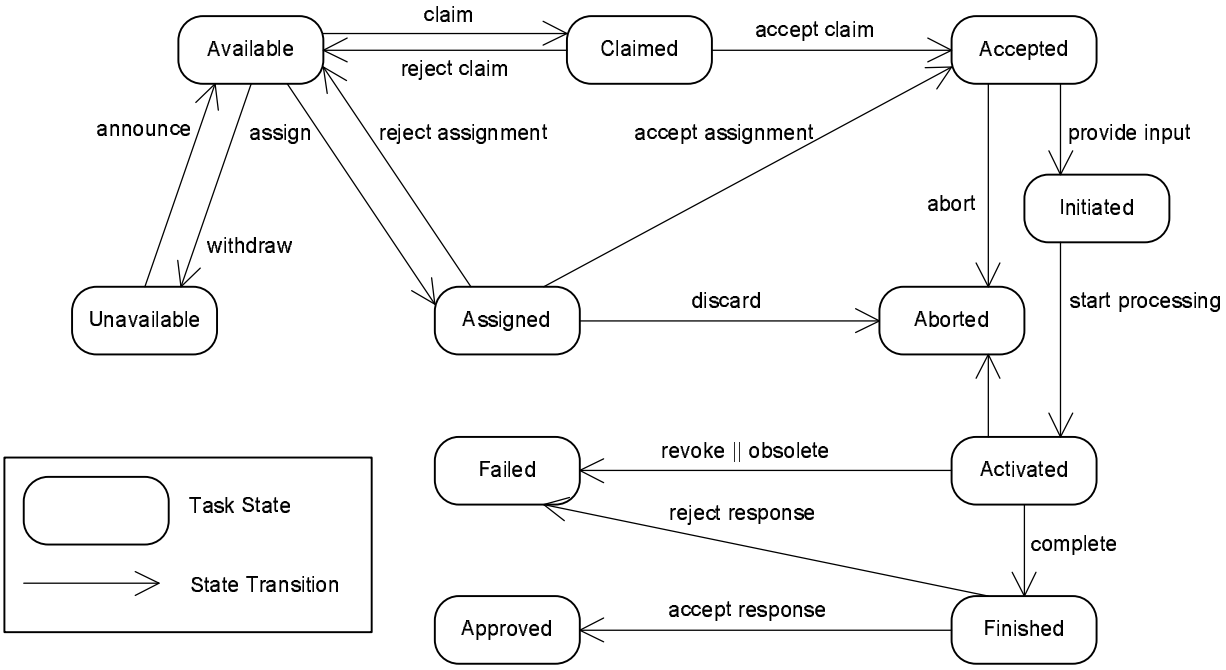
Before we start with the definition of various HPS-based metrics, we define the symbols used in the following sections.

Symbol	Meaning
T	Denotes the set of human tasks, $ht \in T$.
T^Q	The set of all possible task states, $q \in T^Q$.
$H(c)$	The set of tasks in context c, $H(c) = \{ht_i \| ht_i \in T, c \in C\}$.
s	Depicts an HPS given $s \in S$.
$HT(s)$	The set of tasks processed by s.
$S(c)$	The set of services associated with a context c.

Table 5.3: HPS metrics and related symbols.

5.7.2 Task Metrics

Furthermore, we defined a task model in Fig. 3.6, Sec. 3.4.3, showing task states and transitions between those states. Here we show the task model again — as define in Fig. 3.6 — making it easier to follow the definition and calculation of presented metrics.



- We use functions to calculate the time spent in task states
  - $TS(\texttt{Initiated}, \texttt{Activated})$: The time interval between `Initiated`, when the input was provided, and `Activated`, start working on a task.
  - $TS(\texttt{Activated}, \texttt{Finished})$: The time interval between `Activated` and `Finished`, i.e., the actual time needed to complete a task.
- Processing time PT of a task ht is defined as the time interval $TS(\texttt{Activated}, \texttt{Finished})$.

Definition 5.7.1 (Expected Processing Time) *The expected processing time of tasks in context c*

$$PT_{ex}(H(c)) = \frac{1}{|H(c)|} \sum_{ht \in H(c)} PT(ht) \tag{5.2}$$

Definition 5.7.2 (Maximum Processing Time) *The maximum processing time of tasks* $ht \in H(c)$

$$PT_{max}(H(c)) = max(PT(ht)) \tag{5.3}$$

Let us define the set $H_{approved}(c)$ comprising tasks that were `Approved` by requesters. The set $H(c) = H_{approved}(c) \cup \{ht|$ task state reached `Aborted` or `Failed`$\}$ contains `Failed` and `Approved` tasks.

Definition 5.7.3 (Success Rate) *The success rate* $SCR(H(c)) \in [0, 1]$ *is calculated as*

$$SCR(H(c)) = \begin{cases} \frac{|H_{approved}(c)|}{|H(c)|} & , \text{ if } H(c) \neq \emptyset \\ 0 & , \text{ otherwise} \end{cases} \tag{5.4}$$

5.7.3 Service Metrics

An interaction between a requester and HPS is a pair (ht, s). An interaction requires a task to be `Initiated` and set to the `Activated` state. A task is `Initiated` by sending the corresponding request to the HPS. The task goes to the `Activated` state as soon as the users starts processing the task.

Definition 5.7.4 (Service Availability Factor) *We calculate the availability factor* SA *of a service as the ratio of the actual uptime* $UT(\Delta t, s)$*, i.e., the time a service is available, in a given time interval* Δt *compared to the maximum availability of a set of HPSs* $\forall s \in S(c)$*.* $UT_{max}(\Delta t, S(c))$ *is defined as*

$$UT_{max}(\Delta t, S(c)) = max(UT(\Delta t, s)) \tag{5.5}$$

Suppose $HT_{accepted}(s)$ and $HT_{finished}(s)$ represent the set of `Accepted` tasks and the set of `Finished` tasks, respectively.

Definition 5.7.5 (Request Response Ratio) *The function* $RR(s) \in [0, 1]$ *is calculated as*

$$RR(s) = \frac{|HT_{finished}(s)|}{|HT_{accepted}(s)|} \tag{5.6}$$

$RR(s)$ calculates the request/response ratio of s. In other words, $RR(s)$ is a *measure of reliability* that s will finish accepted tasks.

Our next step is to define a metric for *responsiveness*. Successful tasks are those tasks that were accepted and then finished by the user. Let us define the set $HT'(s) = \{ht|$ task state reached `Accepted` and `Finished`$\}$ of successful tasks with the path $\pi =$ (`Accepted`, `Initiated`, `Activated`, `Finished`) in their execution traces (states that were reached when executing the task). We calculate the activation time AT of a task as $TS($`Initiated`, `Activated`$)$. We define the mean activation time as

$$AT_{mean}(S) = \sum_{s \in S} \left[\frac{1}{|HT'(s)|} \sum_{ht \in HT'(s)} AT(ht) \right] \tag{5.7}$$

We use the z-test function to determine whether the difference between $AT_{mean}(s)$ and $AT_{mean}(S(c) \setminus \{s\})$ is statistically significant. The standard error STE of the population $S' = S(c) \setminus \{s\}$ is given as $STE(S') = \sigma_{S'}/\sqrt{|S'|}$, $\sigma_{S'}$ is the standard deviation of $AT(S')$. The z score for s is calculated as

$$z(s) = \frac{AT_{mean}(\{s\}) - AT_{mean}(S')}{STE(S')} \tag{5.8}$$

Definition 5.7.6 (Responsiveness) *We define the responsiveness based on* $RP(s) = 1 - z(s)$ *as*

$$\texttt{responsiveness}(s) = \begin{cases} Low & , \text{ if } 0 < RP(s) < 0.5 \\ Expected & , \text{ if } RP(s) = 0.5 \\ High & , \text{ otherwise} \end{cases} \tag{5.9}$$

The next step is to define an aggregation function for the metrics SA, RR, and RP. The importance when synthesizing service metrics is to model the relationship between each metric. For example, how RR is related to RP and how to model the dependency between these metrics. High responsiveness is desirable, but we need to take RR into account. For example, a user might be highly responsive but unreliable in delivering desired response (i.e., RR).

We express this dependency as $RP - (1 - RR)$, calculated as $EXP(RP - (1 - RR))$ to avoid non-positive values. (EXP is a function to calculate powers of Euler's number e.) The term $(1 - RR)$ becomes negligibly small if a user has a good request/response ratio but decimates RP otherwise. Also, we need to integrate the dependency on SA in our calculation because low availability makes an HPS less valuable; even if RR and RP are high.

Definition 5.7.7 (Relative Service Goodness) *Let us define the relative goodness of an HPS as*

$$RSG(s) = EXP(w_{RSG} * (RP - (1 - RR)) - (1 - w_{RSG}) * SA) \tag{5.10}$$

The weight $w_{RSG} \in [0, 1]$ allows us to put more importance on how fast a user processes a request/task, the term $(RP - (1 - RR))$ or to put more priority on availability. For equal importance we use $w_{RSG} = 0.5$ (equal importance for both).

Figure 5.7 shows an example dataset (left), which we generated to demonstrate the effect of RSG. The vertical axis shows the data sample for SA, RR, and RP; the horizontal axis in Fig. 5.7 (left) shows each metric; and the color schema is based on generated values [0, 1]. For example, if SA has a high value in a particular data sample (e.g., sample 2, 5, and 8), we see "hot" colors as defined by the color or "temperature" scale. Furthermore, we calculate the euclidean distance $dist(1) = \sqrt{(1 - SA)^2 + (1 - RR)^2 + (1 - RP)^2}$.

$(1 - SA)$, $(1 - RR)$, and $(1 - RP)$ can be seen as components of a three dimensional vector. In the best case, each service metric has the value 1 (the highest score). Thus, $dist(1)$ returns lower numerical values of metrics that are closer to 1 (each metric is defined within [0, 1]). However, the RSG values in Fig. 5.7 depict the scores based on aggregated metric values. In other words, we show which score a particular HPS would obtain if SA, RR, and RP would obtain values according to the generated data samples.

For $w_{RSG} = 0.5$, we make the following observations: In sample 18 we see a good ranking score based on high RP bot rather moderate SA. However, since RR is also high, the user with such service metric values obtains a good RSA score. On the other hand, 17 shows very low RP and RR but good SA. Generally, if distance values $dist(1)$ are low, we also see better RSG scores. We could additionally parameterize RSG by using the weight w_{RSG}.

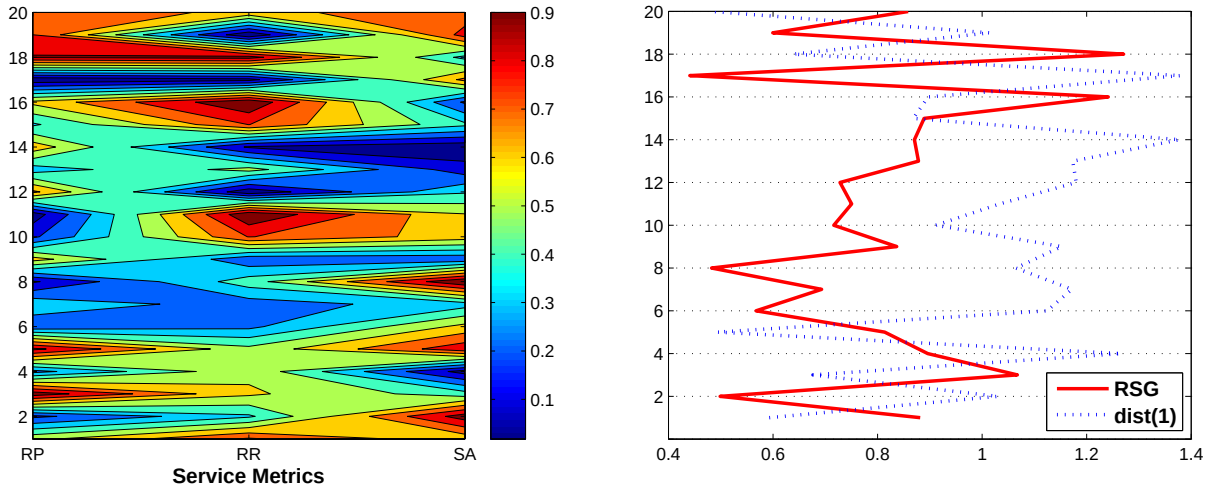


Figure 5.7: Intensity plot of artificial dataset generated for RP, RR, and SA within the range [0,1] (left) and RSA based ranking scores (right).

5.7.4 User Rating of HPS

Requesters have the ability to associate *ratings* to interactions[10]. Ratings reflect the opinion of requesters how valuable (i.e., level of satisfaction) the response is which was provided by a particular user offering an HPS.

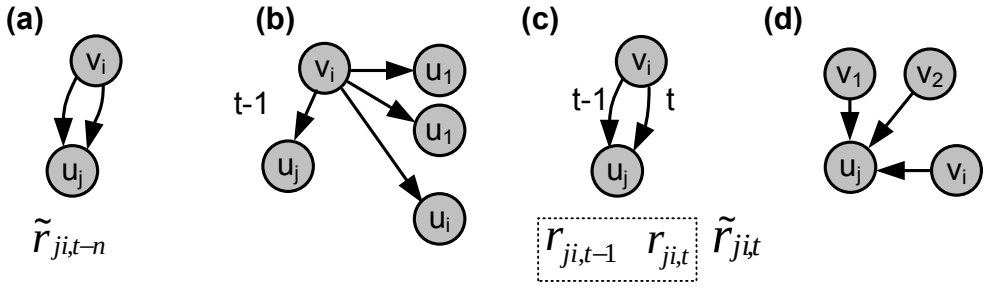


Figure 5.8: Schematic illustration of HPS rating.

We calculate $\tilde{r}_{ji}$ as the exponential moving average (EMA) to smoothen the sequence of ratings (i.e., short-term fluctuations). EMA gives more importance to recent ratings while not discarding older ratings. This model is a simple yet effective method. We perform the following steps:

- Consider interactions between v_i and u_j as an interaction trace $v_i \overset{r(t-n)}{\rightarrow} u_j, \ldots, v_i \overset{r(t-1)}{\rightarrow} u_j, v_i \overset{r(t)}{\rightarrow} u_j$. We obtain a set of ratings $\{r_{ji,t-n}, \ldots, r_{ji,t-1}, r_{ji,t}\}$, where i rates j (Fig. 5.8 (a)).

- We define η, $0 < \eta < 1$, as a coefficient to smoothen previous ratings of v_i; in particular, based on the ratings v_i has given to all other HPSs (Fig. 5.8 (b)). Therefore, the factor η expresses the relationship between the two sets of ratings $X = \{r_{u_1 i,t}, r_{u_2 i,t}, \ldots, r_{u_n i,t}\}$ and $Y = \{r_{u_1 i,t-1}, r_{u_2 i,t-1}, \ldots, r_{u_n i,t-1}\}$, $\forall u_i \in \texttt{outlinks}(v_i)$. The calculation of η is performed based on the correlation coefficient $correl(X, Y)$ and a mapping of the coefficient (see 5.4.1.3 for the definition and mapping).

- We aggregate the set of ratings into $\tilde{r}_{ji}$ associated with an interaction between i and j (Fig. 5.8 (c)). We use this formula: $\tilde{r}_{ji,t} = \eta r_{ji,t} + (1 - \eta)\tilde{r}_{ji,t-1}$.

- Finally, we calculate aggregated ratings $AR(u_j)$ for u_j based on inbound interaction links (Fig. 5.8 (d)). AR is calculated as the weighted sum of smoothened ratings, $n = |inlinks(u_j)|$:

$$AR(u) = \sum_{i=1}^{n} \frac{w(v_i)\tilde{r}_{ji,t}}{\sum_{k=1}^{n} w(v_k)} \tag{5.11}$$

[10]Providing reference or evidence.

However, this definition currently has no notion of *trust*. For example, how trustworthy the ratings of a particular user are or whether ratings should be filtered if somebody is trying to "underrate" a certain HPS.

5.8 Task Rewarding Model

In Sec. 5.7 we discussed human task related metrics including *expected processing time* PT_{ex}, *maximum processing time* PT_{max}, and *success rate* SCR. These metrics represent properties of a particular domain in which tasks are processed. In particular, $H(c)$ contains tasks associated with a certain context or category. Various functions can be used to express the relationship between task metrics.

5.8.1 Initial Rewarding Model

In this work, a function belonging to the family of sigmoid functions with the general form $P(t) = \frac{1}{1+e^{-t}}$ is used. Sigmoid functions are typically used to model systems that saturate at large values of t, for example, the processing time of tasks in HPS.

Definition 5.8.1 (Task Rewarding Function) *Let us define a function $f_T(PT(ht))$ to determine the reward that can be obtained based on the task processing time PT as:*

$$f_T(PT(ht)) = \frac{\psi}{1 + EXP\left(-\left(\frac{PT(ht)-\sigma}{\delta}\right)\right)} \tag{5.12}$$

Symbol	Meaning
$f_T(PT(ht))$	Task rewarding function based on task processing time $PT(ht)$. Lower numerical values indicate a better score.
τ	Time interval (expiration time) after which ht fails.
ψ	Defines the saturation of $f_T : [0, \tau] \rightarrow [0, \psi]$, $\psi \in [0, 1]$.
σ	Parameter to define the horizontal displacement of f_T.
δ	Parameter to define the steepness of f_T's slope.
$f_T \langle M \rangle$	Task rewarding function parameterized by the model $M(\psi, \sigma, \delta)$.
M_{α_n}	Depicts a rewarding model identified by the index α_n.

Table 5.4: Task rewarding model and related symbols.

Mathematical Background

Let us define λ as $PT_{max} - PT_{ex}$. The factor $\sigma = PT_{ex} + \frac{\lambda}{2}$ determines the displacement of f_T. The slope can be parameterized by calculating δ. Suppose that we want f_T to obtain a value $y = f_T(PT(ht))$ at PT_{max}, thus $PT(ht)$ being $f_T^{-1}(y) = PT_{ex} + \lambda$. Therefore, we need to solve $(1 + e^{-\frac{\lambda}{2\delta}})^{-1}$. We can then horizontally expand or compress f_T (the steepness of the slope) by calculating

$$\delta = \frac{0.5\lambda}{ln(y) - ln(y-1)} \tag{5.13}$$

Variable	**Description**
PT_{ex}	Determines the point where f_T monotonically increases. From this point on the score of ht exponentially declines.
PT_{max}	The point where f_T penetrates a certain threshold such that $f_T^{-1}(y) = PT_{max}$.
SCR	Determines the slope of f_T. The success rate indicates whether tasks will be successfully finished or aborted. We can model the risk in a task category by adjusting how steeply f_T should increase.
f_T''	Inflection point of f_T. The derivative of $P(t)$ is given as $\frac{dP}{dt} = P(1-P)$. Thus, given $\frac{d^2P}{dt} = 1 - 2P$, f_T'' can be approximated as: $$f_T'' \approx \min_{ht \in H(c)} \lvert 1 - 2(f_T(ht)) \rvert$$

Table 5.5: Mathematical background initial task rewarding model.

Risk	Range	Numerical Example: $f_T^{-1}(y) = PT_{ex} + \lambda$
Low	$1 \geq SCR > \xi_1$	Given that $y = 0.9$ and $\xi_1 = 0.8$, the numerical value of $\delta = 13.6536$. Case $f_T \langle M_1 \rangle$.
Medium	$\xi_1 \geq SCR \geq \xi_2$	The penetration of f_T at $PT_{ex} + \lambda$ is set to $y = 0.8$ and $\xi_2 = 0.5$, δ becomes 21.6404 (see $f_T \langle M_2 \rangle$).
High	$\xi_2 > SCR \geq 0$	We set $y = 0.65$. The variable δ obtains the value 48.4622. The corresponding function is $f_T \langle M_3 \rangle$.

Table 5.6: $f_T \langle M \rangle, M \in \{M_1, M_2, M_3\}$.

Remark 5.8.2 (Numerical Examples) *Table 5.6 shows the numerical values for $f_T \langle M \rangle$ as depicted in Figure 5.9 ($\sigma_1 = \sigma_2 = \sigma_3$ and $\psi_{1...3} = 1$).*

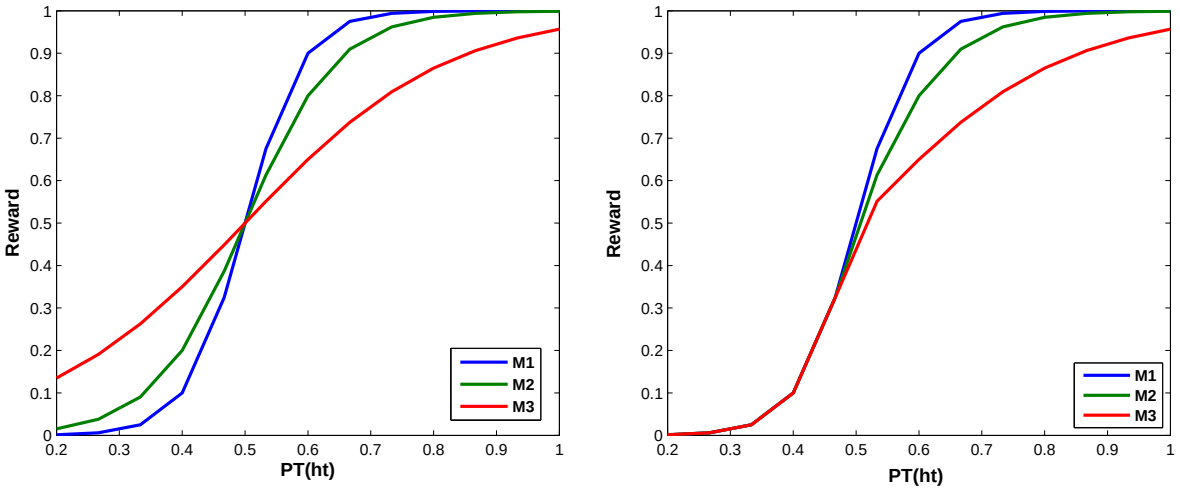


Figure 5.9: $f_T\langle M\rangle(PT(ht))$ (left) and refined model $f_T^*\langle M\rangle(PT(ht))$ (right): horizontal axis shows the processing time PT of a task, normalized given time interval τ (expiration time after which ht fails); vertical axis shows the reward or score of a task.

Figure 5.9 shows examples for f_T whose shape is parameterized by rewarding models M_1, M_2, and M_3. For example, the case $SCR_{M_1} > SCR_{M_2} > SCR_{M_3}$. As we can see in Fig. 5.9, f_T gives a higher score to those tasks that were processed in domains where the success rate SCR is low. The intuitive justification for this behavior is that tasks that are processed in a "high-risk" domain should be rewarded higher since each finished task is of high value compared to domains where SCR is generally high.

However, we see that f_T gives lower scores at $PT(ht) < \sigma$ (in Fig. 5.9 (left), the task processing time 0.5) for tasks being situated in $\xi_2 > SCR$. Therefore, we need to refine $f_T\langle M\rangle(PT(ht))$ and define a step-wise function.

Definition 5.8.3 *Let us define f_T^* as follows:*

$$f_T^*\langle M\rangle(PT(ht)) = \begin{cases} f_T\langle M_\Lambda\rangle(PT(ht)) & , \text{ if } \frac{d^2P\langle M\rangle}{dt} < f_T''\langle M\rangle \\ f_T\langle M\rangle(PT(ht)) & , \text{ otherwise} \end{cases} \tag{5.14}$$

M_Λ determines f_T's shape before reaching the inflection point. In our experiments we set $M_\Lambda = M_1$ (i.e., low risk determined by high SCR). For an example, see Fig. 5.9 (right).

5.8.2 Trend-based Rewarding Model

The function f_T associates a score or reward tr with tasks based on the task processing time PT such that $f_T : [0, \tau] \rightarrow tr$. Thus, given that a user processes a set of tasks over

time, we obtain a set of task rewards $\{tr_{ij} | i \in \{HT(s)\} \text{ and } j \in \{S\}\}$. (For simplicity, we depict j as the user providing a particular HPS to process tasks.)

Proposition 5.8.4 *A trend can be calculated by correlating tr_{i-1j}, obtained in $f_T^* \langle M_a \rangle$, and tr_{ij} in $f_T^* \langle M_b \rangle$, $a = b \vee a \neq b$.*

Proof 5.8.5 *$\mathcal{M} = \{M_{\alpha_1}, M_{\alpha_2}, \ldots, M_{\alpha_n}\}$ is a set of rewarding models depending on, for example, time $\alpha_1 = t_1, \alpha_2 = t_2, \ldots, t_1 < t_2 < \ldots < t_n$. Figure 5.11 shows the surface of $f_T^* \langle M_\alpha \rangle PT(ht)$. Let us define the vectors $\vec{v}_m(M_{\alpha_m}) = \binom{PT(ht_{i-1})}{tr_{i-1j}}$ and $\vec{v}_n(M_{\alpha_n}) = \binom{PT(ht_i)}{tr_{ij}}$. These vectors define points on the surface in Figure 5.11. The projected vector $\vec{p}_{mn} = \binom{PT(ht_{i-1})}{\hat{tr}_{ij}}$ transposes $tr_{i-1j}(M_{\alpha_m})$ into an estimated task reward $\hat{tr}_{ij}(M_{\alpha_n})$ in $f_T^* \langle M_{\alpha_n} \rangle$. The cosine of the angle between two vectors is calculated as $\cos\varphi = \vec{v}_n \vec{p}_{mn} / (\|\vec{v}_n\| \, \|\vec{p}_{mn}\|)$. The reward tr_{ij} can be rewritten as a complex number where φ is the argument of $\underline{tr}_{ij} = tr_{ij} \cdot e^{\mathbf{i}\varphi}$.*

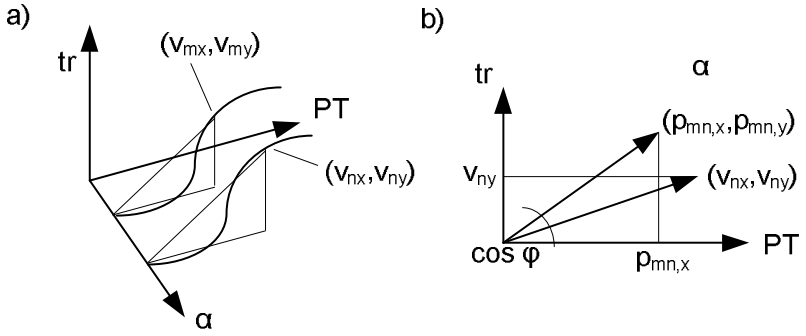


Figure 5.10: Calculating scoring trend.

Figure 5.10 visualizes the calculation of $\underline{tr}$. On the left hand side in Fig. 5.10 (a), f_T^* is illustrated and the vectors $\vec{v}_m$ and $\vec{v}_n$ in their coordinate representations. In Fig. 5.10 (b), which shows the $PT - tr$ space by projecting $\vec{v}_m$ into M_{α_n}. We determine an *increasing*, *decreasing*, or *unchanged* trend by comparing the scales of $\vec{v}_n$ and the projected vector $\vec{p}_{mn}$, $sc = \sqrt{p_{mn,x}^2 + v_{n,y}^2} / \sqrt{p_{mn,x}^2 + p_{mn,y}^2}$.

Quotient	$sign(\log(c))$	Trend	Remark
$sc < 1$	-1	increasing trend	
$sc = 1$	1	decreasing trend	$\cos\varphi = 0$
$sc > 1$	1	unchanged	

Table 5.7: Determining the imaginary unit of $e^{\mathbf{i}\varphi}$.

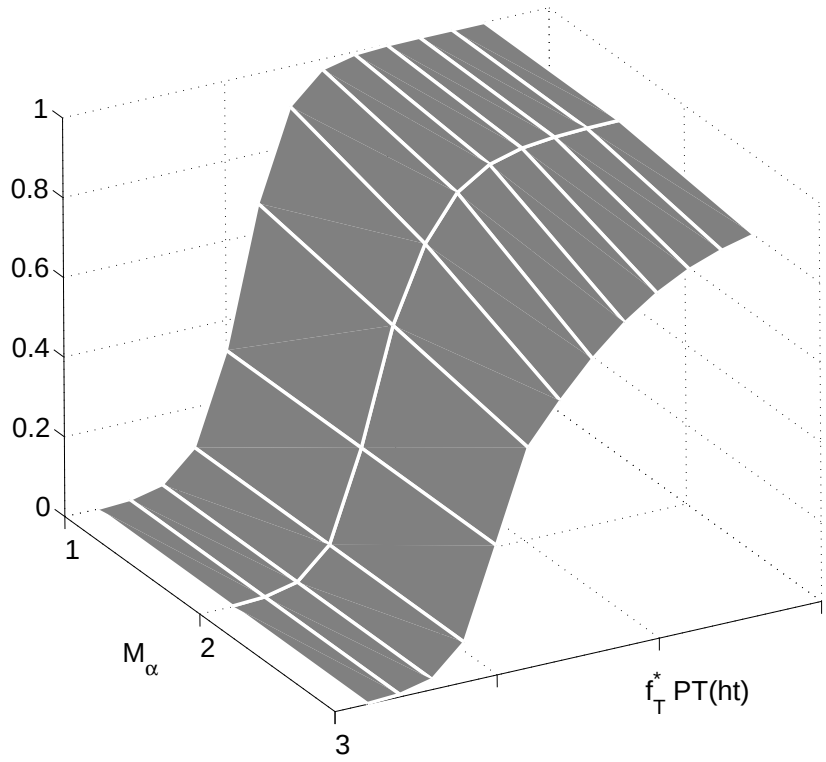


Figure 5.11: $f_T^* \langle M_\alpha \rangle PT(ht)$: illustrating task metrics for different scoring models.

Chapter 6

Implementation

6.1 Abstract

In this chapter we present the implementation details of the HPS framework. The HPS framework architecture has been presented in Chap. 5, Fig. 5.4, and comprises the *Middleware Layer*, the *API Layer*, *Design*, and *Runtime*. We discuss the technologies used in this work as well as the implementation details of various services. Furthermore, we present a detailed overview of *HPS Ranking* and interaction analysis implementations. In Appendix F, we present code listing relevant for interaction analysis. In addition, Appendix A provides screenshots of user interfaces. A discussion on middleware related components and services was presented in (Schall et al. 2008a).

6.2 Middleware Layer

The middleware layer has been designed with the objective of providing a set of reusable APIs and libraries. Specifically, we adopted *OSGi* (Open Services Gateway Initiative) as deployment and runtime container.

6.2.1 Background

OSGi has been specified by the Open Service Gateway Alliance[1]. Various Java implementations exist of OSGi containers. We give some application examples of OSGi in the following:

- Enterprise-grade services — to enable a mix of services implemented in heterogeneous technologies; see Apache ServiceMix[2].
- Platform runtime — to provide a managed runtime for dynamic, service-oriented platforms; for example, see Equinox[3] OSGi.
- Sensor networks — to enable dynamic provisioning of distributed, managed services on embedded, networked sensor platforms; see (Schall et al. 2007, Section 5).
- Mobile phones — to manage and compose various services on embedded devices.
- Automotive domain — to provide an integrated environment for telematic services.

6.2.2 Managing XML Collections in HPS Framework

The *HPS FS* is an abstraction layer to manage XML collections within the HPS framework. It provides the basic features to manipulate XML data as well as means to discover/register services, metadata, and users.

6.2.2.1 Design Considerations

- Interoperability: We chose XML to represent collections of data to account for interoperability and standard compliance grounded in Web services technologies. XML is a mature technology to model data using XML schema, annotation of data, and exchange of XML data in distributed systems.
- Extensibility: Data models can be extended by using existing models and agreed upon specifications.

[1] `www.osgi.org/main/`
[2] `http://servicemix.apache.org/`
[3] `http://www.eclipse.org/equinox/`

- Scalability: XML collections of data can be distributed across multiple repositories, thereby achieving scalability in distributed environments.
- Accessibility: Data can be accessed by locating and querying repositories to obtain resources expressed in XML.

6.2.2.2 Implementation Aspects

- Query & Update: We use the XQuery API for Java[4] to connect to XML data sources. On top of this API, the HPS framework offers CRUD capabilities to manage tasks, users, service artifacts, and so forth. We use a native XML database to manage resources (XSDs, WSDL, profiles, personal services definitions) and an SQL database with XQuery capabilities to manage various types of messages.

  This design decision was made to provide better performance in dispatching requests, logging of interactions, and processing of large numbers of messages in interaction analysis algorithms. Entity-relationship models describe the relations between entities and associated metadata; whereas the actual XML data can be saved in remote repositories. The detailed database schema is shown in Fig. C.1 in Appendix C.

- Modules: Allow for extensions of elements in XML documents. Specifically, we adopt and extend the Atom schema as a container format for elements such as tasks, personal services, etc. A model can be specified by defining an interface containing the definition of the namespace to determine the scope of the model; and the methods `setDraft`, `getDraft`. The interface is inherited from the `Module` interface as defined in *Rome*'s API[5]. Each module implements a `ModuleParser` and a `ModuleGenerator`, which are responsible for managing the content under a given namespace.

- Protocol Layer: The protocol layer defines how resources can be manipulated. In particular, resources can be accessed and manipulated by using the HTTP methods GET, PUT, POST, HEAD, DELETE in a transparent way, regardless of the resources' actual location (i.e., repository hosting a resource). This resource-centric interaction style is based on the *Representational State Transfer (REST)* architecture as introduced by Fielding (2000).

6.2.2.3 XML Examples

In this section we show actual XML examples illustrating i) the information saved in the HPS registry and ii) public tasks that can be associated with HPSs.

HPS registry. Listing 6.1 shows an XML example of a service feed. Atom is used as the XML format to associate metadata such as `title`, `updated`, etc. with resources. As mentioned before, the Atom format can be extended by specifying modules to handle

[4]XQuery API for Java (JSR 225): `http://jcp.org/en/jsr/detail?id=225`
[5]Java tools for RSS and Atom feeds: `https://rome.dev.java.net/`

additional content elements, for example new extensions and definitions or inclusion of existing metadata standards.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <feed xml:lang="en" xmlns="http://www.w3.org/2005/Atom">
3    <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af8</id>
4    <title>HPS Index</title>
5    <updated>2007-09-24T23:00:35Z</updated>
6    <link href="http://{HAL}/atom?services"
7          rel="self" type="application/atom+xml"/>
8    <entry>
9      <id>urn:uuid:60a76c80-d399-11d9-b93C-1113939e0af8</id>
10     <updated>2007-09-24T23:00:35Z</updated>
11     <title>Review Service</title>
12     <link rel="alternate" type="application/atom+xml"
13           href="/services/reviewservice.xml"/>
14     <category term="humanrevieweddata" label="documentreview"/>
15     <category term="soap" label="soap+xml"/>
16     <category term="rest" label="json"/>
17   </entry>
18 </feed>
```

Listing 6.1: XML example of HPS registry feed.

Listing 6.1 contains the following elements:

- A (self) reference to the HPS index document (line 6 - 7), which is the service feed containing *definitions* of HPSs. The service feed can be retrieved from an `atom` servlet hosted by HAL.
- The service feed contains multiple entries (for example, one entry is shown in line 8 - 17). These entries contain a `link` to the XML document that defines the HPS interface. For example, the "reviewservice.xml" would contain a WSDL definition of a human-provided "review service". The focus of our implementation is the use of Web service technology and the support of related standards such as WSDL and SOAP. Thus, the default HPS interaction format (standard) is in this example SOAP-based (i.e., `soap+xml`).

  In addition, other Web formats can be used in interactions. In this specific example, we show the `category json` associated with the review service entry. This means that both technologies, SOAP and/or JSON, can be used for interactions. Typically, a one-to-one mapping of more complex XML data structures to JSON is not possible. However, a detailed discussion on JSON versus XML-based Web formats and protocol mappings are not provided in this work.

Task registry. Human tasks are managed by the task registry. The task registry offers an API to publish announcements and to manage task instances (control tasks). Listing 6.2 shows an example of a task announcement.

```
1  <feed xmlns="http://www.w3.org/2005/Atom"
2        xmlns:ht="http://www.myhps.org/schemas/task.xsd">
3   <title>HPS Tasks</title>
4   <updated>2007-09-24T18:30:02Z</updated>
5   <id>urn:uuid:63a99c80-d399-12d9-b93C-0003939e0a</id>
6   <entry>
7    <title>HPS Public Tasks</title>
8    <updated>2007-09-19T18:30:02Z</updated>
9    <id>urn:uuid:1223c696-cfb8-4ebb-aaaa-80da344ea6</id>
10   <link href="http://{HAL}/atom?tasks" rel="self"
11         type="application/atom+xml" />
12   <link rel="alternate" type="application/atom+xml"
13         href="/services/reviewservice.xml" />
14   <category term="humanreviewedata"
15         label="documentreview" />
16   <description>
17    <![CDATA[ description of human task ]]>
18   </description>
19  </entry>
20 </feed>
```

Listing 6.2: XML example of public tasks.

- Similar to the service feed, the HPS tasks document contains a (self) reference (line 10 - 11) to the Atom-based servlet hosted by HAL.
- In this example we show that announcements are tagged with HPS information, which may already exist in form of existing taxonomies, interface descriptions, or other expertise and interaction information. The announcement contains a list of public tasks that should be processed by a particular HPS type ("reviewservice.xml" line 12 - 13).

  Also, the `category` element can be used to add additional tags to public tasks. The excerpt of the XML structure depicts that tasks are associated with a certain `category` of HPSs by applying keywords such as "humanreviewedata" and "documentreview" (line 14 - 15).
- Detailed definitions of human tasks are omitted in this example (`description` line 16 - 18) because human tasks are well-known in Web-based platforms such as Amazon Mechanical Turk and in workflow-based systems (e.g., WS-HT).

6.2.3 HPS Access Layer

The HPS Access Layer (*HAL*) manages access to all resources in the framework, routes requests, and logs interactions. It provides the entry point for all interactions with HPSs.

6.2.3.1 Design Considerations

- Transparance: Interactions between requesters and HPSs take place regardless where the user is located. Interactions are not limited by, or dependent on, the device at hand.
- Security: The privacy of users and security of data should be protected by providing pre-filtering of requests and blacklisting of untrustworthy requesters.
- Continuity: Access to various resources should be guaranteed as well as interactions in environments with disruptive network connectivity.
- Synchronicity: Interactions can take place in different modes:
  - Online: dispatch and forward requests to the demanded service(s) immediately.
  - Near-real time: dispatch and cache requests; delivery of requests to the demanded service(s) may take place in pull mode.
  - Offline: manage requests in collections independent of the users availability.

6.2.3.2 Implementation Aspects

- Message Processing: The implementation of the front-end servlet passes each request to the `MessageProcessor`, which dispatches and logs messages as service interactions. We implemented message header/body processing capabilities — depending on the `content-type` of the underlying message exchanged in an interaction. A `ConcurrentQueue` manages messages to be logged in the desired format and repository. The default format in HPS is a `SOAPInteraction` object which contains header information and typed message bodies. Both, headers as well as the content of a message, are made available for inspection, filtering and routing; thus providing the capabilities for blacklisting and content-based routing. This can be accomplished by specifying interaction rules. We use the *JBoss Drools*[6] rules engine, which can be deployed within the HPS middleware environment.
- Routing: Interactions in HPS may take place using various message formats. Although the aim is to use Web services technologies, for example SOAP, and various Web services standards; interactions in other formats are supported by HAL.

[6] JBoss Drools: `http://www.jboss.org/drools/`

– SOAP Router: Routes messages based on WS-Addressing[7] information such as `To` and `From`. HAL uses the `WSDL4J` library to check each message for conformance with the WSDL interface definitions. Furthermore, automatic responses can be generated, for example, to generate conversation identifiers. Therefore, interactions with HPSs take place asynchronously by maintaining conversation (session) identifiers.

– JSON Router: The aim of the HPS framework is not only to support interactions by using standard Web services formats such as SOAP and XML, but also integration with other Web technologies such as JSON. Specifically, JSON is well suited to enable the exchange of messages when requesters use Web browsers to interface with HPSs. Requests are then specified by providing the input information needed so that HPSs can work on requests and tasks.

  For this propose, HAL allows HPSs to be addressed via `URI`s and supports message conversion, JSON to XML and back, depending on the service endpoint's (technical) capabilities. We have implemented various scripts to support the AJAX-based exchange of messages and the dynamic inclusion ("injection") of HTML rendering elements in Web browser-based client environments. A detailed example will be given in Sec. 7.3.

Listing 6.3 shows an example configuration to deploy HAL as a bundle in OSGi (Equinox) container environments. Various OSGi-based services — such as HAL and HPS FS — can be discovered in a single container, or even distributed containers by extending OSGi's registry/discovery capabilities, to composed capabilities. For example, HAL uses the HPS FS and its XML APIs to manage the discovery, registration, and update of HPS related information via an `AtomServlet`.

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: HAL OSGi Plug-in
4  Bundle-SymbolicName: hps.osgi.hal;singleton:=true
5  Bundle-Version: 1.0.0
6  Bundle-Activator: org.myhps.hal.Activator
7  Import-Package: org.eclipse.equinox.http.registry,
8   org.osgi.framework;version="1.3.0",
9   org.osgi.util.tracker,
10  org.myhps.framework
11 Require-Bundle: javax.servlet
12 Bundle-ClassPath: .,
13  lib/kxml2.jar,
14  lib/sqljdbc.jar
```

Listing 6.3: Sample of HAL configuration and deployment in OSGi container environment.

[7] WS-Addressing: `http://schemas.xmlsoap.org/ws/2004/08/addressing`

6.3 HPS Design

The design of HPS interfaces has to be aided by tools suitable for people who do not have any programming skills; people who are not familiar with Web service technologies and service-oriented architectures. First, we discuss APIs and services within the *API Layer*.

6.3.1 HPS Design APIs

The HPS middleware services have been implemented in Java using an OSGi container as hosting environment. The HPS design APIs and services have been developed in ASP.NET/C# utilizing different subsystems via Web/REST services such as WSDL4J. Figure 6.1 details the implementation aspects.

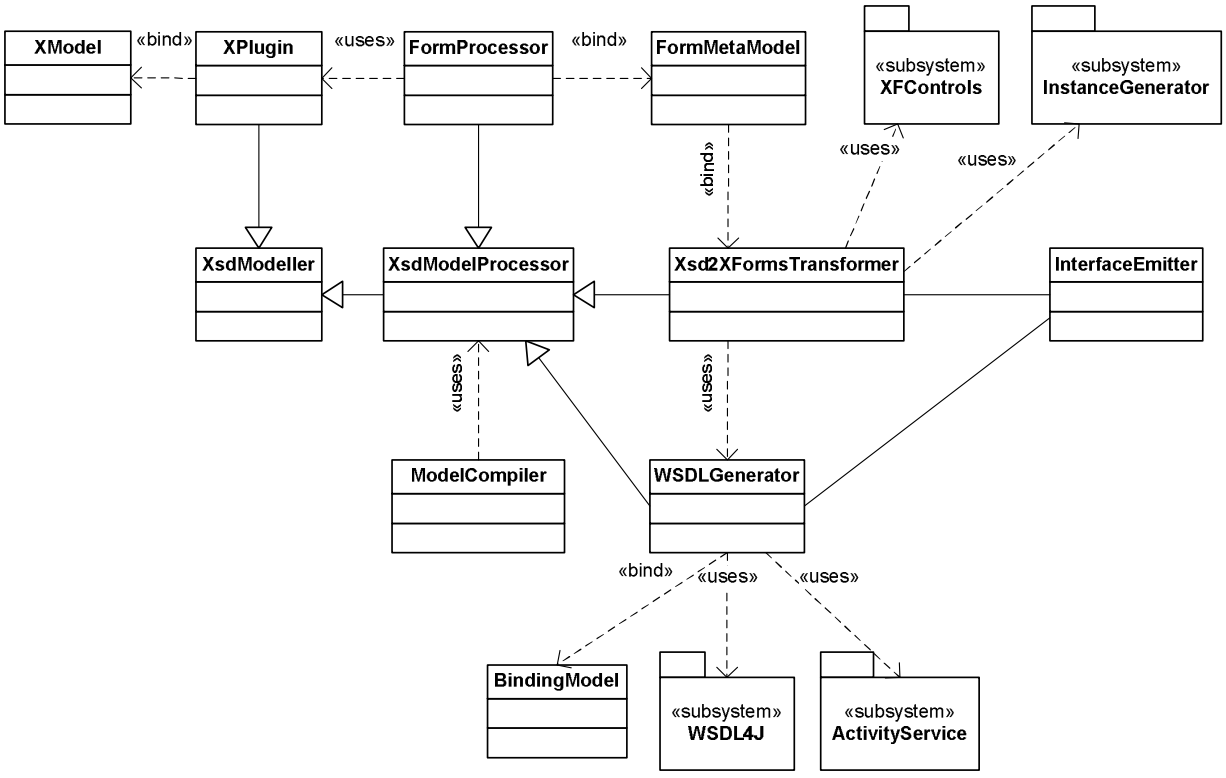


Figure 6.1: Implementation HPS design.

- The `FormProcessor` can be extended by various plugins, denoted as `XPlugin`. These plugins allow the user to specify complex data structures. The `XModel` defines the *Meta Model* to translate user descriptions into XML schema.
- The `ModelCompiler` invokes the `XsdModelProcessor` to transform user descriptions (HPS interfaces) into XML schema.

- The `Xsd2XFormsTransformer` translates the XML schema into XForms, which can be rendered in browser-based environments, for example by using the Mozilla XForms plugin[8]. On the other hand, various devices can be used by binding XML forms descriptions to specific controls (e.g., Midlet controls in JavaME environments).
- The `InterfaceEmitter` generates WSDLs or forms depending on the HPS requester's preferences.
- The `InstanceGenerator` automatically generates and prepopulates instance documents based on the XML schema because at run-time XForms are bound to instance documents (e.g., SOAP messages).
- The `ActivityService` allows users to manage activities that are performed by users by associating *Actions* with them.

6.3.2 Meta Model for Interface Mappings

In this section, we present an exemplary *Meta Model* to support the mapping of GUI elements to XML schema and forms. Table 6.1 shows the mapping of an XML type into an XForms representation.

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

Table 6.1: Example of interface mapping.

- *XSD* is the predefined XML schema type.
- *Form* defines the mapping of XML schema to forms (GUI) representations.
- *Restriction/Model* is used to map a user control (`XPlugin`) to XML schema. In other words, the user selects the desired elements (enumerations, choice, etc.) which are then automatically translated to schema types.

The design of domain specific services can be tailored to the requirements of users. For example, by modifying the behavior of user interface elements or by changing the schema binding. The meta model can also be changed to better fit the appearance of the user interface based on device requirements (e.g., binding and rendering of XForms on Java mobile platforms). More descriptions of interface mappings are given in Appendix Sec. B.

[8]Mozilla XForms: `http://www.mozilla.org/projects/xforms/`

6.3.3 XML Examples

The following XML examples show how XForm technology is used for SOAP-based interactions. A form represents an XML document, for example a WSDL, and its complex data types. There are two essential steps:

1. The user inserts data or selects predefined elements in a given form.
2. When a user submits a form, the XML instance document, which is a SOAP envelope, is populated with the data that are specified in the form's GUI elements.

6.3.3.1 Input Form Representation

Listing 6.4 shows the XML representation of a form, which can be displayed by XForm-compatible clients. Forms are used by a) human requesters to create an HPS request (i.e., the request document) and b) by users offering HPSs to work on those requests. The `switch/case` construct defines the behavior of the form — *request* and *response* representation.

```
1  <xf:switch>
2  <xf:case id="request" selected="true">
3   <xf:label>Input definitions</xf:label>...
4   <xf:submit submission="submit-envelope">
5   <xf:message ev:event="xforms-submit-done"
6     ev:observer="submit-envelope">...
7   </xf:message>
8   </xf:submit>
9  </xf:case>
10 <xf:case id="response">
11  <xf:output ref="{reference-in-instance-document}"
12    model="model-envelope">...
13  <xf:label>Submission output.</xf:label>
14  </xf:output>
15 </xf:case>
16 </xf:switch>
```

Listing 6.4: Input form representation.

6.3.3.2 XForm Model for SOAP

The next step is to wrap requests in SOAP envelopes. Each message and request is sent towards HAL, which inspects, routes, and saves requests. SOAP messages contain header elements such as WS-Addressing information which comprise *MessageID*, *To*, and other information that is used for message routing and filtering. Listing 6.5 shows how XForms are used to submit HPS requests to HAL as SOAP instance documents.

```
1 <xf:model id="model-envelope">
2 <xf:instance id="instance-envelope" src="review.xml" />
3 <xf:submission id="submit-envelope" action="{HAL}"
4     method="post" mediatype="text/xml" replace="instance">
5  <xf:toggle ev:event="xforms-submit-done" case="response" />
6 </xf:submission>
7 </xf:model>
```

Listing 6.5: SOAP interaction model.

- The `submit-envelope` element defines what happens when the user clicks submit. The `action` tag defines the endpoint (HAL in this case) of the submission target.
- The instance document is modified and submitted upon submission ("review.xml").
- The `toggle` tag defines that the response should be shown once the submit action was performed.

6.3.4 Implementation User Tools

In this section we show user tools to find HPSs and software services; and furthermore tools supporting the design of HPS.

6.3.4.1 Registry and Lookup

The first tool is based on a screenshot showing aspects of a portal that was implemented by the European Microsoft Innovation Center as part of the inContext service management system. Although the service management portal was designed and implemented for Web services professionals and developers; the screenshot will be used to illustrate the basic idea of the HPS registry.

Figure 6.2: Registry maintaining WSDL descriptions of human and software services.

Two features are shown in the screenshot:

- `Registry:` new services can be registered or existing services can be listed and/or unregistered.
- `Lookup:` user can search for existing services.
  1. The user specifies a query, which comprises a set of keywords.
  2. Existing services are matched and ranked. The ranking can be performed as discussed in the HPS design, for example based on user profiles or preferences.
  3. A list of services is returned and displayed in a ranked order. The first result is a `ReviewService` — the example we already used in our previous discussions. This service is provided by a human actor and has two Web service operations: `GetReview` (the request) and `GetReviewReply` to obtain the review response. Notice, `GetReviewReply` is not the synchronous response of `GetReview`. At the technical (Web services) level, it is a Web service request to obtain the review reply in an asynchronous manner.

6.3.4.2 User Control for HPS Design

The HPS framework provides a set of design tools offering features in a Web 2.0 enabled portal. Using the Web portal, people can define control elements, options, and definitions of what format a request and the corresponding response should have. The specified high level description is automatically translated into low-level XML documents such as XML schema, instance documents, WSDL descriptions and XForms depending on the requester; a human requester or (software) process. The approach we take is that users should have the ability to design which service they want to provide. The HPS design tools are flexible and can be extended by adding different controls. These controls enable the design of complex data structures. With the help of meta models, user interface elements can be translated to XML schema. In addition, integration with other systems, for example B4P implementations, can be achieved because in HPS we follow a Web services/SOA-based approach; that is, based on open XML standards such as WSDL and SOAP.

A screenshot of a control is given in Figure 6.3, allowing users (experts) to define elements of activity types that are mapped onto Web services. In the given example, we see a generic question template which is customized by users who design activity types. Using the HPS design tool, all XML artifacts needed to enable human interactions in SOA are generated on the fly.

- `Preview as Form` to render an XForm presentation.
- `Show as XML` (for demo purposes) translates user defined elements into XML schema.
- `Show as XML Instance` (for demo purposes) displays the associated XML schema instance document.

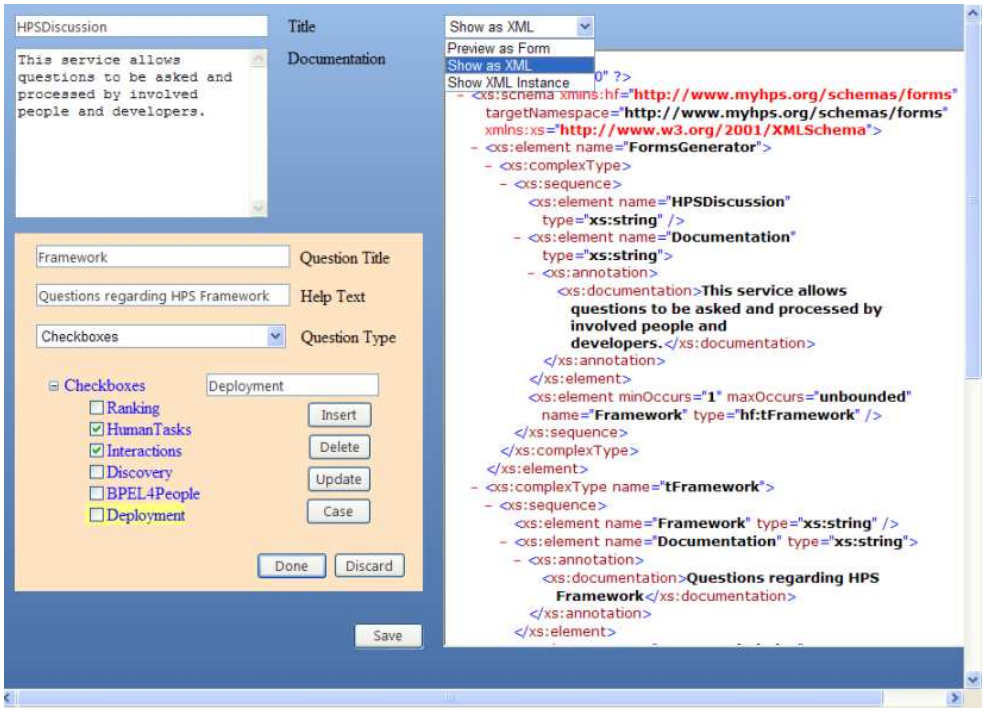


Figure 6.3: Example user control.

6.4 HPS Interaction Analysis and Ranking

In this section we discuss the *HPS Ranking* architecture and its implementation within the HPS framework. Packages implemented in this subsystem include the global importance ranking algorithm as presented previously in Chap. 4 and collaborative filtering algorithms that are used in the design of HPS.

6.4.1 Architecture

The architecture of implemented interaction analysis APIs is illustrated in Fig. 6.4. At a high level, these packages can be categorized in `Link Analysis API`, `Graph API`, and `Interaction DB`. The database schema for capturing HPS interactions can be found in Appendix C. The aim of HPS ranking and related libraries is not only to study importance ranking based on HPS interactions (i.e., in SOA), but human and service interactions in general. Therefore, the methodology we apply is to use different datasets, for example, interactions based on human collaborations, to study different metrics and ranking algorithms.

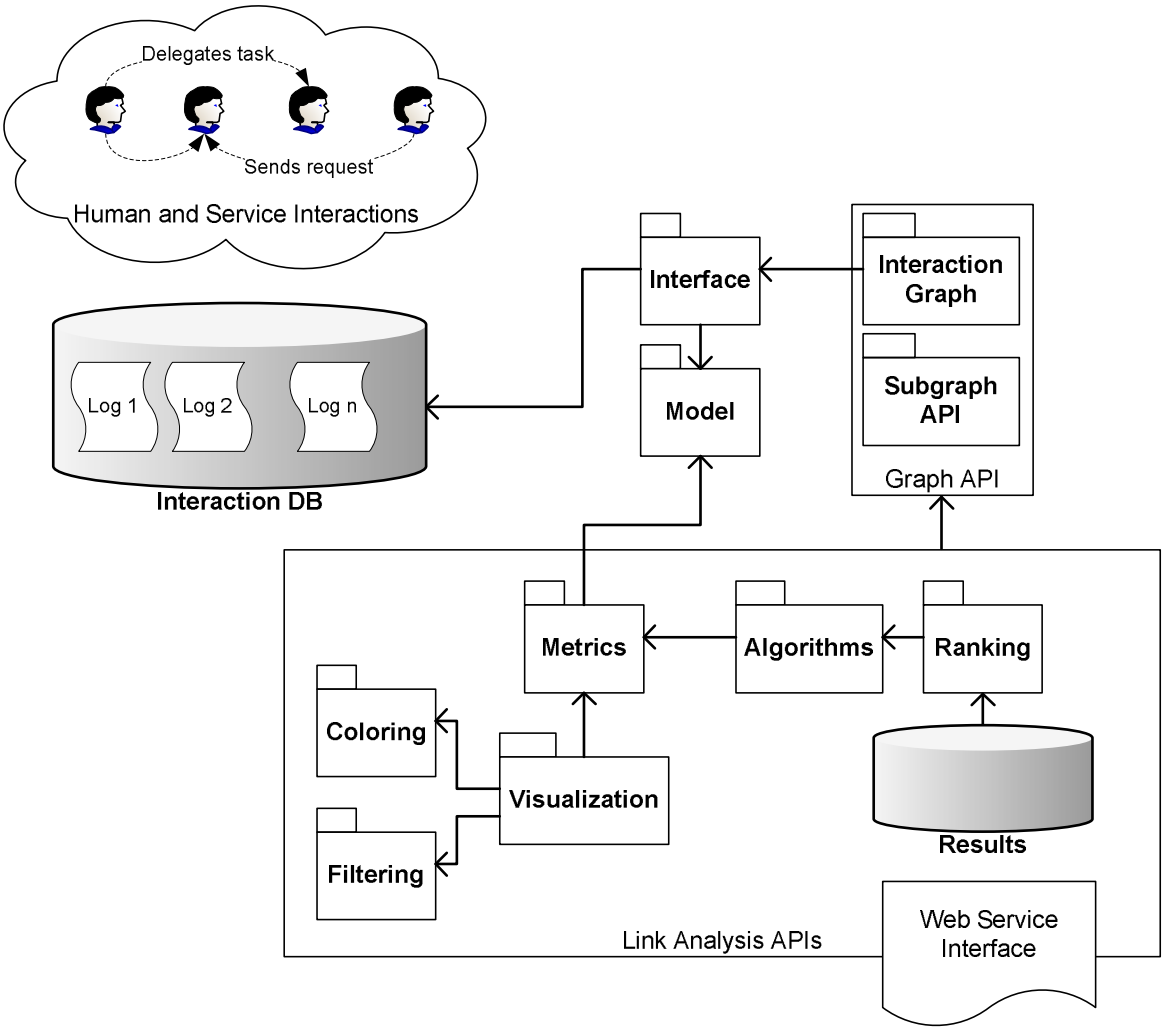


Figure 6.4: Interaction analysis overview.

6.4.2 Implementation Aspects

- `Interaction DB`: At the lowest level, the `db` package contains the facilities to interact with various databases. Each dataset including HPS interactions, human interactions — email graphs, cellphone communications, to name a few — are made available as a *datasource* for analysis.
  - `model`: defining abstractions and concepts in order to map raw logs, interactions, artifact representations in collaborations (examples include coediting of documents, comments in discussion forum), into a basic model accessible to the programmer.
  - `interface`: defining the retrieval API and parameterized queries, selection and filtering, and type information.

  - `querymanager`: manages queries for a given dataset including SQL statements and XQueries.
  - `implementation`: the implementation of a specific datasource. This is usually done by mapping interactions into a *graph* representing the underlying collaborations and interactions. Also, the implementation may contain visualizations specifically designed for a given datasource.

- `Graph API`: There are many different open-source implementations of basic graph APIs in Java. In this work, we use the software libraries provided by the JUNG project[9]. These libraries are open-source and implement the basic facilities for modeling, analysis, and visualization of graphs.

  We extend these API — for graph modeling — by providing features for subgraph decomposition, management, metric computation and subgraph specific operations (e.g., analysis and ranking).

- `Link Analysis API`: We implemented a set of graph algorithms for interaction analysis. To this end, these algorithm are mainly based on *authority* ranking in networks and furthermore offer features to compute graph statistics. Our analysis mainly focuses on context-dependent link analysis. Thus, most experiments focus on *personalizations* (e.g., personalized recommendations) and context-dependent metrics.

  Furthermore, we implemented a set of ranking evaluation metrics including Kendall's τ, relative ranking change, top-k set, and utility metrics. However, visualizations of networks and ranking results are an important tool to understand and interpret results. In addition to above mentioned metrics, which are applicable to most experiments irrespective of the underlying interaction network, we define metrics for specific interactions graphs (i.e., for a given datasource).

6.4.3 Visualization

We have implemented various graph visualization tools to study ranking and analysis algorithms. The implemented extensions on top of JUNG's visualization framework have been defined based on importance ranking, intensity and weighting metrics, and include:

- Graph coloring for vertices and edges using different coloring schemes.
- Shape transformations based on above mentioned metrics to easily detect relations between entities and the importance thereof.
- Clustering and filtering such as subgraph visualizations and top-k set visualization viewers.

[9] Java Universal Network and Graph Framework: `http://jung.sourceforge.net/`

An example instance of a graph is shown in Fig. 6.5. In this particular screenshot, we see labeled links with:

- Each link is associated with multiple context identifiers (tags). The level of detail (i.e., how many of those context identifiers to be displayed) depends on the weight of a context tag. Filtering can be applied to show only the most important context tags. Thus, not only links can be filtered but also the level of detail of displayed context information for each link.
    1. First line: link-context information with context identifier (tag) and associated weight in parenthesis (e.g., 3.2 (0.33)).
    2. Second line: link-context ranked at second place (i.e., the second most important context used in an interaction link) with the same format.
- Additional metrics can be visualized based on the dataset characteristics and/or performed analysis. In this example (third line) we show link affinity.

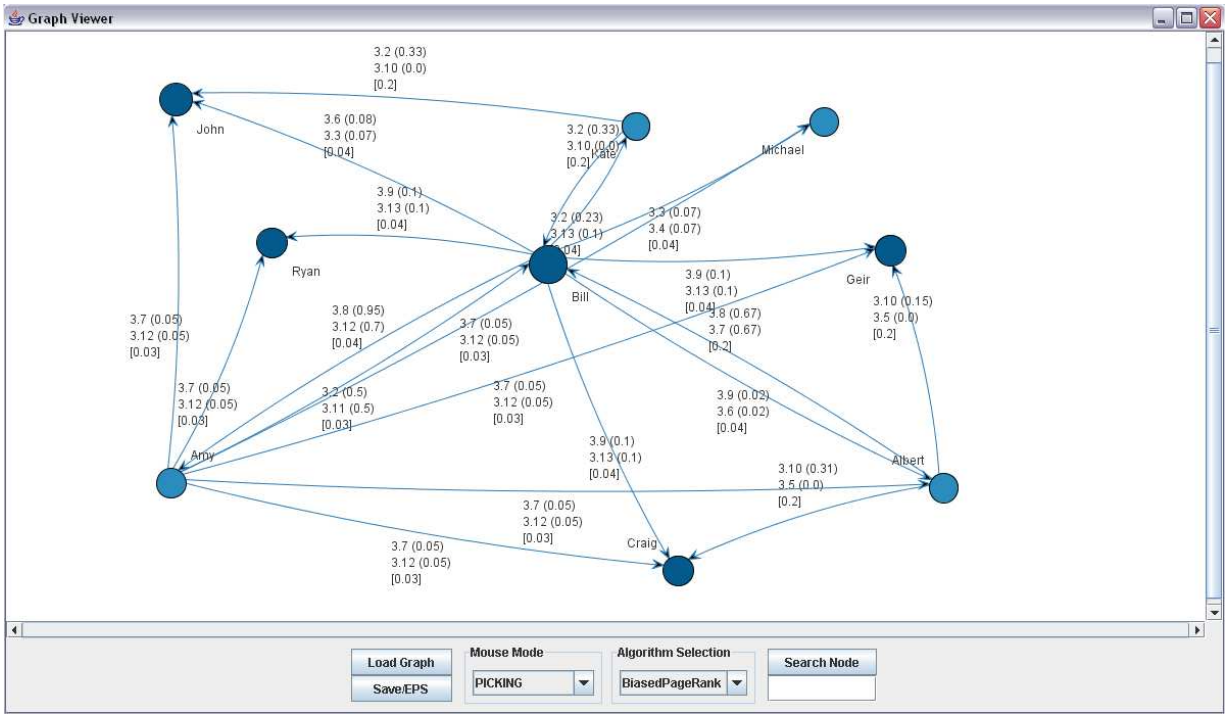


Figure 6.5: Example screenshot of interaction visualization and ranking.

The coloring of vertices and edges is performed by using a sequential, fixed class-size, color mapping. Currently we use 5 different classes, separated by the geometric mean of the distribution of ranking scores. We use a sequential color mapping provided by "ColorBrewer"[10]. Algorithm 6.1 shows the coloring of a set of ranked vertices based on raw ranking scores. Edge coloring is performed in a similar manner by ranking edges based on link metrics, for example, weight or intensity.

Algorithm 6.1 Vertex coloring based on ranking scores.

1: **input**: Vector VR with ranked vertices.
2: **input**: N coloring classes ($|classes|$).
3: **output**: Color info for each vertex in the graph.
4: /* Initialize variables. */
5: $class \leftarrow |classes|$
6: $length \leftarrow |VR|$
7: /* Sort VR by descending rankings. */
8: $sort(VR)$
9: **while** coloring not finished **do**
10: **if** $class = 0$ **then**
11: /* Vertex coloring finished. */
12: Exit loop.
13: **end if**
14: /* We calculate the geometric mean to create N classes of colored vertices. */
15: $mean \leftarrow 0$
16: **for** each vertex $v \in VR$ **do**
17: $mean \leftarrow mean * getRank(v)$
18: **end for**
19: $mean \leftarrow \sqrt[length]{mean}$
20: $index \leftarrow length$
21: **while** threshold not reached **do**
22: **if** $getRank(VR[index]) > mean$ and $class > 1$ **then**
23: /* Ranking threshold using raw scores has been reached. */
24: $length \leftarrow index.$
25: Break loop.
26: **else**
27: $setColorInfo(VR[index], class)$
28: $decrement(index)$
29: **end if**
30: **end while**
31: $decrement(class)$
32: **end while**

[10]ColorBrewer: `http://www.personal.psu.edu/cab38/ColorBrewer/ColorBrewer.html`

Chapter 7

Evaluation

7.1 Abstract

In this chapter, we present the evaluations performed in the course of this thesis. We performed the following experiments:

1. Evaluation of the HPS interaction analysis approach: we provide a detailed analysis using various datasets including tagged-email messages and cellphone communications. We focus on a detailed evaluation and discussion of ranking results using these datasets. This approach is needed because at this stage we were not able to deploy the HPS framework for public use. Therefore, access to a sufficiently large repository of real interactions was only possible using aforementioned sources of human conversations and communications.

2. We demonstrate an HPS use case in open collaboration environments. This use case is implemented on top of the HPS framework showing various artifacts used in collaborations and the steps performed by users (i.e., HPS) and requesters.

3. We provide an overview of mobile Web services toolkits and present a methodology and evaluation of Web services on mobile devices. Understanding whether Web services and related toolkits are supported on such devices was one the first tasks in the research presented in this thesis; e.g., (Schall, Aiello, and Dustdar 2006) and (Schall et al. 2007). Use cases involving mobility aspects are compelling application scenarios for HPS because collaboration in an anytime, anywhere manner is becoming increasingly important.

7.2 Ranking Experiments

First, we establish a set of ranking metrics to test the effectiveness of DSARank compared to PageRank. We provide graph visualizations of ranking results using the intensity-based DSARank as well as the context-aware DSARank, followed by discussions of results based on ranking statistics.

7.2.1 Evaluation Metrics and Comparison of Ranking Algorithms

- **Kendall's τ**: We use the definition of Kendall's τ by Fogaras et al. (2005). Consider the pairs of vertices v, w. The pair is *concordant* if two rankings agree on the order, *disconcordant* if both rankings disagree on the order, *exact-tie* if the exact ranking does not order the pair, and *approximate-tie* if the approximate ranking does not order the pair. Denote the number of these pairs by $c, d, e,$ and a, respectively. Given the total number of pairs as $m = \frac{n(n-1)}{2}, n = |\mathcal{U}|$, then Kendall's $\tau \in [-1, 1]$ is defined as:
$$\tau = \frac{c - d}{\sqrt{(m - e)(m - a)}} \tag{7.1}$$
Kendall's τ helps us to understand whether two algorithms are rank-similar. In other words, if τ equals 1, there are no cases where the pair v, w is ranked in a different order.

- **Relative Ranking Change** (RRC) (Wang et al. 2008): Suppose a user u is ranked at position d by $DSARank$ and at position p by $PageRank$, then u's $RRC \in [-1, 1]$ is given as:
$$RRC(u) = \frac{d - p}{d + p} \tag{7.2}$$
If $RRC(u) < 0$, then u is in the set RRC_p of those users *promoted* by $DSARank$, otherwise *demoted* RRC_d. We calculate the percentage of promoted users as:
$$PP = \frac{|\sum_{u \in RRC_p} RRC(u)|}{|\sum_{u \in RRC_p \cup RRC_d} RRC(u)|} \tag{7.3}$$

- **Top-k Set Metrics**: We define the overlap similarity $OSim(T_{k1}, T_{k2})$ of the top-k sets T_k ranked by $DSARank$ and $PageRank$ as $OSim(T_{k1}, T_{k2}) = \frac{T_{k1} \cap T_{k2}}{k}$ (Haveliwala 2002).

For a small teleportation probability $(1 - \alpha)$, ranking results are robust in terms of τ. Low α (e.g., $\alpha < 0.45$) results in frequent teleportation swamping the resulting ranked network. For all experiments, we use a damping factor of $\alpha = 0.85$.

7.2.2 Effect of Interaction Intensities

Here we compare DSARank and PageRank results using i_{in}, i_{out}, and IIL metrics given the structure and dynamics of the interaction network. We make no assumptions about the nature of underlying interactions, position of people, or the context of a conversation (private or business-related). We entirely focus on the effect of interaction intensities on ranking results. In Fig. 7.1[1], we see the visualization of ranked nodes in a network.

Remark 7.2.1 (Applied Filtering) *We perform filtering of users to select those users with* `indegree` *greater than 1; that is, if a user interacts with at least two different users. In addition, we remove duplicated call information, for example, multiple call records between two users that have the same time interval.*

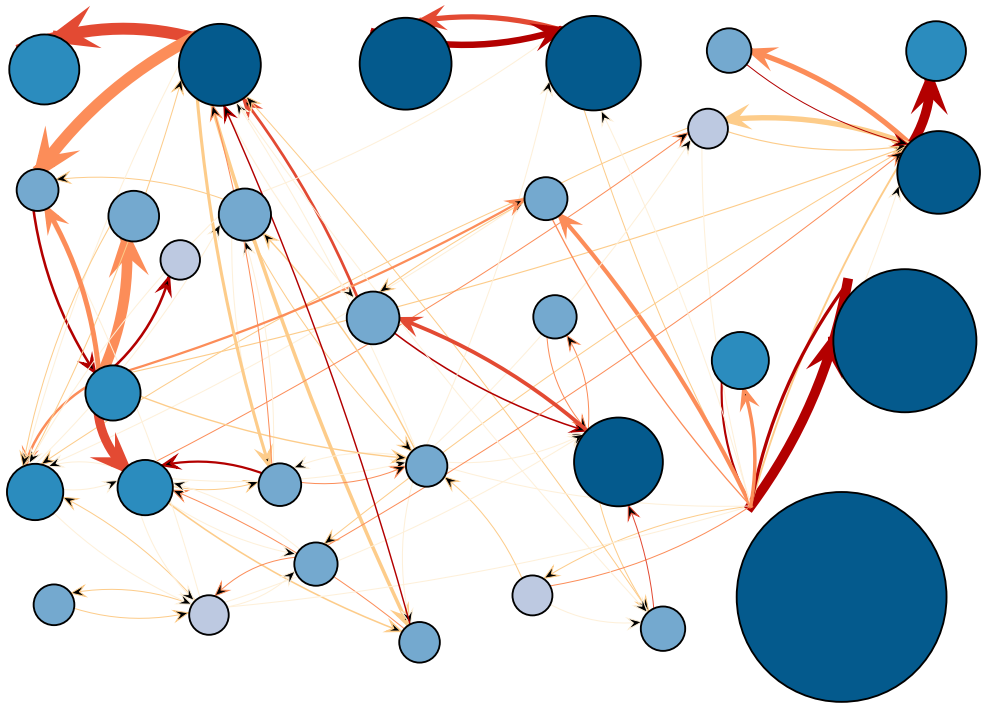


Figure 7.1: (Color online) Top-30 ranked users by DSARank based on selected communications. Users are depicted as nodes in the graph where the size of each circle is proportional to the user's importance (a cut-off is applied to limit the maximum size of a node). Edges are colored based on edge-strengths proportional to interaction intensities (normalized). Personalization is performed by using dynamic properties of the human interaction network. High IIL yields high importance because IIL is a metric for the users' contribution to the information flow. Acting with high intensity (locally) matters for global importance.

[1]Due to graph-visualization reasons, there is a small offset of the tip and the tail of incoming/outgoing edges (the arrow depicting edges) connected to the largest and the second largest node.

The properties of the resulting rankings are that the importance of users is not only influenced by the degree of incoming links but also the intensities of interactions with other (important) users. Interestingly, we find that users having high IIL with important nodes, but low indegrees, still fill top positions in the ranking results. We use equal metrics weights for IIL and `availability`. Figure 7.1 shows the resulting visualization.

In the following we show two tables: Table 7.1 contains $D\vec{S}A$ and $\vec{PR}$ scores and rankings of users based on selected conversations in the mobile phone network. In particular, we select only those calls in *2004-08*. Users are sorted by decreasing DSARank score. Table 7.2 is created in a similar manner; however, users are sorted based on PageRank scores. Also, in both tables i_{out} and i_{in} intensities are percentage values and the metric `availability` is abbreviated as AV.

ID	$score_{D\vec{S}A}$	$rank_{D\vec{S}A}$	$score_{\vec{PR}}$	$rank_{\vec{PR}}$	i_{out}	i_{in}	IIL	AV
43	0.110183	1	0.030705	6	2.74	0.58	0.027978	0.077728
187	0.071380	2	0.004090	90	6.60	8.85	0.110395	0.052550
39	0.050639	3	0.028502	8	1.11	0.62	0.012730	0.032829
313	0.049208	4	0.004369	74	13.21	4.72	0.140244	0.021839
21	0.047523	5	0.029582	13	1.26	0.78	0.014839	0.018402
29	0.043959	6	0.031956	11	5.29	0.72	0.053370	0.090770
83	0.043492	7	0.042618	1	2.71	0.57	0.027711	0.101583
49	0.035641	8	0.003925	100	0.00	10.76	0.107589	0.045642
95	0.027682	9	0.003708	105	0.00	8.57	0.085738	0.048684
50	0.025403	10	0.004090	93	4.38	3.14	0.053894	0.004553

Table 7.1: Top-10 list of users based on DSARank.

Discussion Table 7.1. The first observation in Table 7.1 is that high-IIL with important users increases the position of users dramatically. For example, although the user 187 (in Fig. 7.1, 187 is displayed as the second-largest node located next to the most important user — largest node in the network) is ranked as "unimportant" by the unbiased PageRank (only at position 90 in $\vec{PR}$); user 187 is promoted to the top-10 list due to very high-IIL with a user, whose PageRank is already very high. This is the desired behavior in our ranking model because we expect close collaborators of important users to be important as well; regardless of `indegree`.

Additionally, we see in Table 7.1 two users with $imb(IIL) = 1$ moving up to the top-10 ranked users. Given that the dataset contains partial observations of interactions, it is likely that only a fraction of both users' interactions have been captured. However, as in real collaboration scenarios, it is unlikely that we can capture all interactions of users at all times. We can verify in Fig. 7.1 that ID 49 (located in top-left corner having a single high-weighted, `inlink` from a high-ranked user) and ID 95 (located in top-right corner and similarly connected with a single high-ranked user) are not well connected with the rest of the network, but have i_{in} links from very important users.

ID	$score_{\vec{PR}}$	$rank_{\vec{PR}}$	$score_{D\vec{S}A}$	$rank_{D\vec{S}A}$	i_{out}	i_{in}	IIL	AV
83	0.042618	1	0.043492	7	2.71	0.57	0.027711	0.101583
85	0.037551	2	0.008548	25	0.40	0.08	0.004114	0.009620
8	0.033555	3	0.003750	48	0.08	0.08	0.001195	0.005848
57	0.032718	4	0.020943	15	0.35	0.19	0.004004	0.019368
29	0.031956	5	0.043959	6	5.29	0.72	0.053370	0.090770
43	0.030705	6	0.110183	1	2.74	0.58	0.027978	0.077728
21	0.029582	7	0.047523	5	1.26	0.78	0.014839	0.018402
39	0.028502	8	0.050639	3	1.11	0.62	0.012730	0.032829
20	0.024837	9	0.023832	12	0.71	1.10	0.013113	0.038612
18	0.021950	10	0.009347	24	0.26	0.21	0.003324	0.014028

Table 7.2: Top-10 list of users based on PageRank.

Discussion Table 7.2. We show in Table 7.2 the same collaboration network sorted by $\vec{PR}$ scores. For example, we see that the user with ID 8 was demoted substantially because IIL is very low compared to other top ranked users.

7.2.2.1 Summary of First Observations

To conclude our discussion on these first observations, we believe that DSARank accounting for IIL and `availability` metrics is better suited to recommend users. In our ranking model two facts help to discover important users. On the one hand i_{in} intensities with high-ranked users promote the importance of individuals. For example, in collaborations these users receive much information from knowledgeable people — even if the link-degree of those nodes (users) is low.

On the other hand, high-`indegree` nodes with low IIL are demoted by DSARank. A possible interpretation of this behavior is, for example: if ranking scores are updated within relatively short time frames, say every week or second week, we may discover that individuals are perhaps overloaded in terms of the amount of tasks or requests users have to work on. Thus, users may not interact with high IIL because of their workload that has accumulated over time. Thus, DSARank can prevent individuals from being overloaded by considering the IIL of users. If low IIL is preferred in the ranking model, we could simple use the inverse value of IIL. One might argue that users with few inbound links from a single source, potentially with i_{in}-skewed IIL characteristic like in the interactions of the nodes with ID 49 and 95 (see Figure 7.1); should not be able to improve their importance rankings substantially. We could additionally penalize these cases by varying the IIL parameter β.

In the following we discuss Kendall's τ and ranking changes in the mobile phone network using a 1 month time window to update rankings. We index each month from 1 - 11 by starting at *2004-07*, depicted as period 1, until *2005-05*, depicted as period 11.

7.2.2.2 Kendall's τ

Table 7.3 shows the comparison of DSARank and the unbiased PageRank with $p(u) = 1$ and `outdegree`-based edge weights; and also PageRank in the weighted network using intensity-based weights.

Period	1	2	3	4	5	6	7	8	9	10	11
τ (unbiased)	0.35	0.34	0.51	0.54	0.48	0.51	0.44	0.43	0.33	0.26	0.36
τ (weighted)	0.63	0.61	0.82	0.87	0.82	0.81	0.74	0.70	0.65	0.68	0.51

Table 7.3: Comparison DSARank and PageRank showing Kendall's τ in different periods.

The first comparison (`unbiased`) shows that there is no strong disagreement between $\vec{DSA}$ and $\vec{PR}$. Otherwise, we would violate our initial assumption that PageRank applied to human interaction analysis is suitable to determine the importance of users. A stronger agreement in rankings is achieved by using i_{out} intensity based link weights (comparison `weighted`).

7.2.2.3 Relative Ranking Change

The next step in our evaluation is to determine for which users we observe ranking changes. We measure whether users get *promoted* or *demoted* given the users' `availability` and IIL. We calculate the range for both metrics as $max($`availability`$) - min($`availability`$)$ and $max(IIL) - min(IIL)$, respectively. For both metrics, we segment the range linearly into a number of buckets. Then, we calculate RRC and PP for each bucket to show how many users given `availability` and IIL were promoted; that is $RRC < 0$. We compare the relative ranking change by using DSARank and the unbiased PageRank. Over the entire period (period 1 - 11) we observer that on average the number of promoted users PP equals 0.46 for both metrics. In other words, given the total number of users in a certain period, on average a fraction of 0.46% were promoted and 0.54% demoted.

In Fig. 7.2, we show two sets of figures delimited by the horizontal line: the top set shows RRC for `availability` and the bottom set of figures RRC for IIL. Each set has 11 sub-figures, which depict the ranking changes for each metric in periods ranging from for 1 to 11.

Given a set of figures (for a specific metric), we show ranking changes for period 1 in the top-left sub-figure, continue with the second sub-figure in the same row to denote period 2 and continue in this manner (left to right and top to bottom) until period 11. Typically, 5 buckets are shown at the horizontal axis. However, empty buckets are not shown.

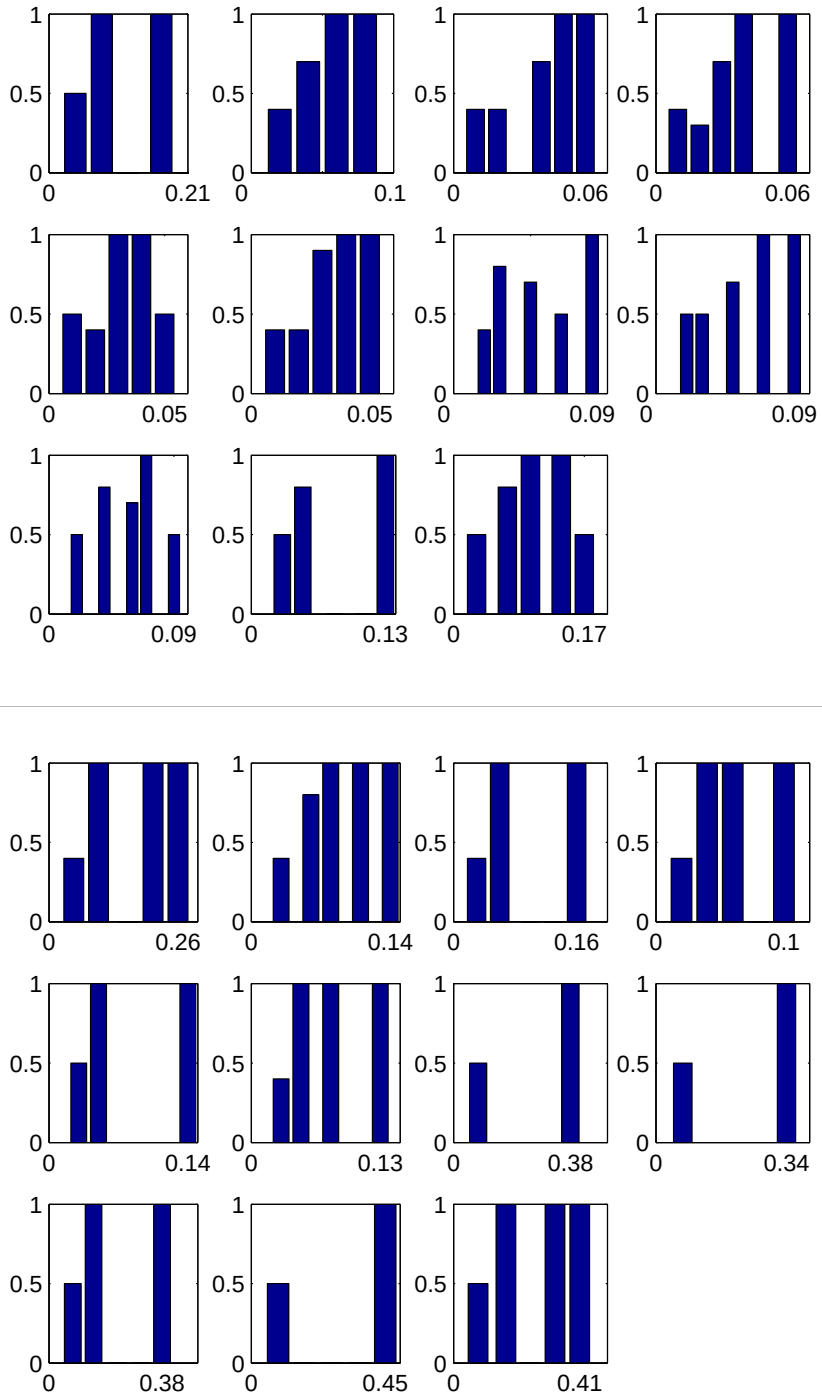


Figure 7.2: *RRC* and *PP* measured for `availability` (top figures) and *IIL* (bottom figures) delimited by horizontal line.

Discussion `availability`. In most cases we see regularities in promoting users with increasing availabilities. A 10% rule applies: 90% of the users in RRC_p are in the first bucket (lower availability segment), while the remaining 10% are distributed across the other buckets. Interaction intensities have a strong impact on link-weights and thus PP in general. If both metrics are equally weighted; intensities dominate promotion of users, which we confirm in the IIL-related sub-figures.

Discussion IIL. A similar distribution rule of users across buckets applies. All users having high IIL were promoted but never demoted. Also, we see an upper threshold of $PP = 0.5$ in the lower IIL segment (e.g., first bucket). Users with low IIL are more likely demoted than promoted.

7.2.2.4 Overlap Similarities

In Table 7.4, we show $OSim$ in various top-k sets of obtained rankings. $OSim$ of DSARank and the unbiased PageRank are denoted as *nobias*.

$D\vec{S}A, \vec{PR}$	$k = 10$ (nobias)	$k = 10$ (weighted)	$k = 30$ (nobias)	$k = 30$ (weighted)	$k = 50$ (nobias)	$k = 50$ (weighted)
avg	0.38	0.65	0.55	0.81	0.67	0.85
σ	0.08	0.13	0.06	0.05	0.11	0.05

Table 7.4: $OSim$ DSARank versus PageRank: avg is the average overlap similarity (period 1 - 11) and σ the standard deviation of overlap similarities.

Table 7.4 confirms our initial assumption of not having strong disagreements between DSARank and the unbiased PageRank. On average, we see an overlap of $OSim = 0.3$ in the top-10 segment compared to the unbiased PageRank.

7.2.2.5 Summary

To summarize our evaluation on the role of intensity and availability metrics: `availability` is coupled with IIL and not always dominant in the upper availability segments. For IIL, we see clear regularities. Users with high-IIL are always promoted and, on the contrary, low-IIL more likely demotes users' importance. We believe that this behavior covers the requirements of real-life collaboration environments: availability itself does not guarantee promotion of users because individuals need to be active players and involved in interactions and collaborations.

7.2.3 Experiments in Labeled Interaction Graph

In this section we specifically focus on the proposal of ranking users in context-based interactions. For this purpose, we use the Enron email dataset[2].

[2]Enron email: `http://bailando.sims.berkeley.edu/enron_email.html`

7.2.3.1 Tagged Message Corpus

A subset of messages of the entire message corpus were labeled by UC Berkeley's Natural Language Processing group. These tags were applied to about 1700 messages. The tagged message corpus serves well to test our context-based ranking approach. Different categories of tags were applied to interaction links comprised of messages between people with the focus on business-related emails. 13 categories or *contexts* are available (Table 7.5).

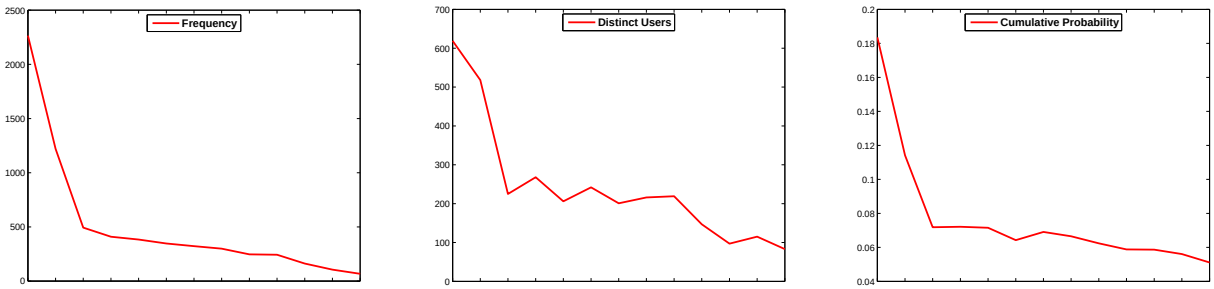


Figure 7.3: Tag-statistics in email interactions: left figure shows the frequency of tags given as the total number of occurrences; middle figure depicts the distinct number of users participating in an interaction context (i.e., if one of the user's interaction links contains the tag); right figure shows the cumulative smoothened probability of tags.

ID	Index	Description
3.1	2	Regulations and regulators (includes price caps)
3.2	5	Internal projects – progress and strategy
3.3	3	Company image – current
3.4	6	Company image – changing / influencing
3.5	10	Political influence / contributions / contacts
3.6	1	California energy crisis / California politics
3.7	7	Internal company policy
3.8	9	Internal company operations
3.9	11	Alliances / partnerships
3.10	4	Legal advice
3.11	13	Talking points
3.12	8	Meeting minutes
3.13	12	Trip reports

Table 7.5: Primary categories in labeled interaction graph. The index establishes the correspondence to the tag-statistics in Fig. 7.3 (horizontal axis).

7.2.3.2 Applied Expansion and Filtering

We expand the subset of labeled messages by searching the entire email-message corpus for related messages. For example, given a labeled message, we search for messages which are most likely related to the labeled message; for example, in `reply-to` or `forwarded` messages. Thereby, we expand the existing labeled message corpus by adding 5248 related messages. However, some messages are simply "broadcast" messages (e.g., announcements or periodic auto-generated messages from a person), which we filter out because these messages might distort ranking results. In addition, sender and recipient of messages must be real people (e.g., we filter messages from — and to — distribution lists) because we are interested in link-based importance rankings of people.

7.2.3.3 Ranking Parameters

In all experiments presented in this section, we set the IIL parameter β to 1.2; therefore assigning a bias to out-intensities. In our previous experiments, we did not use an imb threshold (imb denoting the imbalance of interactions). Here we use a filter of $-0.9 < imb(IIL) < 0.9$. If $imb(IIL)$ of a user is not within this range, we "downgrade" the user's $IIL(u; c)$ to 0. This is motivated by the following reason: phone calls are synchronous, thereby guaranteeing an information flow between users. Whereas email links between users might be irrelevant if IIL is strongly imbalanced. For example, a user who is active in a given context, but never receives a reply ($\texttt{indegree}(u) = 0$ as well as $imb(IIL) = -1$). In addition, we will focus on the impact of IIL and SE, which we equally weight, without parameterizing DSARank with `availability`.

7.2.3.4 Context Coupling and Subgraph Intensities

Figure 7.4 shows whether different interaction contexts have many shared (i.e., overlapping) links. In other words, interaction links may contain tags that belong to different contexts. Therefore, we speak of an overlapping link $\ell(C')$ between c_1 and c_2, if $\{c_1, c_2\} \subseteq C'$.

ID	3.9	3.6	3.2	3.10	3.1	3.3	3.4	3.7	3.8	3.12	3.5	3.11	3.13
$i(g)$	7.43	6.03	5.98	4.54	4.42	4.04	3.96	3.10	3.06	2.83	2.12	1.81	1.06

Table 7.6: Intensities $i(g)$ for different subgraphs ($10^3 \times i(g)$).

We connect two categories if links are annotated with the same context tags. The node size and coloring scheme is based on subgraph intensities as depicted in Table 7.6. The context (category) 3.9 constitutes the subgraph with the highest intensity, whereas 3.6 has the highest degree of shared links.

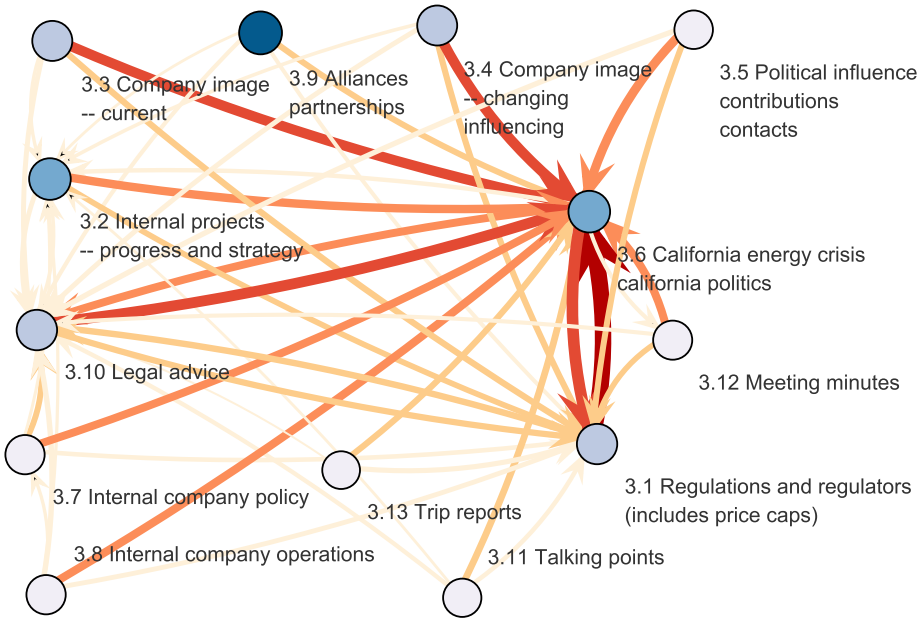


Figure 7.4: (Color online) Visualization of shared links between different contexts.

7.2.3.5 Filtering Algorithm

In the following experiments, the visualization of ranking results and interactions between users are filtered using the following algorithm:

1. We create two sets T_{k1} and T_{k2} of top-k ranked users.
2. If $\texttt{rank}(u) \leq k1$, we add u to T_{k1}, otherwise to T_{k2} if $\texttt{rank}(u) \leq k2$.
3. We remove all users $u \in g$ which are not in $T_{k1} \cup T_{k2}$.
4. For each user in T_{k2} we demand a minimum degree $\texttt{min}_{k=1}$ of connectedness to T_{k1} users. We remove $u \in T_{k2}$, if u is not connected to at least $\texttt{min}_{k=1}$ users.
5. For each user in T_{k1}, we test whether $u \in T_{k1}$ is connected to at least $\texttt{min}_{k=1\cup 2}$ users in $T_{k1} \cup T_{k2}$.

By using the above algorithm, we ensure that all users in the visualized graph are connected to a minimum number of top-ranked users.

7.2.3.6 Applying DSARank in Context-Dependent Interactions

As a first example, we select the subgraph for 3.6 (i.e., the specific interaction context) and rank all users. The detailed results are provided in Table 7.7. Figure 7.5 shows the visualization of interactions of top-ranked users (according to the filtering algorithm). We set $\texttt{min}_{k=1} = 2$, $\texttt{min}_{k=1\cup 2} = 2$, $|T_{k1}| = 5$, and $|T_{k2}| = 15$. With these parameters we have a reasonable number of interactions and users in our graph visualization.

In Fig. 7.5, we see similar characteristics in terms of importance rankings as we observed previously in rankings based on mobile phone calls (i.e., Fig. 7.1). Not only many incoming interactions with important users matter, intensity in a given context dependent subgraph g plays a keyrole. It is well possible that users get promoted (in some cases substantially) because they interact with important users in a given context with high intensity. Thus, DSARank provides accurate results as users are not ranked in a single context.

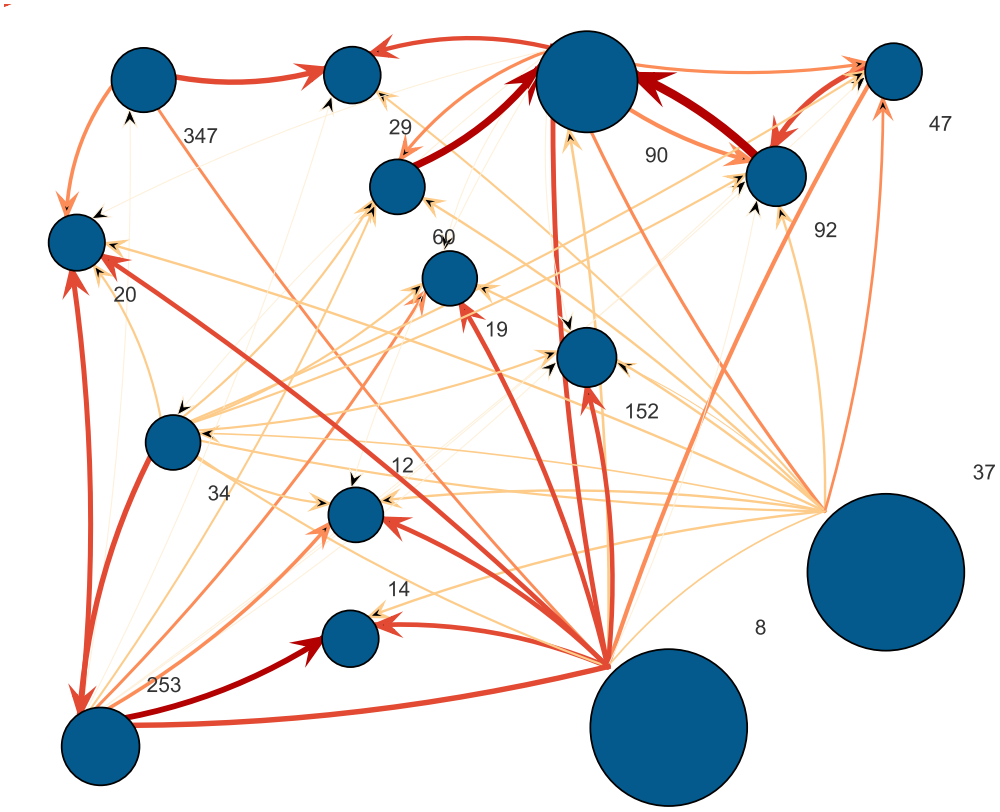


Figure 7.5: (Color online) Example of context-aware DSARank: we select context 3.6 (California energy crisis / politics) and perform ranking. The subgraph g comprises 11839 messages, 1852 links, and 469 users. We use two metrics, IIL with bias $\beta = 1.2$ and calculate SE within g. Both metrics are weighted with 0.5.

ID	$score_{\vec{DSA}}$	$rank_{\vec{DSA}}$	$score_{\vec{PR}}$	$rank_{\vec{PR}}$	IIL	imb
37	0.109475	1	0.004121	21	7.31	-0.81
8	0.102375	2	0.020758	1	5.13	0.11
90	0.043071	3	0.008326	9	1.1	0.08
253	0.029936	4	0.001733	170	2.07	-0.85
347	0.020443	5	0.001665	282	1.39	-0.87
92	0.016677	6	0.003912	23	0.39	0.82
152	0.016375	7	0.013148	2	1.16	1.0
47	0.014248	8	0.003593	27	0.66	0.41
29	0.014195	9	0.005415	16	1.14	1.0
14	0.014084	10	0.010911	4	2.27	1.0

Table 7.7: Top-10 ranked users by DSARank. Selected subgraph corresponds to category 3.6. Users are sorted by decreasing DSARank score.

7.2.3.7 Kendall's τ

In the next step we compare Kendall's τ of DSARank when compared to PageRank. In particular, we rank in different subgraphs and combine results using the formula for the context-aware DSARank (see Equation 4.15) to create composite DSARank scores. Each context-dependent result vector $w_{c1} * \vec{DSA}(c_1)$ and $w_{c2} * \vec{DSA}(c_2)$ is combined with $w_{c1} = w_{c2}$.

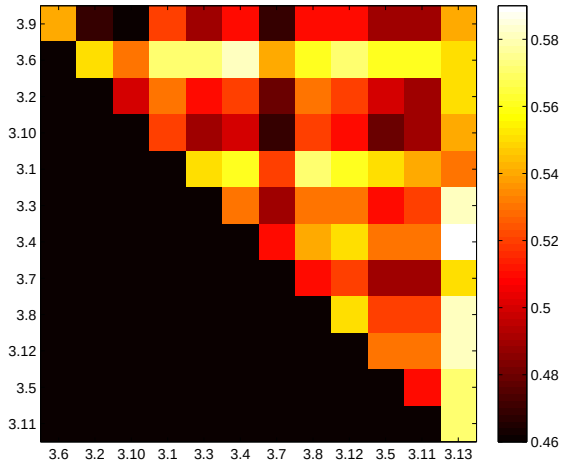


Figure 7.6: Kendall's τ for composite contexts corresponding to entries in Table 7.8.

To create PageRank scores, we use the entire interaction graph $\mathcal{G}$ to create the vector $\vec{PR}$. Kendall's τ for different combinations of contexts is shown in Table 7.8. We created Fig. 7.6 based on the data in Table 7.8; making it easier to see whether there is a strong disagreement in terms of τ in different contexts.

	3.6	3.2	3.10	3.1	3.3	3.4	3.7	3.8	3.12	3.5	3.11	3.13
3.9	0.54	0.47	0.46	0.52	0.49	0.51	0.47	0.51	0.51	0.49	0.49	0.54
3.6		0.55	0.53	0.57	0.57	0.58	0.54	0.56	0.57	0.56	0.56	0.55
3.2			0.5	0.53	0.51	0.52	0.48	0.53	0.52	0.5	0.49	0.55
3.10				0.52	0.49	0.5	0.47	0.52	0.51	0.48	0.49	0.54
3.1					0.55	0.56	0.52	0.57	0.56	0.55	0.54	0.53
3.3						0.53	0.49	0.53	0.53	0.51	0.52	0.58
3.4							0.51	0.54	0.55	0.53	0.53	0.59
3.7								0.51	0.52	0.49	0.49	0.55
3.8									0.55	0.52	0.52	0.58
3.12										0.53	0.53	0.58
3.5											0.51	0.57
3.11												0.57

Table 7.8: Kendall's τ for comparison of PageRank and DSARank in composite contexts.

Discussion Table 7.8. Contexts with many shared links (for example 3.1 and 3.6 as depicted in Fig. 7.4) yield stronger agreements between DSARank and PageRank. Intuitively, 3.6 and 3.1 become more dominant when combined with other contexts. It is therefore less likely that the order of rankings change. On the other hand, if a context, for example 3.13 has few shared links with other contexts; and also low subgraph intensity (3.13 has the lowest subgraph intensity), then we observe also stronger agreements in rankings. This can be explained as the limited impact of low intensity contexts on changing the position of users within ranking results.

7.2.3.8 Overlap Similarities

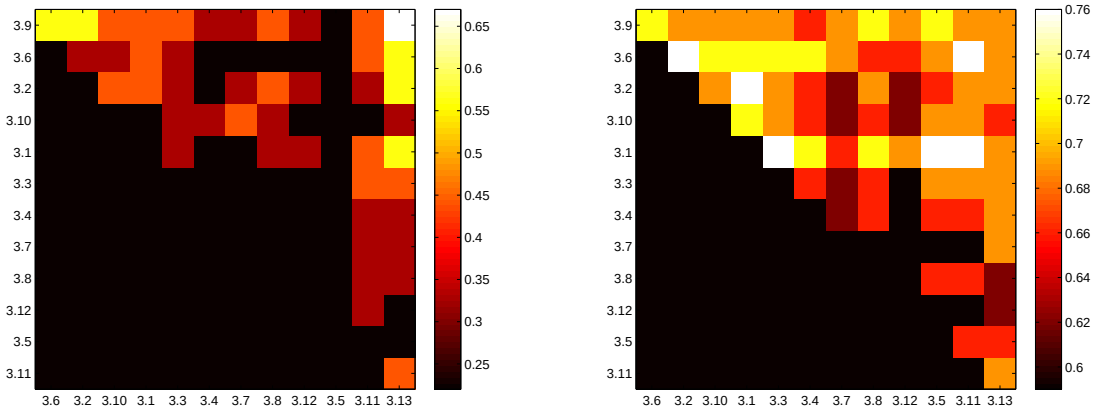


Figure 7.7: Overlap similarities for composite contexts: left figure shows $OSim_{k=10}$ and right figure $OSim_{k=30}$.

Here we compare DSARank and PageRank in terms of overlap similarities. Table 7.9 contains the results for $OSim_{k=10}$ and Table 7.10 the results for $OSim_{k=30}$. In Fig. 7.7 we show the visualizations of the results in both tables. By comparing the top-10 segment of ranked users (Fig. 7.7 left), we see higher overlap similarities between high-intensity contexts, for example, the context pairs (3.9, 3.6), (3.9, 3.2). Low intensity contexts such as 3.13 combined with, for example (3.13, 3.9), yields also high similarities. The top-30 segment (Fig. 7.7 right) shows stronger similarities in 3.1 as well as 3.6 — both contexts have many shared links with other contexts.

	3.6	3.2	3.10	3.1	3.3	3.4	3.7	3.8	3.12	3.5	3.11	3.13
3.9	0.56	0.56	0.44	0.44	0.44	0.33	0.33	0.44	0.33	0.22	0.44	0.67
3.6		0.33	0.33	0.44	0.33	0.22	0.22	0.22	0.22	0.22	0.44	0.56
3.2			0.44	0.44	0.33	0.22	0.33	0.44	0.33	0.22	0.33	0.56
3.10				0.22	0.33	0.33	0.44	0.33	0.22	0.22	0.22	0.33
3.1					0.33	0.22	0.22	0.33	0.33	0.22	0.44	0.56
3.3						0.22	0.22	0.22	0.22	0.22	0.44	0.44
3.4							0.22	0.22	0.22	0.22	0.33	0.33
3.7								0.22	0.22	0.22	0.33	0.33
3.8									0.22	0.22	0.33	0.33
3.12										0.22	0.33	0.22
3.5											0.22	0.22
3.11												0.44

Table 7.9: $OSim_{k=10}$ for composite contexts.

	3.6	3.2	3.10	3.1	3.3	3.4	3.7	3.8	3.12	3.5	3.11	3.13
3.9	0.72	0.69	0.69	0.69	0.69	0.66	0.69	0.72	0.69	0.72	0.69	0.69
3.6		0.76	0.72	0.72	0.72	0.72	0.69	0.66	0.66	0.69	0.76	0.69
3.2			0.69	0.76	0.69	0.66	0.62	0.69	0.62	0.66	0.69	0.69
3.10				0.72	0.69	0.66	0.62	0.66	0.62	0.69	0.69	0.66
3.1					0.76	0.72	0.66	0.72	0.69	0.76	0.76	0.69
3.3						0.66	0.62	0.66	0.59	0.69	0.69	0.69
3.4							0.62	0.66	0.59	0.66	0.66	0.69
3.7								0.59	0.59	0.59	0.59	0.69
3.8									0.59	0.66	0.66	0.62
3.12										0.59	0.59	0.62
3.5											0.66	0.66
3.11												0.69

Table 7.10: $OSim_{k=30}$ for composite contexts.

7.2.3.9 Skill and Expertise Rank in Subgraphs

We implemented the algorithm to compute SE as a variant of the iterative PageRank algorithm (Jaccobi iteration introduced in Algorithm 4.1). However, SE is not personalized by adding teleport vectors $\vec{p}$. In other words, we do not add $(1-\alpha)\vec{p}$ in each iteration. In addition, we perform a *fixed number* of iterations to compute SE for each subgraph g to keep computational complexity low. In our experiments, we perform 6 iterations in each subgraph. When comparing the ranking results of SE and the unbiased PageRank, we observed that even though $\vec{p}$ was not used in the computation, both vectors $\vec{SE}$ and $\vec{PR}$ were rank-similar with Kendall's τ approximately equal to 1.

7.2.4 Score Distributions of Ranking Algorithms

To conclude the evaluation of interaction analysis algorithms developed within the HPS framework, we discuss the characteristics of ranking-score distributions. The presented results are based on the mobile phone dataset. We show ranking scores by using interaction information over the entire period comprising about 445254 aggregated calls.

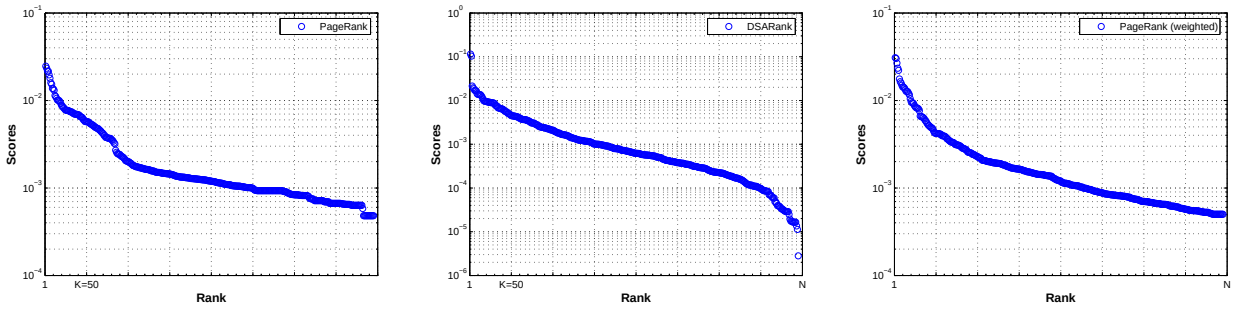


Figure 7.8: Distributions of ranking scores: the horizontal axis shows the rank of a person and the vertical axis the ranking score on a log-scale. Left figure: PageRank scores not using weighted edges and $p(u) = 1$ achieving a precision of 7.3721E-9 after 29 iterations. Middle: intensity-based DSARank in weighted interaction graph. The number of iterations was 51 achieving a precision of 2.1727E-8. Right figure: PageRank using edge weights; convergence after 51 iterations with a of precision 1.6428E-8.

7.3 HPS in Open Collaboration Environments

In this section we discuss an HPS use case scenario to give a summary of introduced concepts. This scenario illustrates an ad-hoc collaboration between humans. The essential steps are depicted by Fig. 7.9, and detail the role of the HPS framework in each step. XML listings and namespaces are abbreviated (depicted as: ...) for readability.

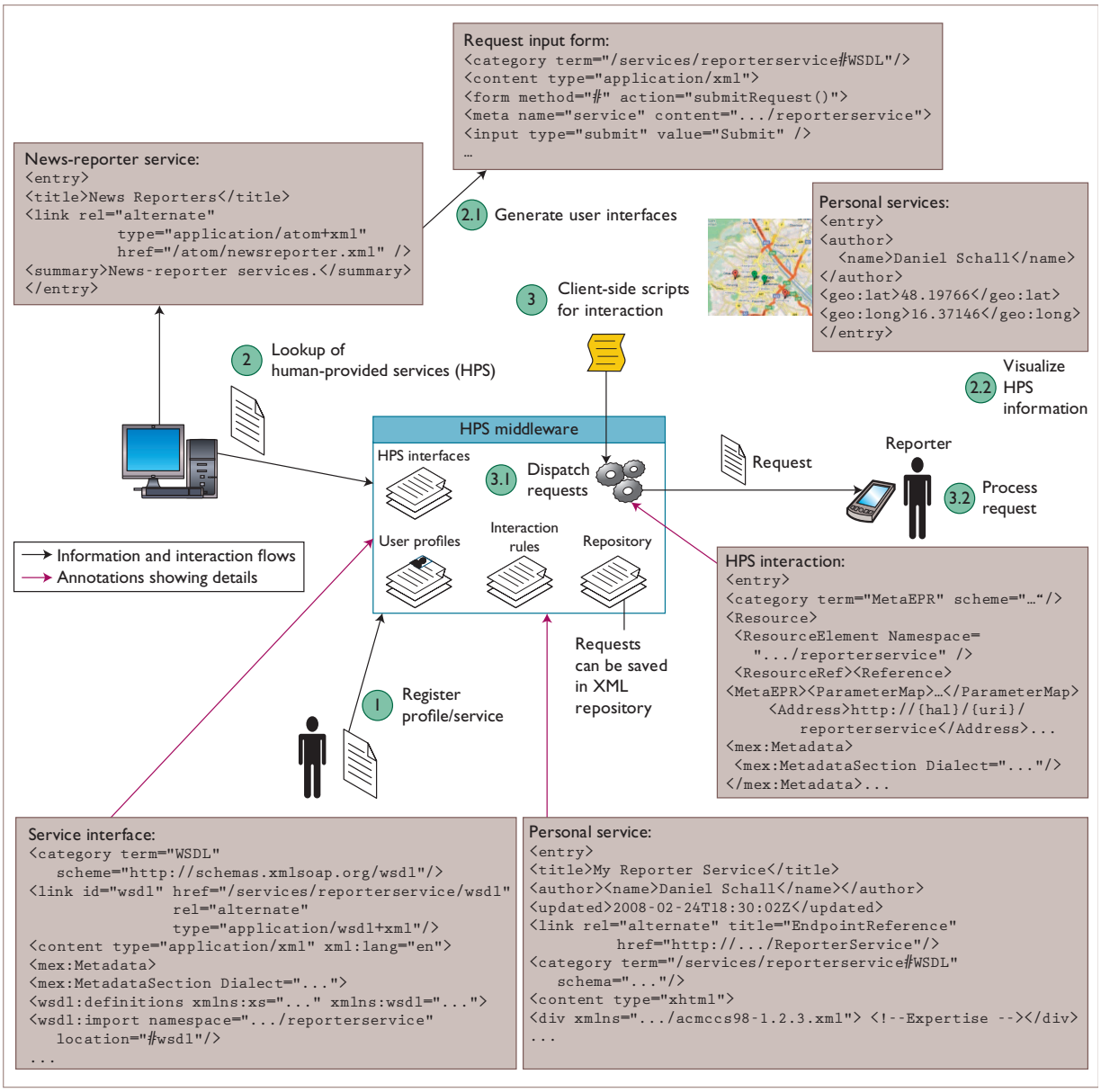


Figure 7.9: Example application scenario of HPS in human collaborations: the "reporter service" is provided by multiple users and can be dynamically discovered based on the location and availability context.

1. **Register the profile and service:** Human actors define high-level collaboration activities (for example, `createReport`) using an HPS interface editor that the framework hosts. The HPS framework automatically translates these activities into low-level service interfaces described in WSDL. User profile information includes name, skills, and competency, which the HPS framework uses to enhance the discovery, selection, and recommendation process to find the most suitable HPS. The user specifies basic personal profile information or uploads this information as a vCard file. Humans provide a service by registering it as a personal service. The scenario in Fig. 7.9 shows an example in which humans provide reporter services to contribute Web content such as news reports. The middleware hosts a set of XML documents in the service registry that is managing the interface description and personal service information.

```
<category term="WSDL" scheme="http://schemas.xmlsoap.org/wsdl" />
<link id="wsdl" href="/services/reporterservice/wsdl"
      rel="alternate" type="application/wsdl+xml" />
<content type="application/xml">
 <mex:Metadata>
 <mex:MetadataSection Dialect="...">
 <wsdl:definitions xmlns:xs="..." xmlns:wsdl="...">
 <wsdl:import namespace=".../reporterservice" location="#wsdl" />
...
```

Listing 7.1: Excerpt service interface.

Therefore, it is easier to achieve cross-organizational collaboration because companies can share information stored in the service registry; the very foundation to achieve dynamic collaborations in open service-oriented environments. Other people who want to provide the same type of service can then reuse the service interfaces.

```
<entry>
 <title>My Reporter Service</title>
 <author><name>Daniel Schall</name></author>
 <updated>2008-02-24T18:30:02Z</updated>
 <link rel="alternate" title="EndpointReference"
       href="http://.../ReporterService" />
 <category term="/services/reporterservice#WSDL" schema="..." />
 <content type="xhtml">
  <div xmlns=".../acmccs98-1.2.3.xml"><!-- Expertise --></div>
...
```

Listing 7.2: Personal service description.

Listing 7.2 shows an excerpt of the XML description of a personal service. The description contains user-related information, a reference to the service interface description, and information regarding the user's expertise rooted in taxonomies. This information is embedded in Atom feed entries. The Atom Syndication Format is an XML language describing frequently updated content such as news. Atom feeds

contain, for example, author information, links to content, and summaries. The framework uses Atom feeds as a container format for WSDL documents and various content including taxonomies describing users' expertise; additional context information, such as location (`<geo>` tags); and category information to tag services.

The HPS framework supplies the personal service hosting environment, which users can download to their desktop computer or mobile devices using mobile Java technology (JavaME). This environment lets the computer or device deploy software for personal services as gadgets. It comprises a micro OSGi environment, a set of tools to manage the gadgets (services), a common lightweight SOAP library, and a user-interface rendering engine displaying user interfaces described in XML.

2. **Look up a service:** HPSs can be discovered through an interface implementing the Atom protocol model or a Web service interface. We show an example in which location and availability information enhance the discovery process given that requesters might want to find reporter services located in some area of interest.

```
<entry>
  <title>News Reporters</title>
  <link rel="alternate" type="application/atom+xml"
        href="/atom/newsreporter.xml" />
  <summary>News-reporter services.</summary>
</entry>
```

Listing 7.3: Service entry in lookup result.

The Atom-based lookup interface returns a feed containing a ranked list of entries (Listing 7.3) comprising personal HPS information. It ranks the services based on various HPS metrics, such as skill level and user response time.

The lookup returns additional user interface rendering information. On the one hand, XForms, which are automatically generated based on WSDL interfaces (step 2.1), can be used by human requesters to interact with HPSs. (XForms are a forms technology expressed in XML that describe user interfaces in a device-independent way.) If XForms are not supported by the client, HTML forms can be used to insert request data; see Listing 7.4 for an example. HTML forms need to be created manually (i.e., these forms cannot be generated based on WSDLs).

```
<category term="/services/reporterservice#WSDL" />
<content type="application/xml">
 <form method="#" action="submitRequest()">
 <meta name="service" content=".../reporterservice">
 <input type="submit" value="Submit" />
...
```

Listing 7.4: Request input form.

Both XML or HTML forms can be embedded in markers of a geo map.

3. **Interact with HPSs:** AJAX scripts can issue requests asynchronously towards the middleware platform. The middleware implements HAL to dispatch HPS requests. HAL provides a security module to prevent unauthorized access, policy management to protect the users' privacy, and request filtering to shield HPSs from denial of service attacks. HAL dispatches and routes service requests to the appropriate HPS and device.

```
<entry>
 <category term="MetaEPR" scheme="...resourceCatalog#MetaEPRType" />
 <content type="application/xml" xml:base="...resourceCatalog">
 <Resource>
  <ResourceElement Namespace="...reviewservice"
   LocalName="ReviewService" />
   <ResourceRef><Reference>
    <MetaEPR>
     <ParameterMap>...</ParameterMap>
     <Address>http://{HAL}/{URI}/reporterservice</Address>
    </MetaEPR>
    <mex:Metadata>
     <mex:MetadataSection Dialect="..." />
    </mex:Metadata>
   </Reference>
  </ResourceRef>
 </Resource>
 </content>
</entry>
```

Listing 7.5: HAL definition used for HPS interactions.

The HAL interface description is denoted as HPS interaction (Listing 7.5) using Web Services Resource Catalog (WS-RC[3]) Meta-Endpoint definitions that are parameterized by HPS addressing information, such as user identifiers.

HPSs are not always online, because the personal service hosting environment might be deployed on mobile devices, which rely on wireless network availability and coverage. If the HPS is not available at the time of interaction, an XML-based repository stores service requests and process them whenever the HPS is back online (step 3.2). Pending requests can be received via push- and pull-based mechanisms depending on the hosting environment's configuration. At this stage, HAL comprises request processing and routing capabilities and request filtering.

[3]WS-RC: `http://schemas.xmlsoap.org/ws/2007/05/resourceCatalog/`

7.4 Web Services on Mobile Devices

In this section, we provide an overview of Web services toolkits that can be used on mobile devices such as cellphones or PDAs. We focus on Java-based implementations suitable for JavaME (Java Mobile Edition) enabled devices and a C++ Web services stack. We present a simple approach to test and compare the performance of Java/C++ based implementations. Furthermore, we selected two open source toolkits to evaluate the performance of Web services on mobile devices. Mobility aspects in HPS-based applications are important to enable pervasive interactions and information access. Thus, it is critical to understand the features and limits of various Web services stacks.

7.4.1 Web Services Toolkits

We have two technology choices; *platform specific* and *platform independent* implementations. Here we give a short overview of various technologies, the basic platform characteristics, and architecture of a C++ based Web services stack.

7.4.1.1 Platform Specific Implementations

- The Java 2 Platform Micro Edition (J2ME) platform is a set of standard Java APIs defined through the Java Community Process (JCP). The J2ME specifications define the Connected Device Configuration (CDC) (i.e., a subset of J2SE) and the Connected Limited Device Configuration (CLDC). In contrast to CDC, CLDC provides libraries such as the *Connection Framework* which are suitable for devices with a small memory footprint. These libraries are not part of J2SE. CLDC targets hardware platforms with 128 KB to 512 KB memory and 16-bit or 32-bit CPUs. The Mobile Information Device Profile (MIDP) is specifically designed for cell phones and provides the user interface, network connectivity, local data storage, and application management needed by these devices. The following SOAP APIs and Web services toolkits are suitable for J2ME/MIDP based devices.

    1. kSOAP[4] is an open source SOAP API for J2ME devices. It provides a lightweight approach to access SOAP based Web services. However, kSOAP cannot generate client side stubs from a Web service's WSDL interface.
    2. JSR-172 is a set of Web service APIs (WSA) for J2ME[5] available in Sun's wireless toolkit (WTK) 2.2. In contrast to kSOAP, client side stubs can be generated by using WSDL files, thus accelerating the development process.

- The .NET Compact Framework (CF) is a subset of Microsoft's .NET framework. The .NET CF is supported on various devices/platforms that are based on PocketPC

[4]kSOAP: `http://kobjects.sourceforge.net`
[5]J2ME Web Services: `http://java.sun.com/j2me/reference`

and Smartphone architectures. Web services on .NET CF support synchronous or asynchronous invocations. Development of embedded Web services is performed in the same manner as in .NET. A *Web Reference* (i.e., a reference to the actual service) has to be added to a project and the code is automatically generated.

7.4.1.2 Platform Independent Implementations

The gSOAP toolkit is a platform independent C/C++ based Web services stack (Engelen 2004). We show the basic architecture of the gSOAP client in Fig. 7.10 comprising the *development* and *deployment* phase. The development process starts with C/C++ header file creation based on the service's WSDL file. Next, the gSOAP compiler is used to create the code files. The actual run-time interactions are shown as `Client Request` and `Service Response` (lower right corner).

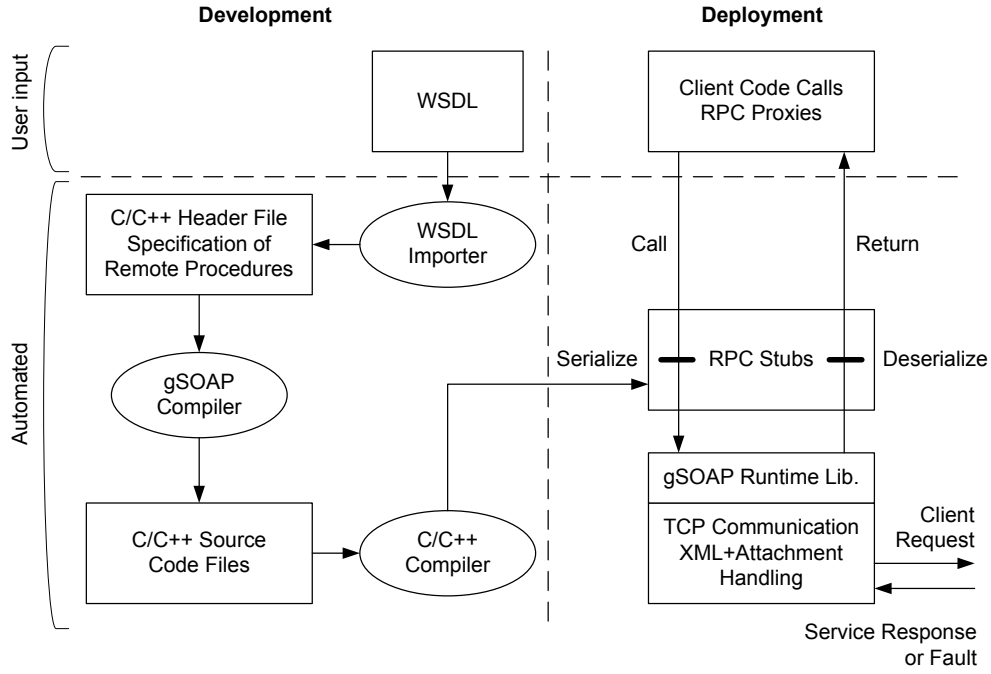


Figure 7.10: gSOAP Web services run-time.

The gSOAP toolkit includes the following libraries and APIs:

- A WSDL parser `wsdl2h` to create header files.
- The stub/skeleton compiler `soapcpp2`. Depending on the SOAP client/server requirements, `.C` or `.CPP` files can be generated.
- The run-time library, `stdsoap2`, serializes and deserializes RPC calls. This library is the only dependency needed on the target platform.

Compared to the gSOAP toolkit, WSA for J2ME has similar capabilities in terms of stub generation. However, it is important to note that WSA can only be used to consume Web services on mobile devices; it cannot provide or host Web services on mobile devices.

7.4.2 Performance Metrics

The aim of our performance study is to compare C++ Web services with Java-based implementations using the Symbian OS. We evaluate performance in terms of latency obtained through roundtrip-delay measurements of SOAP/XML messages per second. To choose a tool for implementing the Web service stack on mobile devices, we need to consider a number of factors on which we establish our platform choice. In the following, we identify a number performance metrics and design considerations:

- *Average time* needed to execute a given request. The average time is calculated as the average time interval the mobile SOAP stack needs to process a request and the corresponding response.
- The *latency* given a number of requests to be executed. Similar to fully featured Web services toolkits (e.g., for standard hardware and server machines) mobile Web services need to process multiple requests concurrently. The request processing latency is tested by issuing several request to be processed by the mobile device.
- The *maximum number of concurrent requests* that can be executed. Since mobile devices have limited resources, the maximum number of requests that can be processed at the same time is limited.
- *Overhead* of using Java (e.g., multi-threaded Java application) in terms of startup overhead, CPU usage and memory consumption (i.e., allocation given a number of requests to be executed).

In our following performance study, we focus on a client or Web services consumer scenario. However, discussions have analog significance for service providers implemented on mobile devices.

7.4.3 Methodology

Performance estimates can be obtained through analytical modeling and simulation or by performing an empirical study to obtain measurements in the real system. For example, we can measure the roundtrip delay of time-stamped messages. In addition, various profilers are available to analyze the performance and bottlenecks of Java-based applications at runtime. Parameters include memory overhead or function calls. In our study, we use a *black box* approach and measure roundtrip delay obtained through time-stamped messages

as we do not assume any detailed knowledge of the server's configuration or load. In addition, we use Java profiling tools available in Sun's WTK 2.2 to obtain "offline" information about the Java SOAP runtime. We gather information by utilizing the memory monitor and the method invocation graph. Notice, WTK profiling tools are available for the wireless terminal emulator. However, the Java profiler adds significant CPU load and memory overhead to the observed system, and should therefore not be used for a comparative online performance study.

7.4.3.1 Approach

A straightforward way to obtain packet statistics is to ping remote hosts by sending ICMP packets. For a given packet size, we measure the roundtrip delay (roundtrip time RTT) for each packet and statistics such *minimum*, *maximum* and *average* RTT in millisecond. In our experiments, we take a similar approach and add time-stamps to each SOAP request/response invocation pair. To trace a request, we add a time stamp (i) t_1 when invoking a method at the SOAP client, (ii) t_2 upon sending the SOAP message through the socket interface, (iii) t_3 when receiving a response at the socket interface, and (iv) t_4 when getting the result of a SOAP call.

7.4.3.2 Calculations

The network is typically governed by various random variables; for example, by the actual load of the network. Additionally, by using Java, we add an time-offset caused by the Java-socket to native interface. We denote this time interval by t'. We calculate the following time intervals: $TI_{wstack}(t_2 - t_1)$ to obtain the time needed to create a SOAP message (time spent in the Web services stack), $TI_{network}(t_3 - (t_2 + t'))$ to calculate the time to receive a response at the socket interface and $TI_{wstack}(t_4 - t_3)$ to calculate the processing time of the response message in the stack. The time offset t' will not be included in our measurements. Furthermore, we calculate

- Maximum time interval: $TI_{stack\_max} = max(TI_{req}) - avg(TI_{network})$
- Average time interval: $TI_{stack\_avg} = avg(TI_{req}) - avg(TI_{network})$
- Minimum time interval:

$$TI_{stack\_min} = \begin{cases} min(TI_{req}) - avg(TI_{network}) & \text{, if } min(TI) - avg(TI) > 0 \\ min(TI_{req}) - min(TI_{network}) & \text{, otherwise} \end{cases}$$

These calculations will be used in our experimental study as shown in Fig. 7.11 (c).

Symbol	Meaning
TI	Time interval measured in the Web services stack when processing a SOAP message.
t'	Time offset governed by random network delay and overhead at Java native interface.

Table 7.11: Performance evaluation of mobile Web services and related symbols.

7.4.4 Setup and Implementation

We use a Symbian OS v8.0a based device[6] to invoke Web services. For example, a SOAP-based search service. Using the aforementioned Symbian platform, the choice of the toolkit is limited to Java and C++. The goal of our architecture to measure performance of mobile Web services toolkits is to execute multiple requests concurrently. Therefore, we use multiple threads — called `RThread` in Symbian's API — to achieve non-blocking operation. We have the following options to develop concurrent applications on Symbian:

- `CActive`: Suitable when all objects run within the same thread. The `RequestStatus` is used to receive asynchronous notifications. This API object cannot be used in our experiments as it would block all Web service calls until they return (i.e., finished execution of request by receiving corresponding response).
- `RThread`: These are standard threads on the Symbian platform. We use `RThreads` and the shared memory model to execute concurrent requests.
- `RProcess`: The kernel object on Symbian. Threads that belong to different processes do not share the same address space. The `RProcess` object is "expensive" in terms of CPU overhead and cannot by used when access to a shared memory space is needed.

On J2ME, standard Java threads can be used to accomplish concurrent execution of requests. The total number of request in execution (at the same time) can be limited depending on the thread pool's configuration. Limiting the size of thread pools has important implications that are based on the memory constraints of devices. Indeed, the Java-based thread pool is more limited in terms of the maximum number of threads that can be executed concurrently.

7.4.4.1 Using Web Services Toolkits in Performance Study

We modify both SOAP stacks, kSOAP and gSOAP, to include a performance "context". In gSOAP, we modify the stack by adding time-stamps when calling the `soap_begin_send` and `soap_begin_recv` methods. In kSOAP, we modify the method "`call`" in the `HttpTransport` class.

[6]S60 2nd Edition FP 2 developer platform.

Remark on the generation of stubs in gSOAP: The gSOAP toolkit supports automatic generation of client side stubs based on WSDLs. In particular, C++ stubs can be generated by using `wsdl2h` and `soapcpp2`. However, there was one shortcoming in gSOAP (at the time of working on the performance study): in some cases modifications in the automatically generated stub code were needed because XML schema type definitions (e.g., `xsd:string`) were not correctly added as input parameters to gSOAP's `soap_out_xsd` functions.

Remark on RPC binding support in WSA (WTK 2.2): When we tried to consume Web services that used the RPC-binding style, WSA's stub generation tool failed because of the unsupported RPC-binding style. However, the document style is typically more often used in today's Web services.

7.4.4.2 Code Optimization

In general, Java-based Web services stacks show larger deviations in terms of minimum and maximum execution time of requests (`min`/`max` values). The reason is non-deterministic garbage collection of the Java Kilobyte Virtual Machine (KVM) and code optimization techniques at run-time. In particular, repeatedly executed parts of the code (called hotspts) are optimized. This gives up to 50% faster execution of subsequent requests when compared to the initial requests which were not optimized by the KVM.

7.4.5 Results

Our results are based on measurements over a one-week time period. We collected statistics of about 2000 test cases. The presented results were calculate based on the Google search service. The figures in 7.11 are based on the number of requests concurrently in execution. For example, 4 to 16 requests denote the amount of concurrent requests that were issued at the same time. In our experiments, the maximum number of concurrent requests that can be issued in gSOAP was significantly higher than in kSOAP. This is a quite intuitive observation because gSOAP is implemented in native C/C++ code. However, in order to obtain comparable measurements (e.g., time needed to execute multiple requests), we limited the maximum number of requests to be executed — in gSOAP as well as kSOAP — to a maximum of 16 (i.e., the thread pool size in Java).

In Fig. 7.11, we show the direct comparison of C++ and Java-based toolkits for different request rates (depending on the maximum threshold between 4 and 16). The performance figures show average time values for TI_{send}, the time interval to create a request; $TI_{receive}$, the time interval to receive the corresponding response; $TI_{network}$, the network latency; TI_{req}, total time required to execute a given request and receiving the corresponding response (without $TI_{network}$); and finally TI_{total} which is the total time needed to process requests and responses (including $TI_{network}$).

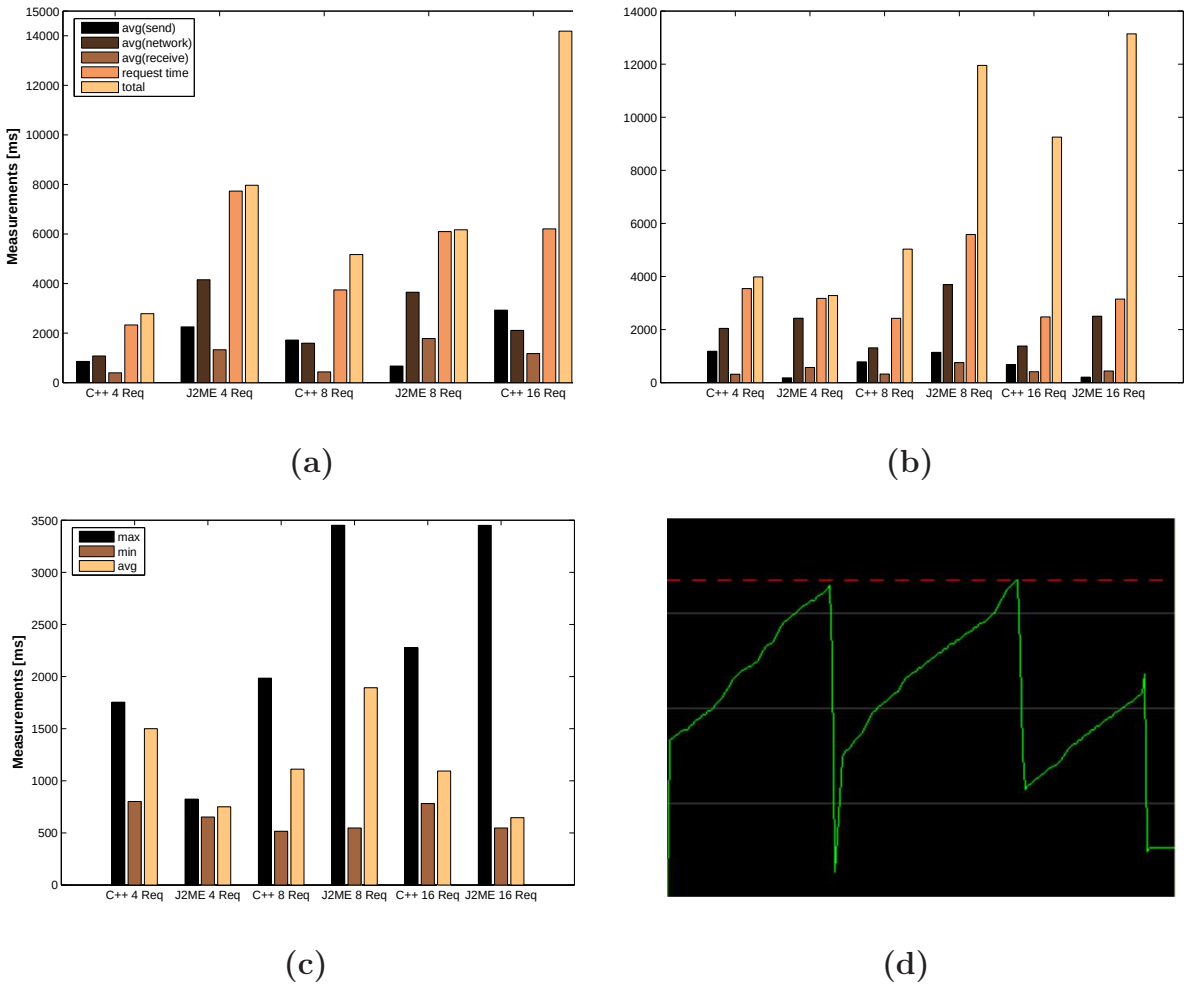


Figure 7.11: **(a)** Time intervals showing the maximum throughput and **(b)** using a fixed number of threads. Java (kSOAP) performs better in creating a request TI_{send}, but is slower in receiving a response $TI_{receive}$. Slow SOAP response processing becomes more apparent when executing the Java application on the WTK emulator. On average, the C++ toolkit is faster in executing a number of requests TI_{total} and faster in executing a given request TI_{req}. **(c)** Deviations of measurements in performance study showing best, worst and average processing time, given a number of requests to be executed. In this diagram, the *net* processing time in the Web services stack is shown (i.e., we do not add $TI_{network}$). **(d)** KVM profiler showing memory monitor (KVM profiling in WTK 2.2). We see the monitor while executing eight concurrent requests with a given thread pool size of 4. The emulated Java test application on the mobile device, however, may crash due to the allocation of large amounts of memory. In this example, the current memory allocation of the emulator is 67068 bytes, given a maximum memory size of 419468 bytes. We observed a sawtooth behavior where memory allocation hits the maximum memory limit, dashed line indicating maximum memory usage, and thus resulting in runtime exceptions on the KVM and unpredictable execution of our performance test application. Parsing responses takes a very long time due to the method calls `readUnknown` in kSOAP's `SoapSerializationEnvelope`; the main bottleneck in the kSOAP stack.

Chapter 8

Conclusion

We believe that Human-Provided Services will become increasingly important in human-centered systems. Service-oriented architectures can no longer be designed in a top-down manner because human interactions in SOA demand for flexibility.

The presented HPS framework was developed with the objective of enabling the user-driven approach to the design and provisioning of HPSs. The application scenarios of HPS range from ad-hoc collaborations, semi-structured processes, and human interactions in formalized processes. In this work, the focus was the architectural design and implementation of a framework supporting ad-hoc interactions in open and dynamic collaboration environments. The prototype has to address many different areas including the integration of existing Web services technologies and standards. Thus, aspects such as compositions of HPS and software services as well as interaction rules have been discussed at the conceptual level. The integration with B4P engines has to be addressed in our future work. Also, legal or privacy issues were not addressed at this stage.

The most promising direction for future research and development of the HPS framework is the automatic generation of HPS interfaces based on user skills and profile information. Furthermore, a detailed analysis of DSARank in networks of human and software services is part of our future work. An important task will be the aggregation of metrics in more general scoring functions.

References

Adamic, L., J. Zhang, E. Bakshy, and M. Ackerman (2008). Knowledge sharing and yahoo answers: Everyone knows something. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pp. 665–674. ACM.

Adams, M., A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst (2006). Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *OTM Conferences (1)*, pp. 291–308.

Agrawal, A., M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller (2007). WS-BPEL Extension for People (BPEL4People), Version 1.0.

Aiello, M. and S. Dustdar (2008). Are our homes ready for services? a domotic infrastructure based on the web service stack. *Pervasive and Mobile Computing 4*(4), 506–525.

Aleman-Meza, B., U. Bojars, H. Boley, J. G. Breslin, M. Mochol, L. J. Nixon, A. Polleres, and A. V. Zhdanova (2007, June). Combining RDF vocabularies for expert finding. In *4th European Semantic Web Conference (ESWC2007)*, pp. 235–250. Springer.

Alonso, G., F. Casati, H. Kuno, and V. Machiraju (2003, November). *Web Services - Concepts, Architectures and Applications*. Springer.

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

Andreozzi, S., P. Ciancarini, D. Montesi, and R. Moretti (2006). An approach to the quantitative evaluation of grid services. *Concurr. Comput.: Pract. Exper. 18*(8), 827–836.

Andrews, T., F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, , and S. Weerawarana (2003). Business process execution language for web services, version 1.1. oasis.

Barabási, A.-L. (2005). The origin of bursts and heavy tails in human dynamics. *Nature 435*, 207.

Barabási, A.-L. (2007). The architecture of complexity: from network structure to human dynamics. *IEEE Control Systems Magazine 27*(4), 33–42.

Barabasi, A.-L. and R. Albert (1999). Emergence of scaling in random networks. *Science 286*, 509.

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

Berkhin, P. (2005). A survey on PageRank computing. *Internet Mathematics 2*(1), 73–120.

Bianchini, M., M. Gori, and F. Scarselli (2005, February). Inside pagerank. *ACM Trans. Inter. Tech. 5*(1), 92–128.

Brin, S. and L. Page (1998). The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst. 30*(1-7), 107–117.

Brinkmeier, M. (2006). PageRank revisited. *ACM Transaction on Internet Technologies 6*(3), 257–279.

Butler, D. (2006, January). Mashups mix data into global service. *439*(7072), 6–7.

Byde, A., H. Wan, and S. Cayzer (2007, March). Personalized tag recommendations via tagging and content-based similarity metrics. In *Proceedings of the International Conference on Weblogs and Social Media*.

Caldarelli, G. (2007, May). *Scale-Free Networks: Complex Webs in Nature and Technology (Oxford Finance)*. Oxford University Press, USA.

Cattuto, C., V. Loreto, and L. Pietronero (2007, January). Semiotic dynamics and collaborative tagging. *PNAS 104*(5), 1461–1464.

Chakrabarti, S. (2007). Dynamic personalized pagerank in entity-relation graphs. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, New York, NY, USA, pp. 571–580. ACM.

Constantin, C., B. Amann, and D. Gross-Amblard (2006). A Link-Based Ranking Model for Services. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pp. 327–344. Springer Berlin / Heidelberg.

Conyon, M. J. and M. R. Muldoon (2006, April). Ranking the importance of boards of directors. Mims eprint, Manchester Institute for Mathematical Sciences, University of Manchester.

Crawley, E., O. de Weck, S. Eppinger, C. Magee, J. Moses, W. Seering, J. Schindall, D. Wallace, and D. Whitney (2004, March). The influence of architecture in engineering systems (monograph).

Cugola, G., E. D. Nitto, A. Fuggetta, and C. Ghezzi (1996). A framework for formalizing inconsistencies and deviations in human-centered systems. *ACM Trans. Softw. Eng. Methodol. 5*(3), 191–230.

Dezsö, Z., E. Almaas, A. Lukács, B. Rácz, I. Szakadát, and A.-L. Barabási (2006, June). Dynamics of information access on the web. *Physical Review E 73*.

Dom, B., I. Eiron, A. Cozzi, and Y. Zhang (2003). Graph-based ranking algorithms for e-mail expertise analysis. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, New York, NY, USA, pp. 42–48. ACM.

Dorn, C., D. Schall, and S. Dustdar (2006). Granular context in collaborative mobile environments. In *International Workshop on Context-Aware Mobile Systems (CAMS'06)*, Montpellier, France. Springer.

Dorn, C., D. Schall, and S. Dustdar (2008, Oct.). Achieving team-awareness in scientific grid environments. In *GCC2008: 7th International Conference on Grid and Cooperative Computing (GCC2008) and Second EchoGRID Conference*, Shenzhen, China. IEEE Computer Society.

Dorn, C., D. Schall, R. Gombotz, and S. Dustdar (2007). A view-based analysis of distributed and mobile teams. In *WETICE '07: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Washington, DC, USA, pp. 198–203. IEEE Computer Society.

Dujmovic, J. J. (2007). Continuous preference logic for system evaluation. In *IEEE Transactions on Fuzzy Systems*, Volume 15, pp. 1082–1099. IEEE Computer Society.

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

Dustdar, S. and T. Hoffmann (2007). Interaction pattern detection in process oriented information systems. *Data Knowl. Eng. 62*(1), 138–155.

Dustdar, S., T. Hoffmann, and W. M. van der Aalst (2005). Mining of ad-hoc business processes with teamlog. *Data Knowl. Eng. 55*(2), 129–158.

Eagle, N. and A. S. Pentland (2006). Reality mining: sensing complex social systems. *Personal Ubiquitous Comput. 10*(4), 255–268.

Engelen, R. A. (2003a). Pushing the SOAP Envelope with Web Services for Scientific Computing. *The International Conference on Web Services ICWS*.

Engelen, R. A. (2003b). *SOAP/XML Web Service Performance*. http://www.cs.fsu.edu/ engelen/soapperformance.html.

Engelen, R. A. (2004). gSOAP: C/C++ Web Services Toolkit. Available from http://gsoap2.sourceforge.net/.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. Ph. D. thesis.

Fogaras, D., K. Csalogany, B. Racz, and T. Sarlos (2005). Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics 2*(3), 333–358.

Garlan, D., V. Poladian, B. R. Schmerl, and J. P. Sousa (2004). Task-based self-adaptation. In *WOSS*, pp. 54–57.

Gekas, J. (2006). Web Service Ranking in Service Networks. In *3rd European Semantic Web Conference (ESWC 2006)*. Springer Berlin / Heidelberg.

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

Golder, S. and B. A. Huberman (2006). Usage patterns of collaborative tagging systems. *Journal of Information Science 32*(2), 198–208.

Gombotz, R., D. Schall, C. Dorn, and S. Dustdar (2006, Nov.). Relevance-based context sharing through interaction patterns. *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on*, 1–7.

Guimerà, R., L. Danon, A. Díaz-Guilera, F. Giralt, and A. Arenas (2003, Dec.). Self-similar community structure in a network of human interactions. *Physical Review E 68*(6).

Guimerà, R., M. Sales-Pardo, and L. A. N. Amaral (2007). Module identification in bipartite and directed networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics) 76*(3), 036102.

Günther, C., S. Rinderle, M. Reichert, W. M. van der Aalst, and J. Recker (2008). Using process mining to learn from process changes in evolutionary systems. *International Journal of Business Process Integration and Management 3*(1), 61–79.

Haveliwala, T. H. (2002). Topic-sensitive pagerank. In *International Conference on World Wide Web*, pp. 517–526.

Heymann, P., D. Ramage, and H. Garcia-Molina (2008). Social tag prediction. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, pp. 531–538. ACM.

Holmes, T., M. Vasko, and S. Dustdar (2008). Viebop: Extending bpel engines with bpel4people. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on 0*, 547–555.

IBM (2005). An architectural blueprint for autonomic computing (whitepaper).

Jammes, F., A. Mensch, and H. Smit (2005). Service-Oriented device communications using the devices profile for web services. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pp. 1–8. ACM Press.

Jäschke, R., L. B. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme (2007). Tag recommendations in folksonomies. In *Knowledge Discovery in Databases: PKDD 2007*, Volume 4702 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, pp. 506–514. Springer.

Jeh, G. and J. Widom (2003). Scaling personalized web search. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, New York, NY, USA, pp. 271–279. ACM.

Karagiannis, T. and M. Vojnovic (2008). Email Information Flow in Large-Scale Enterprises. Technical report, Microsoft Research.

Kassoff, M., D. Kato, and W. Mohsin (2003). Creating GUIs for Web Services. *IEEE Internet Computing 07*(5), 66–73.

Kim, E. and Y. Lee (2005). Oasis web services quality model tc, version 2.0. oasis. http://www.oasis-open.org.

Kleinberg, J. (2008). The convergence of social and technological networks. *Commun. ACM 51*(11), 66–72.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM 46*, 668–677.

Kolmogorov, A. N. (1983). Combinatorial foundations of information theory and the calculus of probability. *Russian Mathematical Surveys 38*, 29–40.

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

Kumar, A., W. M. P. V. D. Aalst, and E. Verbeek (2002). Dynamic work distribution in workflow management systems: How to balance quality and performance. *J. Manage. Inf. Syst. 18*(3), 157–193.

Levis, A. (1999). System architectures. pp. 427–454.

Leymann, F. (2006). Workflow-based coordination and cooperation in a service world. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, 2–16.

Li, X., L. Guo, and Y. E. Zhao (2008). Tag-based social interest discovery. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, New York, NY, USA, pp. 675–684. ACM.

Lieberman, E., C. Hauert, and M. A. Nowak (2005, January). Evolutionary dynamics on graphs. *Nature 433*, 312–316.

Liu, Y., A. H. Ngu, and L. Z. Zeng (2004). Qos computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, New York, NY, USA, pp. 66–73. ACM.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press. Available from http://www.inference.phy.cam.ac.uk/mackay/itila/.

Maximilien, E., A. Ranabahu, and K. Gomadam (2008, Sept.). An online platform for web apis and service mashups. *12*(5), 32–43.

Maximilien, E. M. and M. P. Singh (2004). Toward autonomic web services trust and selection. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, New York, NY, USA, pp. 212–221. ACM.

Mendling, J., K. Ploesser, and M. Strembeck (2008). Specifying separation of duty constraints in bpel4people processes. In *Proceedings of the 11th Int'l Conference on Business Information Systems (BIS 2008)*, LNBIP, Innsbruck, Austria, pp. 273–284. Springer Verlag.

Milo, R., , S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon (2002, October). Network motifs: simple building blocks of complex networks. *Science 298*(5594), 824–827.

Milo, R., S. Itzkovitz, N. Kashtan, R. Levitt, S. S. Orr, I. Ayzenshtat, M. Sheffer, and U. Alon (2004). Superfamilies of evolved and designed networks. *Science 303*, 1538–1542.

Moody, P., D. Gruen, M. J. Muller, J. Tang, and T. P. Moran (2006). Business Activity Patterns: A New Model for Collaborative Business Applications. *45*(4), 683–694.

Naor, M. (1996). Verification of a human in the loop or identification via the turing test.

Newman, M., A.-L. Barabasi, and D. J. Watts (2006). *The Structure and Dynamics of Networks (Princeton Studies in Complexity)*. Princeton, NJ, USA: Princeton University Press.

Newman, M. E. J. (2006, June). Modularity and community structure in networks. *PNAS 103*(23).

Nowak, M. A. (2006, December). Five rules for the evolution of cooperation. *Science 314*(5805), 1560–1563.

Oliveira, J. and A.-L. Barabási (2005). The Correspondence Patterns of Darwin and Einstein. *Nature 437*, 1251.

Onnela, J.-P., J. Saramäki, J. Hyvönen, G. Szabó, M. A. de Menezes, K. Kaski, A.-L. Barabási, and J. Kertész (2007). Analysis of a large-scale weighted network of one-to-one human communication. *New Journal of Physics 9*(6), 179.

Onnela, J.-P., J. Saramäki, J. Kertész, and K. Kaski (2005). Intensity and coherence of motifs in weighted complex networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics) 71*(6).

Page, L., S. Brin, R. Motwani, and T. Winograd (1998). The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project.

Pandurangan, G., P. Raghavan, and E. Upfal (2006). Using pagerank to characterize web structure. *Internet Mathematics 3*(1).

Papazoglou, M. P., P. Traverso, S. Dustdar, and F. Leymann (2007). Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer 40*(11), 38–45.

Pham, L. and G. Gehlen (2005, Apr). Realization and Performance Analysis of a SOAP Server for Mobile Devices. Volume 02, pp. 791–797. VDE Verlag.

Ramakrishnan, R. and A. Tomkins (2007, Aug.). Toward a PeopleWeb. *Computer 40*(8), 63–72.

Ran, S. (2003). A model for web services discovery with qos. *SIGecom Exch. 4*(1), 1–10.

Rellermeyer, J. and G. Alons (2007). Services everywhere: Osgi in distributed environments. *EclipseCon 2007*.

Richardson, M. and P. Domingos (2002). The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *Advances in Neural Information Processing Systems 14*, pp. 1441–1448. MIT Press.

Riolo, R. L., M. D. Cohen, and R. Axelrod (2001, November). Evolution of cooperation without reciprocity. *Nature 414*(6862), 441–443.

Rodrigues, J. F., H. Tong, A. J. M. Traina, C. Faloutsos, and J. Leskovec (2006). Gmine: a system for scalable, interactive graph visualization and mining. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pp. 1195–1198. VLDB Endowment.

Rosenberg, F., C. Platzer, and S. Dustdar (2006). Bootstrapping performance and dependability attributes ofweb services. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, Washington, DC, USA, pp. 205–212. IEEE Computer Society.

Rosvall, M. and C. T. Bergstrom (2008). Maps of random walks on complex networks reveal community structure. *PNAS 105 4*, 1118–1123.

Rosvall, M. and K. Sneppen (2006). Self-assembly of information in networks. *Europhysics Letters 74*, 1109.

Russell, N. and W. M. P. V. D. Aalst (2007). Evaluation of the bpel4people and ws-humantask extensions to ws-bpel 2.0 using the workflow resource patterns. Technical report, BPM Center Brisbane/Eindhoven.

Russell, S. J. and P. Norvig (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education.

Schall, D., M. Aiello, and S. Dustdar (2006). Web services on embedded devices. *IJWIS 2*(1), 45–50.

Schall, D., C. Dorn, and S. Dustdar (2007a). Context-aware mobile computing. *Encyclopedia of Wireless and Mobile Communications, edited by Borko Furht*.

Schall, D., C. Dorn, and S. Dustdar (2007b). Wireless internet applications. *Encyclopedia of Wireless and Mobile Communications, edited by Borko Furht*.

Schall, D., C. Dorn, S. Dustdar, and I. Dadduzio (2008). VieCAR - Enabling Self-adaptive Collaboration Services. In *34th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society.

Schall, D., R. Gombotz, C. Dorn, and S. Dustdar (2007). Human Interactions in Dynamic Environments through Mobile Web Services. In *International Conference on Web Services (ICWS'07)*, Salt Lake City, USA, pp. 912–919. IEEE Computer Society.

Schall, D., R. Gombotz, and S. Dustdar (2006, Nov.). Discovering Service-Interaction Patterns - Methods and Mining Algorithms. Technical report, Vienna University of Technology.

Schall, D., R. Gombotz, and S. Dustdar (2007). Pattern-based collaboration in ad-hoc teams through message annotation. In J. Cardoso, J. Cordeiro, and J. Filipe (Eds.), *ICEIS (4)*, pp. 84–91.

Schall, D., H.-L. Truong, and S. Dustdar (2008a, July). The Human-provided Services Framework. In *IEEE 2008 Conference on Enterprise Computing, E-Commerce and E-Services (EEE '08)*, Crystal City, Washington, D.C., USA. IEEE Computer Society.

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

Sheng, Q. Z., B. Benatallah, Z. Maamar, M. Dumas, and A. H. H. Ngu (2004). Enabling personalized composition and adaptive provisioning of web services. In *CAiSE*, pp. 322–337.

Shetty, J. and J. Adibi (2005). Discovering important nodes through graph entropy the case of enron email database. In *LinkKDD '05: Proceedings of the 3rd international workshop on Link discovery*, New York, NY, USA, pp. 74–81. ACM.

Shi, X., M. Bonner, L. A. Adamic, and A. C. Gilbert (2008). The very small world of the well-connected. In *HT '08: Proceedings of the nineteenth ACM conference on Hypertext and hypermedia*, New York, NY, USA, pp. 61–70. ACM.

Song, K. and K.-H. Lee (9-13 July 2007). An Automated Generation of XForms Interfaces for Web Services. *Web Services, 2007. ICWS 2007. IEEE International Conference on*, 856–863.

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

Tang, J., J. Zhang, D. Zhang, L. Yao, C. Zhu, and J.-Z. Li (2007). Arnetminer: An expertise oriented search system for web community. In *Semantic Web Challenge*.

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

Tierno, J. and C. Campo (2005). Smart Camera Phones: Limits and Applications. *IEEE Pervasive Computing 04*(2), 84–87.

Tilly, M., H. Q. Yu, D. Schall, and S. Peray (2007, July). Design and Implementation of Monitoring and Aggregation Mechanisms for Context-based Services – Version 1.0. Technical report, inContext Consortium.

Tilly, M., H. Q. Yu, D. Schall, and S. Peray (2008, April). Design and implementation of monitoring and aggregation mechanisms for context-based services – version 2.0. Technical report, inContext Consortium.

Treiber, M. and S. Dustdar (2007). Active web service registries. *IEEE Internet Computing 11*(5), 66–71.

Vamos, T. (1983, Aug.). Cooperative systems an evolutionary perspective. *IEEE Control Systems Magazine 3*(3).

van Dongen, B., A. A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst (2005). The prom framework: A new era in process mining tool support. *Application and Theory of Petri Nets 2005 3536*, 444–454.

van Dongen, B. F. and W. van der Aalst (2005). A meta model for process mining data. In *Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop)*, pp. 309–320. ACM Press.

Vázquez, A., J. G. Oliveira, Z. Dezsö, K.-I. Goh, I. Kondor, and A.-L. Barabási (2006). Modeling bursts and heavy tails in human dynamics. *Physical Review E 73*.

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

Wang, X., T. Tao, J.-T. Sun, A. Shakery, and C. Zhai (2008). Dirichletrank: Solving the zero-one gap problem of pagerank. *ACM Trans. Inf. Syst. 26*(2), 1–29.

Wasserman, S. and K. Faust (1994, November). *Social Network Analysis : Methods and Applications*. Cambridge University Press.

Watts, D. J. (1999, August). *Small Worlds*. Princeton University Press.

Watts, D. J. and S. H. Strogatz (1998, June). Collective dynamics of small-world networks. *Nature 393*(6684), 440–442.

White, S. and P. Smyth (2003). Algorithms for estimating relative importance in networks. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 266–275. ACM Press.

Wu, C. and E. Chang (2007). An Atom-based Architecture for Web services Discovery. *Service Oriented Computing and Applications 1*(2), 97–116.

Yang, J., L. Adamic, and M. Ackerman (2008). Competing to share expertise: the taskcn knowledge sharing community. In *International Conference on Weblogs and Social Media*.

Zhang, J., M. S. Ackerman, and L. Adamic (2007). Expertise networks in online communities: structure and algorithms. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, New York, NY, USA, pp. 221–230. ACM.

Appendix A

Screenshots

The first screenshot in Fig. A.1 shows the Web 2.0-based activity management user interface implemented in the inContext project by Softeco Sismat. This screenshot illustrates the hierarchical structure of activity-based work items.

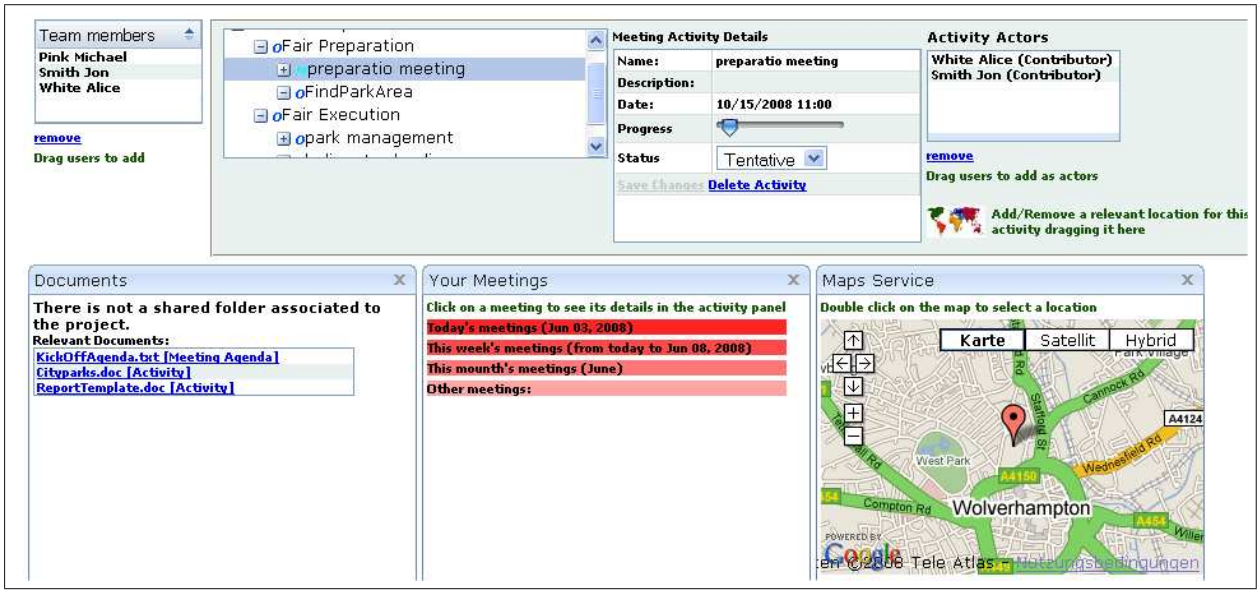


Figure A.1: Activity management GUI screenshot.

By selecting an activity (e.g., `preparation meeting`), activity details are shown such as name, date, and progress. Furthermore, the `Activity Actors` and their roles are shown. Each activity may have multiple services associated with it. For example, in this screenshot we see the document, meeting scheduling, and map service.

In Fig. A.2, an example screenshot of context-based discovery of HPSs is shown. A particular HPS is provided by multiple human actors. The requester queries the HPS lookup (e.g., location-based search — text box and `Go!` button) to retrieve a set of HPSs.

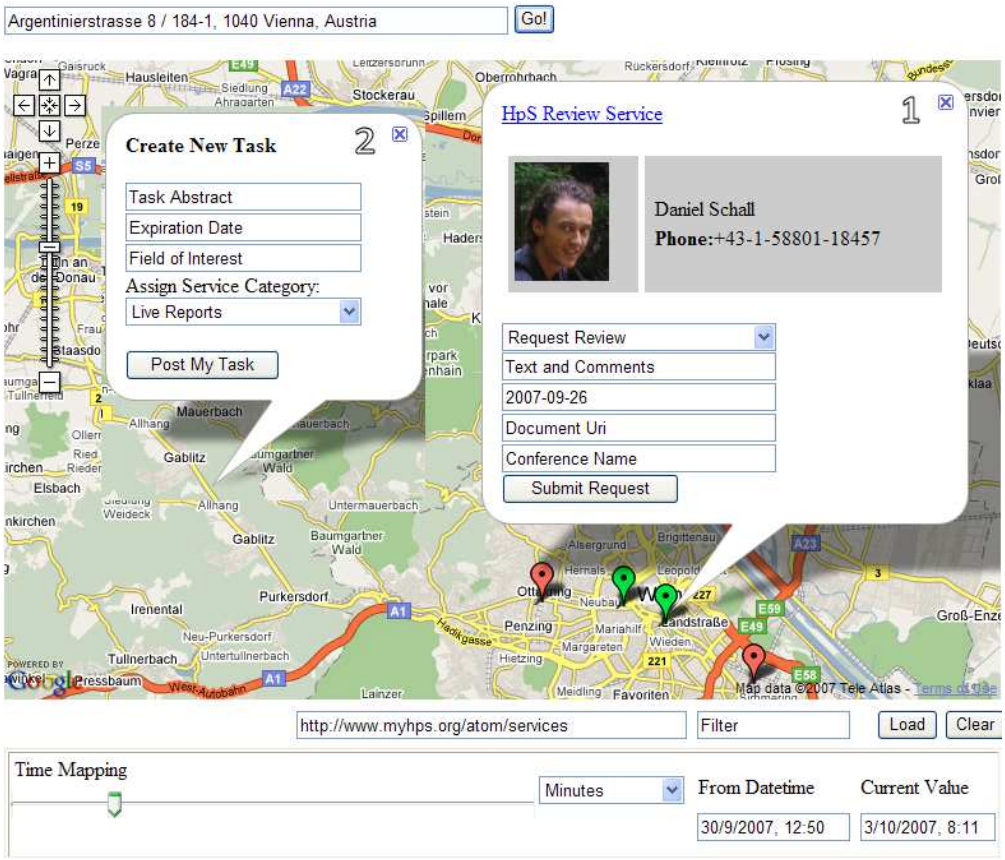


Figure A.2: Context-based discovery of HPSs.

The lookup result is shown in Listing A.1.

```
<entry>
 <title>My HPS Review Service</title>
 <author><name>Daniel Schall</name></author>
 <category term="/services/reviewservice#WSDL"
    schema="http://schemas.xmlsoap.org/soap/http"/>
 <category term="/services/reviewservice#WADL" label="json"/>
 <geo:lat>48.19766</geo:lat>
 <geo:long>16.37146</geo:long>
</entry >
```

Listing A.1: Personal service description of review service.

Filters can be applied to retrieve (i) a certain type of HPSs — expertise-based lookup, (ii) time constraints — the availability of HPSs (depicted as `Time Mapping`), and (iii) location constraints — services at a certain location. The requester has the following options: (1) HPS-based interactions using forms as illustrated by the popup window (right-hand side) in Fig. A.2 or (2) create new task announcements to denote the need for HPSs. Such announcements can be associated with one or more `Service Category`.

Appendix B

XML Listings

We give a detailed example of an HPS, which can be defined by using HPS design tools. An alternative way of defining an HPS is the definition of predefined services and user interfaces, which are deployed within the framework without requiring the user to go through the design process.

In Listing B.1 we show a WSDL that defines types, mapping to HPS, and interactions with HAL. The corresponding XML schema instance document is shown in Listing B.2 and the XHtml displaying the form in Listing B.3 (we abbreviated long XPath statements). Finally, we provide an excerpt of type definitions in Listing B.4.

B.1 Review Service WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=".../reviewservice"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:tns="http://services.myhps.org/reviewservice"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:rt="http://schemas.datacontract.org/2004/07/reviewservice">
<wsdl:documentation>Review HPS</wsdl:documentation>
<wsdl:types>
  <xsd:schema elementFormDefault="qualified"
   targetNamespace="http://services.myhps.org/reviewservice">
   <xsd:complexType name="GenericResource">
    <xsd:sequence>
     <xsd:element name="Location" type="xsd:anyURI">
```

```
      </xsd:element>
      <xsd:element name="Expires" type="xsd:dateTime"
       maxOccurs="1" minOccurs="0">
      </xsd:element>
      <xsd:element name="Tags" type="tns:Category"
       maxOccurs="unbounded" minOccurs="0">
      </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Category">
     <xsd:sequence>
      <xsd:element name="term" type="xsd:string"></xsd:element>
      <xsd:element name="scheme" type="xsd:anyURI"></xsd:element>
      <xsd:element name="label" type="xsd:string"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="ReviewRequest" type="tns:Request">
    </xsd:element>
    <xsd:complexType name="Request">
     <xsd:sequence>
      <xsd:element name="ReviewResource" type="tns:GenericResource"
       minOccurs="1" maxOccurs="1"></xsd:element>
      <xsd:element name="Comments" type="xsd:string"
       minOccurs="0" maxOccurs="unbounded"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="ReviewReply" type="tns:Reply"></xsd:element>
    <xsd:complexType name="Reply">
     <xsd:sequence>
      <xsd:element name="ReplyResource" type="tns:GenericResource"
       minOccurs="1" maxOccurs="unbounded"></xsd:element>
      <xsd:element name="Comments" type="xsd:string"
       minOccurs="0" maxOccurs="unbounded"></xsd:element>
     </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="AckReviewRequest" type="xsd:string">
    </xsd:element>
    <xsd:element name="GetReviewReply" type="xsd:string">
    </xsd:element>
   </xsd:schema>
</wsdl:types>
<wsdl:message name="GetReview">
 <wsdl:part name="part1" element="tns:ReviewRequest" />
```

```
</wsdl:message>
<wsdl:message name="AckReviewRequest">
 <wsdl:part name="part1" element="tns:AckReviewRequest">
 </wsdl:part>
</wsdl:message>
<wsdl:message name="GetReviewReply">
 <wsdl:part name="part1" element="tns:GetReviewReply" />
</wsdl:message>
<wsdl:message name="GetReviewReplyResponse">
 <wsdl:part name="part1" element="tns:ReviewReply"></wsdl:part>
</wsdl:message>
<wsdl:portType name="HPSReviewPortType">
 <wsdl:operation name="GetReview">
  <wsdl:input
   xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
   message="tns:GetReview" wsaw:Action="urn:GetReview" >
  </wsdl:input>
  <wsdl:output message="tns:AckReviewRequest">
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="GetReviewReply">
  <wsdl:input
   xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
   message="tns:GetReviewReply" wsaw:Action="urn:GetReviewReply" />
  <wsdl:output message="tns:GetReviewReplyResponse"></wsdl:output>
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HALSOAPBinding" type="tns:HPSReviewPortType">
 <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="GetReview">
  <soap:operation
  soapAction="http://services.myhps.org/reviewservice/GetReview" />
    <wsdl:input>
     <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal" />
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="GetReviewReply">
    <soap:operation soapAction=
     "http://services.myhps.org/reviewservice/GetReviewReply" />
```

```
    <wsdl:input>
     <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
     <soap:body use="literal" />
    </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
<wsdl:service name="ReviewService">
 <wsdl:port name="HALSOAPBinding_http"
  binding="tns:HALSOAPBinding">
 <soap:address location="" />
 </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Listing B.1: Excerpt XML schema review service WSDL (flattened).

B.2 XML Schema Instance

```
<?xml version="1.0" encoding="ASCII"?>
<soapenv:Envelope soapenv:encodingStyle=
 "http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soapenv:Header xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance">
 <wsa:From>http://www.myhps.org/hal/id1</wsa:From>
 <wsa:To></wsa:To>
 <wsa:Action>http://services.myhps.org/reviewservice/GetReview
 </wsa:Action>
</soapenv:Header>
<soapenv:Body xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance">
 <reviewservice:ReviewRequest xmlns:reviewservice=
  "http://services.myhps.org/reviewservice"
  type="reviewservice:Request">
 <reviewservice:ReviewResource type=
  "reviewservice:GenericResource">
  <reviewservice:Location type="xsd:anyURI" />
```

```
  <reviewservice:Expires type="xsd:dateTime">2008-05-14T09:59:37Z
  </reviewservice:Expires>
  <reviewservice:Tags type="reviewservice:Category">
  <reviewservice:term type="xsd:string"/>
  <reviewservice:scheme type="xsd:anyURI"/>
  <reviewservice:label type="xsd:string"/>
  </reviewservice:Tags>
 </reviewservice:ReviewResource>
 <reviewservice:Comments type="xsd:string">Free Text Comments
 </reviewservice:Comments>
 </reviewservice:ReviewRequest>
</soapenv:Body>
</soapenv:Envelope>
```

Listing B.2: Excerpt XML schema instance of a review HPS.

B.3 XHTML Input Form

```
<?xml version="1.0" encoding="ASCII"?>
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:ev="http://www.w3.org/2001/xml-events"
 xmlns:xforms="http://www.w3.org/2002/xforms"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:reviewservice="http://services.myhps.org/reviewservice"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
<head>
 <title>GetReview</title>
 <xforms:model id="model_Envelope">
 <xforms:instance id="instance_model_Envelope"
  src="GetReview.xml"/>
 <xforms:submission id="submit_model_Envelope" action=""
  method="post" mediatype="text/xml" replace="instance">
  <xforms:toggle ev:event="xforms-submit-done" case="response"/>
 </xforms:submission>
 </xforms:model>
  <link href="style.css" rel="stylesheet"/>
</head>
<body>
<xforms:label>Message Header</xforms:label>
```

```
<xforms:group>
 <xforms:input ref="/soapenv:Envelope/soapenv:Header/wsa:From"
  model="model_Envelope">
 <xforms:label>From</xforms:label>
 </xforms:input>
 <xforms:input ref="/soapenv:Envelope/soapenv:Header/wsa:To"
  model="model_Envelope">
 <xforms:label>To</xforms:label>
 </xforms:input>
</xforms:group>
<xforms:switch>
 <xforms:case id="request" selected="true">
 <xforms:label>ReviewResource</xforms:label>
 <xforms:group>
  <xforms:input
   ref="/soapenv:Envelope/soapenv:Body/reviewservice:..."
   model="model_Envelope">
  <xforms:label>Location</xforms:label>
  </xforms:input>
  <xforms:input
   ref="/soapenv:Envelope/soapenv:Body/reviewservice:..."
   model="model_Envelope">
  <xforms:label>Expires</xforms:label>
 </xforms:input>
       <!-- Remaining elements omitted -->
 <xforms:submit submission="submit_model_Envelope">
  <xforms:label>Submit</xforms:label>
  <xforms:message ev:event="xforms-submit-done"
   ev:observer="submit_model_Envelope"
   level="modal">Submission successful.</xforms:message>
 <xforms:message ev:event="xforms-submit-error"
   ev:observer="submit_model_Envelope"
   level="modal">Ensure that all entry fields are valid.
 </xforms:message>
</xforms:submit>
</xforms:case>
<xforms:case id="response">
 <xforms:output ref=
 "/soapenv:Envelope/soapenv:Body/reviewservice:AckReviewRequest"
  model="model_Envelope">
 <xforms:label>Get Review Response</xforms:label>
 </xforms:output>
</xforms:case>
```

```
</xforms:switch>
</body>
</html>
```

Listing B.3: Excerpt XHTML input form for review HPS.

B.4 Predefined XML Types

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://types.myhps.org/t/2008/03"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://types.myhps.org/t/2008/03"
  elementFormDefault="qualified" attributeFormDefault="qualified">
  <xsd:simpleType name="application">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="application/javascript"/>
      <xsd:enumeration value="application/octet-stream"/>
      <xsd:enumeration value="application/soap+xml"/>
        <!-- Remaining elements omitted -->
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="message">
    <xsd:restriction base="xsd:string">
    <xsd:annotation>
    <xsd:documentation xml:lang="en">
        Email message including any headers
    </xsd:documentation>
    </xsd:annotation>
      <xsd:enumeration value="message/rfc822"></xsd:enumeration>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="Mediatype"/>
  <xsd:complexType name="Artifact"/>
</xsd:schema>
```

Listing B.4: Excerpt XML schema predefined types (reusable in various activities).

B.5 Meta Model for Interface Mappings

- Simple (String)
  - XSD: `<xs:element/>` Simple Type `xs:string`
  - Form: `<xf:input ref=""/>`
  - Restriction/Model: `<xs:length value="$model" />`
    parameter `//*[@value='Text']/prop[@name='length']/@value`
- Simple (Textarea)
  - XSD: `<xs:element/>` Simple Type `xs:string`
  - Form: `<xf:textarea ref=""/>`
  - Restriction/Model: `<xs:length value="$model" />`
    parameter `//*[@value='Paragraph']/prop[@name='length']/@value`
- Complex
  - XSD: `<xs:complexType/>`
  - Form: -
  - Restriction/Model: Recursive inspection of schema particle and construction of comprising elements as form
- Choice
  - XSD: `<xs:choice/>`
  - Form: `<xf:select1 ref="" appearance="$model"/>` with appearance as $model parameter ("full" or "compact")
  - Restriction/Model: `//*[@value='Choice']/prop[@name='type']/@value` and `<xs:choice minOccurs="$model" maxOccurs="$model">` with $model as parameters (minOccurs and maxOccurs being "1")
- Enumeration (Select1)
  - XSD: `<xs:enumeration/>`
  - Form: `<xf:select1 ref="" appearance="minimal">`
  - Restriction/Model: `//*[@value='List']/prop[@name='type']/@value` maps a list to enumeration
- Enumeration (Select)
  - XSD: `<xs:choice/>`

- Form: `<xf:select ref="" appearance="full">`
- Restriction/Model: `//*[@value='Checkboxes']/prop[@name='type']/@value` and `<xs:choice minOccurs="$model" maxOccurs="$model">` with $model as parameters (minOccurs="0" and maxOccurs="unbounded")

Appendix C

Model Diagrams

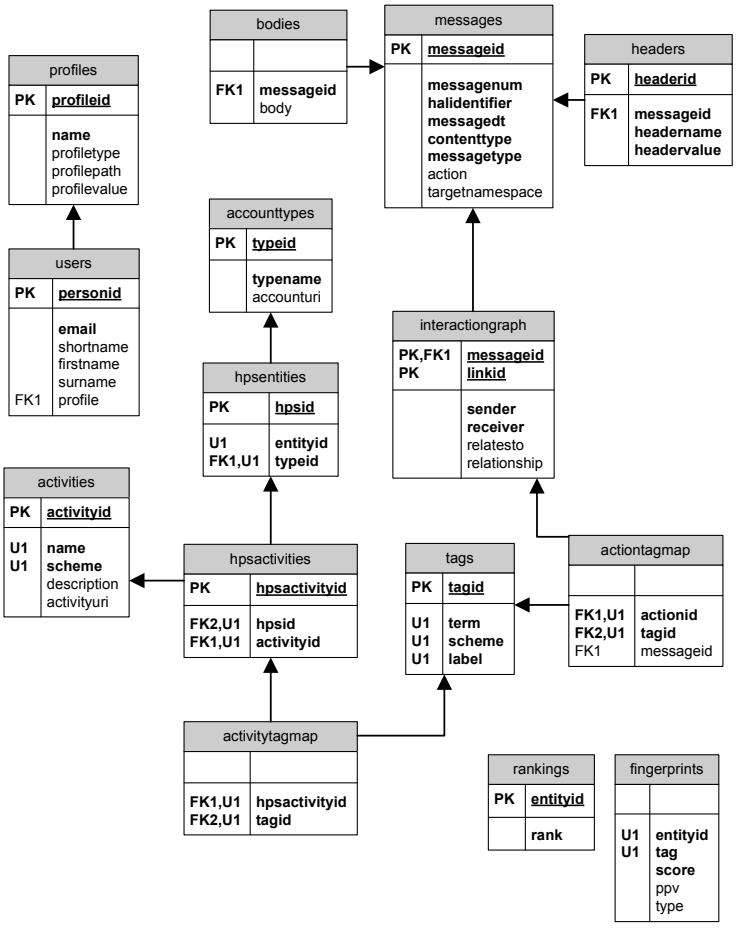


Figure C.1: Database schema designed for HAL.

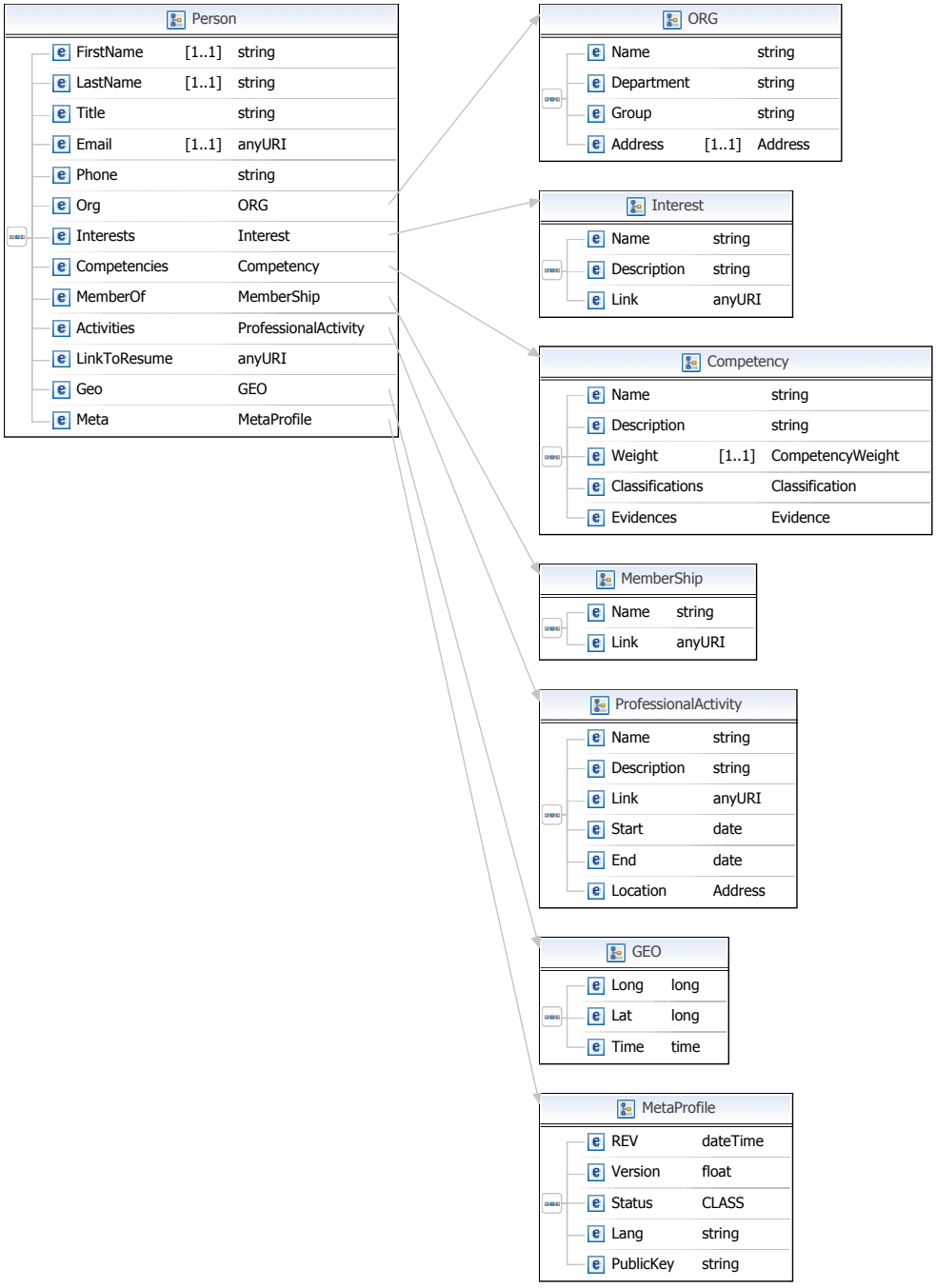


Figure C.2: Model for HPS profiles.

Appendix D

Weight Profiles

Metric weights can be adjusted based on the requirements of various collaboration use cases. We illustrate some HPS use cases and define a method to find useful metric weights based on

- Find expert for collaboration: emphasizing *long-term* collaboration scenarios. The dominant metric is SE. SE is a model to determine the skill level of a user.
- Require input from best available expert: this use case targets process-human interactions with the requirement of finding an expert who can provide input in a timely fashion.
- Require input from best informed expert: in this case we demand input or expertise from a user who is highly involved in a particular context. Thus, IIL plays an important role since intensities in a given context indicate best informed users.

We calculate the weights for different application scenarios as the area underneath a function which establishes the relationship between above mentioned queries. In a straightforward manner, we define linear functions to determine different levels of importance. Let us start with the basic definition of W_{M_R} as the integral over the entire set of metrics:

$$W_{M_R} = \int_{w_m} f(w_m)dw_m = \int_{w_{m1}} f(w_{m1})dw_{m1} + \ldots + \int_{w_{mn}} f(w_{mn})dw_{mn} \tag{D.1}$$

For linear $f(w_m)$, W_{M_R} simplifies to

$$W_{M_R} = \int_{w_m} (a \cdot w_m + b)dw_m = \frac{1}{3a}(a \cdot w_m + b)^2 \tag{D.2}$$

- *Immediate response from best available expert* yields equal importance for each metric ($b = 1/|M_R|$), *Find expert for collaboration* demands high importance in SE. The linear function is $2/a^2 + b$ (i.e., $a^2 = |M_R|$ and $b = 0 \mapsto 2/9 \cdot w_m$).
- *Require input from best available expert* demands equal importance in `availability`, SE (we abandon IIL) and *Require input from best informed expert* using $-2/a^2 + b$ (i.e., $a^2 = |M_R|$ and $b = 2/|M_R| \mapsto -2/9 \cdot w_m + 2/3$).

We provide calculations and descriptions in Table D.1. These queries can be extended based on different application requirements which can be modeled using other relationship functions. However, to discuss our approach to determine weight-profiles, it is sufficient to use linear models. The column SE depicts the sum of context-dependent SE rankings.

Profile ID	`availability`	IIL	SE	Description
01	0.5	0.5	n/a	Basic DSARank for dynamics in human interactions.
02	n/a	n/a	1	Skill-based DSARank. LL depends on user query and preferences vector.
03	1/3	1/3	1/3	Immediate response from best available expert.
04	1/9	1/3	5/9	Find expert for collaboration.
05	1/2	0	1/2	Require input from best available expert.
06	5/9	1/9	1/3	Require input from best informed expert.

Table D.1: Metric-weight profiles (piecewise integration over metric interval).

Appendix E

Task Rewarding Example

Let us define the input set $\{\vec{v}_{\alpha_1}\}$ obtained in M_{α_1} as $\{(0.3, 0.0202)_{s_1}, (0.5, 0.4273)_{s_2}, (0.7, 0.9921)_{s_3}, (0.9, 0.9999)_{s_4}\}$. Table E.1 shows the corresponding data sets that are calculated in M_{α_2} with reference to M_{α_1}.

	$\vec{v}_{\alpha_2}$	$\vec{p}_{\alpha_{1,2}}$	$\cos\varphi$	$\left\|\vec{p}_{\alpha_{1,2}}\right\|$	$\left\|(p_{\alpha_{1,2},x}, v_{\alpha_{2,y}})\right\|$	sc
s_1	$(0.8, 0.9723)$	$(0.3, 0.0795)$	0.784831	0.3330	1.0247	3.0774
s_2	$(0.2, 0.0154)$	$(0.5, 0.4539)$	0.781440	0.6655	0.4869	0.7316
s_3	$(0.9, 0.9922)$	$(0.7, 0.9548)$	0.998065	1.1958	1.2260	1.0252
s_4	$(0.6, 0.7343)$	$(0.9, 0.9977)$	0.995865	1.3685	1.1902	0.8697

Table E.1: Task data set in M_{α_2}.

Similarly, Table E.2 shows a data set in M_{α_3} with reference to M_{α_2}.

	$\vec{v}_{\alpha_3}$	$\vec{p}_{\alpha_{2,3}}$	$\cos\varphi$	$\left\|\vec{p}_{\alpha_{2,3}}\right\|$	$\left\|(p_{\alpha_{2,3},x}, v_{\alpha_{3,y}})\right\|$	sc
s_1	$(1.0, 0.9419)$	$(0.8, 0.8305)$	0.998713	1.1235	1.2082	1.0754
s_2	$(0.7, 0.7605)$	$(0.2, 0.1350)$	0.970740	0.2413	0.7864	3.2587
s_3	$(0.4, 0.3983)$	$(0.9, 0.8972)$	0.997610	1.2359	0.9387	0.7595
s_4	$(0.8, 0.8390)$	$(0.6, 0.6116)$	0.999919	0.8383	1.0162	1.2122

Table E.2: Task data set in M_{α_3}.

These datasets were generated for the task processing time PT $(0.2 \leq PT \leq 1.0)$[1]. The task reward is calculated by the functions $f_T \langle M_\alpha \rangle, \alpha \in \{\alpha_1, \alpha_2, \alpha_3\}$.

We assume that the task reward $\underline{tr}_{ij}(M_{\alpha_2})$ can be calculated with reference to $tr_{i-1j}(M_{\alpha_1})$ and $\underline{tr}_{i+1j}(M_{\alpha_3})$ with reference to $tr_{ij}(M_{\alpha_2})$. Note that it is also possible to calculate the trend $\underline{tr}_{i+1j}(M_{\alpha_3})$ based on $\underline{tr}_{ij}(M_{\alpha_2})$. The illustrated dataset, however, assumes a series of tasks such that $\exists\{tr_{i-1j}, tr_{ij}, tr_{i+1j}\}\forall j \in \{HT(s)\}$.

[1]The assumption is that $HT(s)$ are statistically independent.

To handle situations where such a series not always exists, we can calculate the *mean trend*. Let us define $\text{trend}_{\overline{\alpha}}(M_{\alpha_n}) = (\prod_1^N e^{\mathbf{i}\varphi})^{1/N}, N = |\{tr(M_{\alpha_n})\}|$ to calculate the mean trend as the mean of the angles φ. Thus, we have $\underline{tr}_{ij}(M_{\alpha_n}) = tr_{ij}\,\text{trend}_{\overline{\alpha}}(M_{\alpha_n})$.

Figure E shows rewards obtained in M_{α_1}, M_{α_2}, and M_{α_3}. In particular, M_{α_1} shows the rewards tr as defined in the set $\{\vec{v}_{\alpha_1}\}$. Similarly, $tr(M_{\alpha_2})$ is given by $\vec{v}_{\alpha_2}$ in Table E.1 and $\vec{v}_{\alpha_3}$ in Table E.2 denotes $tr(M_{\alpha_3})$. We see in Table E.3 the computation of $\underline{tr}$, given the aforementioned dataset. The divergence $\uparrow\downarrow$ is calculated as $(\text{trend}_{\overline{\alpha}} - e^{\mathbf{i}\varphi})/\,\text{trend}_{\overline{\alpha}}$.

		$sign(\log(c))$	$e^{\mathbf{i}\varphi}$	$\uparrow\downarrow$	tr	$\underline{tr}$	$\texttt{Rank}(tr)$	$\texttt{Rank}(\underline{tr})$
M_{α_2}	s_1	1	1.9511	-97%	0.9723	1.8970	3	4
	s_2	-1	0.5098	49%	0.0154	0.0078	1	1
	s_3	1	1.0642	-7%	0.9922	1.0559	4	3
	s_4	-1	0.9130	8%	0.7343	0.6705	2	2
M_{α_3}	s_1	1	1.0521	1%	0.9419	0.9909	4	4
	s_2	1	1.2744	-20%	0.7605	0.9692	2	3
	s_3	-1	0.9332	12%	0.3983	0.3717	1	1
	s_4	1	1.0128	5%	0.8390	0.8498	3	2

Table E.3: Calculating task trend for M_{α_2} and M_{α_3}.

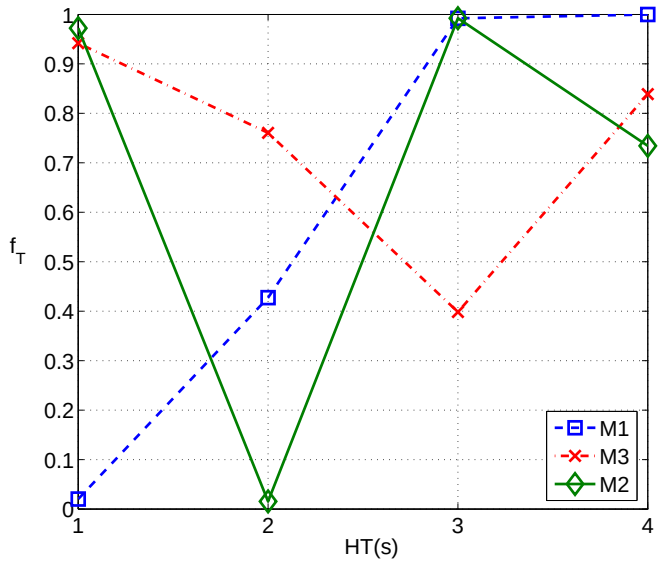


Figure E.1: Rewarding example to illustrate different models for $f_T\,\langle M_\alpha \rangle$.

Appendix F

Code Listing Ranking Service

F.1 Experimental Implementation

Setting up environment for mathematical analysis: We use Matlab as a tool to test and evaluate ranking algorithms. Matlab offers a scripting environment and thus makes it easy to run experiments and evaluate different ranking strategies. The Matlab environment can be configured so researchers can interact with data stored in an SQL database. In particular, we configured and used the MySQL database in our experiments[1]. We can then execute commands, for example SQL queries, by using `mysql` function.

Setting up database: The next step is to setup a suitable database schema for experimental evaluation of interaction-based ranking algorithms. Listing F.1 shows two tables containing information regarding interaction links and a table holding rankings information. The interaction links table captures service interaction links (invocations) and describes such interactions as a *graph*.

```
>> mysql('desc interactionlinks')

  Field          Type            Null   Key   Default   Extra
+-------------+ +--------------+ +----+ +---+ +--------+ +-----+
  serviceid      varchar(45)     NO
  consumerid     varchar(45)     YES
  weight         double          YES
  timestampe     varchar(45)     YES

>> mysql('desc rankings')

  Field          Type            Null   Key   Default   Extra
+-------------+ +--------------+ +----+ +---+ +--------+ +-----+
```

[1] http://www.mmf.utoronto.ca/resrchres/mysql/

```
serviceid    varchar(45)    NO     PRI
rank         double         YES
timestamp    varchar(45)    YES
```

Listing F.1: Database tables.

F.2 Ranking Algorithm in Matlab

We now disuss an experimental implementation of a PageRank-based ranking algorithm that analysis and calculates the rank of a services based on captured interaction links.

```
% compute stationary distribution of markov chain
% using power iteration
for n = 1:length(serviceids)
  ranksum = 0.0
  id = serviceids(n,1)
  q = sprintf('select consumerid from interactionlinks
                    where serviceid = ''%s''',
                    sprintf('%s',id{:}))
  consumers = mysql(q)
  for m = 1:length(consumers)
    consumerid = consumers(m,1)
    fromrank = 0.0
    q = sprintf('select rank from rankings
                    where serviceid = ''%s''',
                    sprintf('%s',consumerid{:}))
    fromrank = mysql(q)
    q = sprintf('select count(*) from interactionlinks
                    where consumerid = ''%s''',
                    sprintf('%s', consumerid{:}))
    indeg = mysql(q)
    if length(fromrank) == 0
      ranksum = ranksum + 0.85 * (0.125 / indeg)
    else
      ranksum = ranksum + 0.85 * (fromrank / indeg)
    end
  end

  u = sprintf('update rankings SET rank = %d, timestamp = ''%s''
                    where serviceid = ''%s''', ranksum, 'empty',
                    sprintf('%s',id{:}))
  mysql (u)
```

```
32 end
```

Listing F.2: Matlab code ranking algorithm.

Plotting rankings results in Matlab.

```
1 % query for rankings
2 q  = sprintf('select serviceid, rank from rankings')
3 % select from rankings
4 [sid,rank] = mysql(q)
5 % plot as discrete sequence data
6 stem (rank, 'DisplayName', 'rank');
```

Listing F.3: Plot rankings script.

F.3 Ranking Algorithm in Java

```
public class ServiceRankAlgorithm {

  public void evaluate(int numIterations) {
4   if (isRunning() == true)
      return;
    else {
      updateStatus(1);
    }
9   init();

   try {
      createConnection();
      statement = connection.createStatement();
14
      for (int iteration = 0; iteration < numIterations; iteration++) {
        ResultSet result = statement
            .executeQuery("select serviceid from "
                + Database.RANKINGS);
19
        ArrayList<String> services = new ArrayList<String>();

        while (result.next()) {
          services.add(result.getString(1));
24      }
        result.close();

        for (String s : services) {
```

```
        double ranksum = 0.15;

        result = statement
            .executeQuery("select consumerid from "
                + Database.LINKS + " where serviceid = '"
                + s + "'");

        ArrayList<String> consumers = new ArrayList<String>();

        while (result.next()) {
          consumers.add(result.getString(1));
        }
        result.close();

        for (String consumer : consumers) {
          double fromrank = 0.0;
          result = statement.executeQuery("select rank from "
              + Database.RANKINGS + " where serviceid = '"
              + consumer + "'");
          if (result.next()) {
            fromrank = result.getDouble(1);
          } else {
            fromrank = 0.125;
          }

          result = statement.executeQuery("select count(*) from "
              + Database.LINKS + " where consumerid = '"
              + consumer + "'");

          if (result.next()) {
            ranksum += 0.85 * (fromrank / result.getInt(1));
          }

          result.close();
        }

        statement.executeUpdate("update " + Database.RANKINGS
            + " SET rank = " + ranksum + " , timestamp = '"
            + System.currentTimeMillis()
            + "' where serviceid = '" + s + "'");
      }
    }

    statement.close();
    closeConnection();
```

```
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
      updateStatus(0);
    }
  }

  private void updateStatus(int status) {
    try {
      createConnection();
      statement = connection.createStatement();
      statement.executeUpdate("update " + Database.HISTORY
          + " set running = " + status);
      statement.close();
      closeConnection();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  private boolean isRunning() {
    boolean running = false;
    try {
      createConnection();
      statement = connection.createStatement();
      ResultSet result = statement
          .executeQuery("select running from " + Database.HISTORY);

      if (result.next() && result.getInt(1) == 1)
        running = true;

      result.close();
      statement.close();
      closeConnection();
    } catch (Exception e) {
      e.printStackTrace();
    }
    return running;
  }

  private void init() {
    try {
      createConnection();
      statement = connection.createStatement();
```

```
      statement.executeUpdate("delete from rankings");

      ResultSet result = statement
          .executeQuery(
            "select distinct serviceid from " + Database.LINKS);
      ArrayList<String> services = new ArrayList<String>();

      while (result.next()) {
        services.add(result.getString(1));
      }
      result.close();

      for (String s : services) {
        statement.executeUpdate("insert into " + Database.RANKINGS
            + " values ('" + s + "' , " + 1.0 + " , '"
            + System.currentTimeMillis() + "')");
      }

      result.close();
      statement.close();
      closeConnection();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void generateTestData(int vertexCount, int iterations) {
    try {
      ArrayList<String> nodes = new ArrayList<String>();

      for (int i = 0; i < vertexCount; i++) {
        nodes.add(UUID.randomUUID().toString());
      }

      createConnection();
      statement = connection.createStatement();

      for (int i = 0; i < iterations; i++) {
        Random rand = new Random();
        int sindex = rand.nextInt(nodes.size());
        int cindex = rand.nextInt(nodes.size());

        if (sindex == cindex) {
          continue;
        }
```

```
164         String service = nodes.get(sindex);
            String consumer = nodes.get(cindex);

            ResultSet result = statement.executeQuery("select * from "
                + Database.LINKS + " where serviceid = '" + service
169             + "' and consumerid = '" + consumer + "'");

            if (result.next()) {
              continue;
            }
174
            statement.executeUpdate(
                "insert into " + Database.LINKS + " values ('"
                + service + "' , '" + consumer + "' , " + 1 + " , '"
                + System.currentTimeMillis() + "')");
179       }

          statement.close();
          closeConnection();
        } catch (Exception e) {
184       e.printStackTrace();
        }
      }
    }
```

Listing F.4: Basic ranking algorithm

F.4 Ranking Web Service

```
public class RankingService {
  public Result getResult(String serviceId) {
3   Result result = null;

    for (Result myResult : new ServiceRankResults().getResults()) {
      if (result.getServiceId().equalsIgnoreCase(serviceId))
        result = myResult;
8   }
    return result;
  }

  public Result[] getResults() {
13  return new ServiceRankResults().getResults();
  }
```

```
  public void updateRankings() {
    new ServiceRankAlgorithm().evaluate(20);

    new Thread() {
      public void run() {
        new ServiceRankAlgorithm().evaluate(20);
      }
    }.start();
  }
}
```

Listing F.5: Basic ranking service

```
public class ServiceRankResults {
  public Result[] getResults() {
    ArrayList<Result> resultList = new ArrayList<Result>();

    try {
      createConnection();
      statement = connection.createStatement();

      ResultSet result = statement.executeQuery(
          "select * from "
          + Database.RANKINGS + " order by rank desc");

      int pos = 0;
      while (result.next()) {
        Result rankingResult = new Result();
        rankingResult.setServiceId(result.getString(1));
        rankingResult.setPosition(pos++);
        rankingResult.setScore(result.getDouble(2));
        resultList.add(rankingResult);
      }

      for (Result rankingResult : resultList) {
        result = statement.executeQuery(
            "select count(*) from " + Database.LINKS
            + " where serviceid = '" + rankingResult.getServiceId()
            + "'");

        if (result.next()) {
          rankingResult.setIndegree(result.getInt(1));
        }
      }
```

```
        result.close();

        statement.close();
        closeConnection();
      } catch (Exception e) {
        e.printStackTrace();
      }

      return resultList.toArray(new Result[resultList.size()]);
    }
}
```

Listing F.6: Ranking service result