Programming Human and Software-Based Web Services

Daniel Schall and Schahram Dustdar, *Vienna University of Technology*

M. Brian Blake, *University of Notre Dame*

Human-provided services harness human capabilities within service-oriented environments while leveraging Web 2.0 innovations.

The Web has evolved from a distributed repository of content to an interactive medium in which end users actively shape the availability of information and services. Part of this evolution is often called Web 2.0 and characterized by the emergence of knowledge sharing and online service composition platforms.

The transformation of how people interact on the Web has been poorly leveraged in existing SOA (service-oriented architecture)-based systems. In traditional composition scenarios, services are created from the top down, without considering the availability and preferences of people, constraints and relationships, and the support of dynamic, ad hoc collaborations.

We propose a new programming paradigm that utilizes human capabilities as computational Web services. People create and share *human-provided services* (HPSs) to indicate their incentive and availability to participate in such collaborations.

DISTRIBUTED HUMAN COMPUTATION

The Web's user-centric nature has led to an unusual role for people in information systems—more often than not, certain problems that are hard for software services to solve are outsourced to humans. Consequently, researchers have introduced the notion of *distributed human computation* in the context of AI-complete problems such as analyzing and tagging images (C. Gentry, Z. Ramzan, and S. Stubblebine, "Secure Distributed Human Computation," *Proc. 6th ACM Conf. Electronic Commerce,* ACM Press, 2005, pp. 155-164).

Human-based computation platforms can be found in both large-scale Web systems and closed enterprise systems. For example, Google Image Labeler (http://images.google.com/imagelabeler) relies on humans' creative ability to label images, which improves the quality of image search results (L. von Ahn and L. Dabbish, "General Techniques for Designing Games with a Purpose," *Comm. ACM,* Aug. 2008, pp. 58-67). Another platform more closely related to human data retrieval is Amazon Mechanical Turk (www.mturk.com), which lets task requesters access a flexible (resizable) human workforce but doesn't support collaborative activities and interactions among workers.

In 2007, the WS-HumanTask (WS-HT) and BPEL4People (B4P) standards introduced models for weaving human interactions into SOA-based compositions. WS-HT and B4P target workflow-based coordination in SOA/Web services environments in enterprise settings. However, they lack the ability to create flexible compositions of human and software-based services. Related B4P standards specify languages for modeling human interactions, the life cycle of human tasks, and generic role models (F. Leymann, "Workflow-Based Coordination and Cooperation in a Service World," *On the Move to Meaningful Internet Systems 2006,* LNCS 4275, Springer, 2006, pp. 2-16).

Compositions and processes are modeled using a language such as the Business Process Execution Language (BPEL)—a widely used and well-accepted composition language in the Web services domain—and executed in the actual environment where the composition model is deployed. These top-down composition models are limited in their use of context and adaptive control and thus fail to deliver the most effective runtime behavior.

Published by the IEEE Computer Society

In previous work, we introduced the HPS framework, emphasizing the importance of flexible interaction models (D. Schall, H.-L.Truong, and S. Dustdar, "Unifying Human and Software Services in Web-Scale Collaborations," *IEEE Internet Computing,* May 2008, pp. 62-68). The framework must be extended by combining top-down and bottom-up composition models.

Not every interaction or task may be known at design time (S. Dustdar, "Caramba—A Process-Aware Collaboration System Supporting Ad Hoc and Collaborative Processes in Virtual Teams," *Distributed and Parallel Databases,* Jan. 2004, pp. 45-66); thus not all interaction links between services and people can be established a priori. As such, an adaptive composition of human and software services is a strong requirement.

TOWARD UNIFIED HUMAN AND SOFTWARE SERVICES

Any user-centric, participative perspective is characterized by a bottom-up design of services and the emergence of social relations, knowledge, and expertise. The term *interaction model* thus has different meanings depending on the system perspective.

Technical interaction 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.

Dynamic systems exhibit rules or models of cooperation—that is, how interactions operate. For 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 among related actors (M.A. Nowak, "Five Rules for the Evolution of Cooperation," *Science,* 8 Dec. 2006, pp. 1560-1563). Interactions and relations emerge and change over time.

The complexity and dynamics of interactions make the design of heterogeneous compositions comprising human and software services a significant problem. A set of building blocks is needed to support the seamless integration of human capabilities in SOA. Such building blocks are important to enable systems to evolve with respect to local interactions. Global composition models are essential to optimize and control processes.

The benefit of human-provided services is a seamless service-oriented infrastructure of human and software-based services.

Building blocks

We propose four building blocks to achieve convergence of ad hoc and formalized process models.

Design. Users should be able to create and share services based on their preferences to work on collaborative activities. The main difference between the design of HPSs and traditional software services in the SOA landscape is that the end user actively creates services in a manner similar to using mashup platforms. The user creates HPSs from scratch or discovers and reuses existing service definitions shared within communities.

Interaction patterns. A particular HPS may work on tasks in the context of an activity by interacting with other HPSs. It accomplishes this by delegating tasks (subtasks) to successfully deliver the demanded output of an HPS-based composition. Such a composition may be defined in an ad hoc manner by the person responsible for the corresponding task. These interaction patterns may be discovered dynamically during runtime based on logging and monitoring mechanisms.

Reputation. As in many collaboration systems, some users contribute more toward certain objectives than others. It's thus desirable to measure the user's reputation within systems such as organizations, communities, or social networking platforms. An expertise recommendation algorithm shouldn't treat all interactions with equal importance. Interactions are in many cases performed in a certain context—for example, within the scope of a certain activity.

Discovery. HPSs can be discovered by accessing a service registry—the late binding mechanism in a service-centric environment. In homogenous software service environments, compositions are realized by discovering and selecting services based on quality-of-service (QoS) information. In mixed service-oriented environments comprising HPSs and software services, selection should be based on users' trust, reputation, and expertise (F. Skopik, D. Schall, and S. Dustdar, "Modeling and Mining of Dynamic Trust in Complex Service-Oriented Systems," to appear in *Informations Systems*, 2010).

Service compositions

Figure 1 shows how the HPS paradigm applies to service compositions.

Composition model. The process model shown on the left side of the figure comprises a set of activities to coordinate interactions among services. Human actors may need to realize some process activities.

A BPEL expert creates a composition model at design time, defining the services that are part of a composition as well as the interactions among them. The expert embeds human capabilities in the system by modeling human tasks.

Role models restrict the set of people who can work on tasks. However, while role models are sufficient to preselect potential qualified actors, they're static and thus don't capture

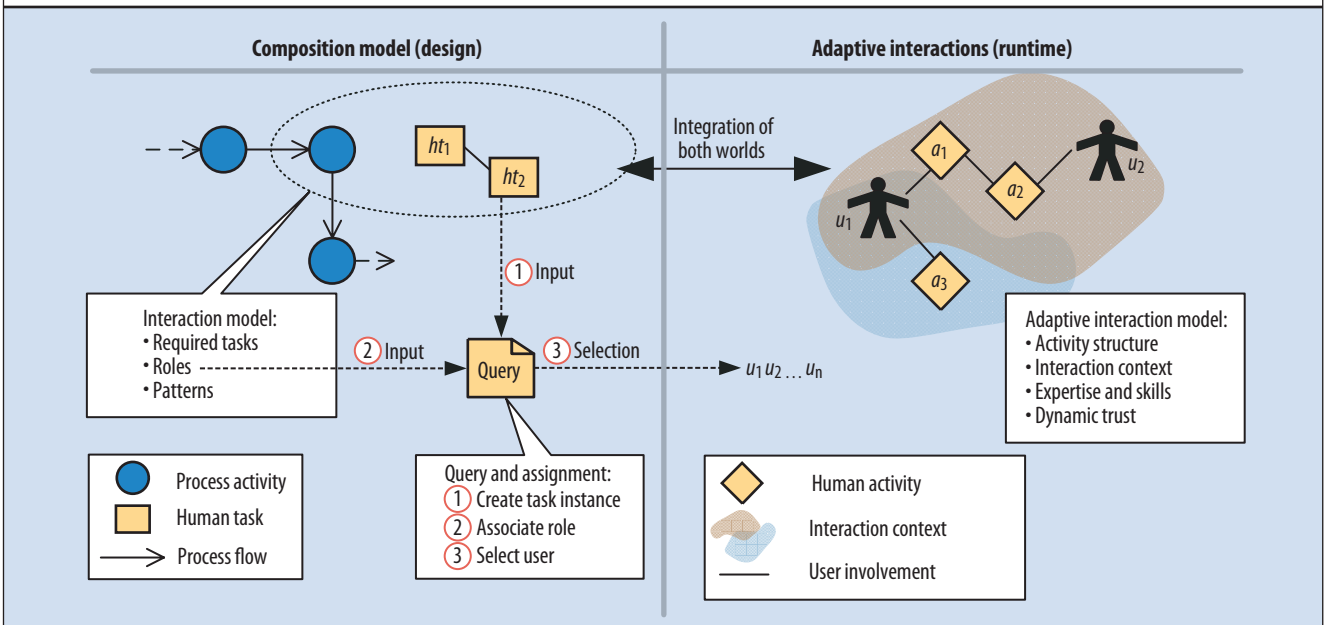

Figure 1. Applying the human-provided services (HPS) paradigm to service compositions.

intrinsic properties arising in real environments. Users with the same role and formal set of skills may have very different interests and levels of expertise. Thus, nonfunctional properties (reputation mechanisms) must be considered as well as functional properties.

B4P supports late binding by defining abstract people groups. If an actor fails to deliver the desired output in a satisfactory manner—for example, due to lack of actual skills and expertise—the entire composition may not be able to deliver its result. The BPEL expert defining the process should be able to associate service-level agreements (SLAs) to the query influencing the selection of HPSs in the environment. This implies a QoS model for HPS.

Ad hoc interactions. While in B4P the end user's role is limited to the process's runtime aspects, the major challenge is adapting to the fluidity and changing preferences of users. For example, the four-eyes principle denotes that an authority must approve certain critical decisions to prevent mistakes or malicious behavior (M. Kloppmann et al., "WS-BPEL Extension for People—BPEL4People," white paper, IBM and SAP, July 2005).

Along with the person processing a task, a second person—the monitor—observes the task's progress or output. Reputation and trust between people may influence how monitoring occurs. If the selected user working on a certain task has low reputation, ad hoc activities and notifications can be created dynamically to include other users acting as monitors.

The right side of Figure 1 illustrates dynamic, activity-centric interactions that can be used to realize monitoring patterns. First, user u_1 may be selected as part of a predefined composition. Depending on u_1's actual expertise and reputation, another trusted actor u_2 can be associated as monitor. Expertise, reputation, and resulting interaction patterns are highly context-sensitive, depending on the actual environment and its properties.

HUMAN-PROVIDED SERVICES

The HPS framework assimilates the Web 2.0 paradigm wherein the end user designs and provides services (D. Schall, "Human Interactions in Mixed Systems—Architecture, Protocols, and Algorithms," PhD dissertation, Faculty of Computer Science, Vienna Univ. Technology, 2009; www.ub.tuwien.ac.at/diss/AC05039868.pdf). As Figure 2 shows, the HPS reference architecture consists of three essential layers: data collection, services, and middleware.

Data collection

The data collection layer includes a registry, which makes available all services used in applications or by users. A State and Agreements module maintains the state of interactions—for example, in-progress or aborted—as well as documents QoS performance and SLAs. This layer also contains an Interaction History module based on collected logs.

Services

The services layer represents the set of services offered by people (HPSs) that can be discovered in a manner similar to traditional software services. HPSs require features such as asynchronous messaging, GUI representations, and the automatic generation of descriptive interfaces using, for example, the

Web Services Description Language (WSDL). The software stack needed for HPSs can be deployed on mobile devices to support pervasive interactions. Also, the middleware platform can host services so that users can manage HPSs online.

The service bus within this layer provides the backbone messaging infrastructure for interactions among services and humans. It provides features for service discovery, invocation, and transformations due to the heterogeneity of different types of services.

Middleware

The middleware layer contains four main modules.

The Collective Design module includes tools for designing HPS interfaces based on human activities as well as a tagging model for activities and services to recommend potentially useful (reusable) HPS interfaces. Service design can be imagined as the definition of a form comprising elements such as input fields, enumerations, lists, and so on. The definition is transformed into complex data structures (type system), Web service interfaces, and XML-based forms (XForms). HPS definitions can be shared so that others can offer the same type of service.

The Protocol Layer is used to automate interactions in a certain context. The user can specify rules to trigger automatic actions based on conditions and thresholds. For example, given a user's current workload, certain tasks may need to be delegated to available HPSs.

The Mining and Discovery module includes techniques to identify interaction patterns, relationships, and actor dependability in the system. The automated computation of reputation and expertise is a suitable way to track changing user preferences and interests. The monitoring pattern between HPSs is an example of a technique used to determine actor dependability.

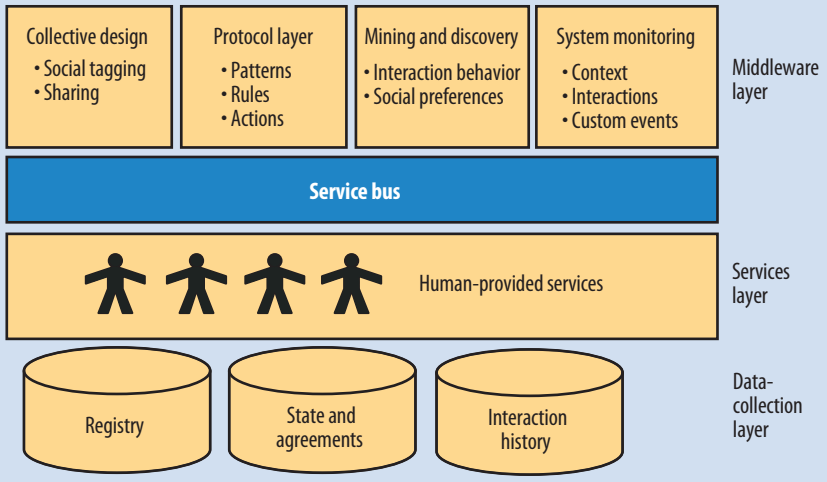

Figure 2. The HPS reference architecture consists of three essential layers: data collection, services, and middleware.

The System Monitoring module relies on interaction logs to observe the system's actual runtime state. These logs are based on Web service calls, lookup requests, and events triggered by collaboration services.

The increasing complexity of interactions, distribution of services, and end-user introduction of content on the Web requires models and languages for service composition. Both bottom-up (user-centric) and top-down (process-centric) interaction models are required in these dynamically changing environments. Human-provided services harness human capabilities within service-oriented environments while leveraging Web 2.0 innovations such as tagging mechanisms and tools to create service and data mashups. The user can define interaction interfaces following the same principles, thereby avoiding the need for parallel systems of software services and HPSs.

While we have focused on the layers responsible for the HPS framework's technical aspects, we suggest an additional architectural layer based on the social aspect of service design. This social layer is a mixture of user-centric services to engage in different Web-based interaction scenarios, community structure, social interest, and evolution. Furthermore, the HPS paradigm must be enhanced with a greater understanding of replaceability strategies between HPSs and software services.

Daniel Schall *is a postdoctoral researcher in the Distributed Systems Group, Information Systems Institute, Vienna University of Technology. Contact him at d.schall@infosys.tuwien.ac.at.*

Schahram Dustdar *is a professor of informatics and heads the Distributed Systems Group, Information Systems Institute, Vienna University of Technology. Contact him at dustdar@infosys.tuwien.ac.at.*

M. Brian Blake *is a professor of computer science and engineering and associate dean of the College of Engineering at the University of Notre Dame. Contact him at m.brian.blake@nd.edu.*

Editor: Simon S.Y. Shim, Dept. of Computer Engineering, San Jose State Univ., San Jose, CA; simon.shim@sjsu.edu