DOI: 10.1145/1400214.1400241

BY QUAN Z. SHENG, BOUALEM BENATALLAH, AND ZAKARIA MAAMAR

User-Centric Services Provisioning in Wireless Environments

THE PROLIFERATION OF MOBILE DEVICES (for example, PDAs, 3G mobile phones) and the deployment of more sophisticated wireless communication infrastructures are empowering the Web with the ability to deliver data and services to mobile users. New mobile applications will take advantage of ubiquitous wireless networking to create virtual worlds, with which we can interact while walking, driving our cars, or riding public transport.[4] Moreover, like stationary users, mobile users also require integrated access to relevant services to achieve complicated goals. For example, a class assistance service is relevant to college students, which helps them manage their class activities using mobile devices, by integrating services like attendance reminder and question post.

However, several obstacles still hinder the seamless provisioning of services in wireless environments. On the one hand, current Web service provisioning techniques that assume a strong Internet access (for example, fast, low latency, reliable and durable network connections), are inappropriate because wireless environments possess distinguishing features and inherent limitations such as low throughput and poor connectivity of wireless networks, limited computing resources, and frequent disconnections of mobile devices. On the other hand, the environments that mobile users interact with are generally highly dynamic. The variability in computing resources, display terminals, communication channels, and user conditions and preferences require applications to be *context aware* so that they can adapt to rapidly changing conditions.[8]

To date, services provided for mobile users are still hard to build. Two main challenges need to be considered. The first challenge is about the personalized access to services. Personalized support becomes even more crucial, when access of services takes place in wireless environments. For example, the access to services by mobile users tends to be *time* and *location* sensitive, meaning that mobile users might need to invoke particular services in a certain period of time and/or a certain place. In addition, mobile users require *integrated* access to relevant services because many of people's daily activities are *not* independent. A service provisioning environment allowing personalized discovery, selection, and composition of services is therefore needed.

The second challenge is about handling limited resources of mobile devices. Mobile devices posses, to a certain extent, limited resources (for example, battery power and input capabilities). Therefore, mobile devices better act as *passive listeners* (for example, receiving the results) than as active tools for service invocation, so that the computational power and battery life of the devices can be extended.

We developed a multi-agent based architecture that aims at providing a distributed, adaptive, and context-aware platform for personalized service

provisioning, which takes into account the needs of mobile users. The foundation of our approach is to enable an effective access to integrated services by combining technologies such as Web services, multi-agent systems, and publish/subscribe systems. In this article, we review the design principles, the architecture, and the implementation of the prototype system.

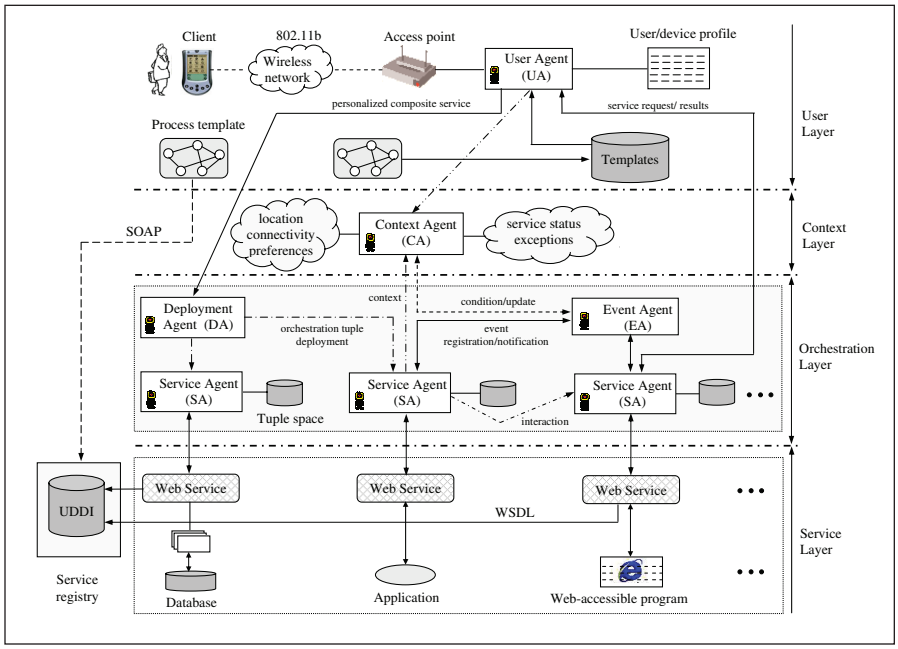
Design Principles

Leveraging Web services and software agents in combination with publish/subscribe systems provides the foundation to enable effective access to integrated services in wireless environments.

Web Services. Web services provide the pillars for evolving the Internet into a service-oriented integration platform of unprecedented scale and agility. The foundation of this platform lies in the modularization and virtualization of system functions and resources as services that: can be described, advertized and discovered using (XML-based) standard languages, and interact through standard Internet protocols. The Web services architecture provides building blocks to enable secure and reliable transactions that is vendor, platform, and device independent, which brings about the convergence of wired and wireless applications and services. Since Web services are described and interacted in a standardized manner, the task of developing complex applications by composing other services is considerably simplified.[9]

Agents. Agents are software entities that exhibit certain autonomy when interacting with other entities.[12] Agents use their internal policies and knowledge to decide when to take actions that are needed to realize a specific goal. Internal policies can use context information (for example, user preferences, device characteristics, and user location) to enable agents to adapt to different computing and user activities. Agents can also be used to proactively perform actions in dynamic environments. In fact, the combination of services and agents will provide a self-managing infrastructure. Agents extend services by embedding extensible knowledge and capabilities (for example, context aware execution and exception handling policies) making them capable of providing personalized and adaptive service provisioning in dynamic environments.

Figure 1.



Publish/Subscribe System. The publish/subscribe paradigm offers a communication infrastructure where senders and receivers of messages interact by producing and consuming messages via designated shared spaces.[2] The communication is *asynchronous* in the sense that it completely decouples the senders and receivers in both space and time. This enables mobile users to disconnect at any time—either voluntarily to save for example, communication cost and battery power, or involuntarily due to breakdowns of connections—and re-synchronize with the underlying infrastructure upon reconnection. This communication paradigm has been identified as an ideal platform for a variety of Internet applications, especially in wireless environments.

System Design

We propose a layered, multi-agent based architecture to provide support for integrated Web services specification, deployment, and execution. Figure 1 shows the elements of this architecture, which are grouped in four layers. The architecture uses five types of agents, namely *user agent, service agent, deployment agent, event agent,* and *context agent*. These agents can engage in cooperative interactions to perform the operations related to Web services composition.

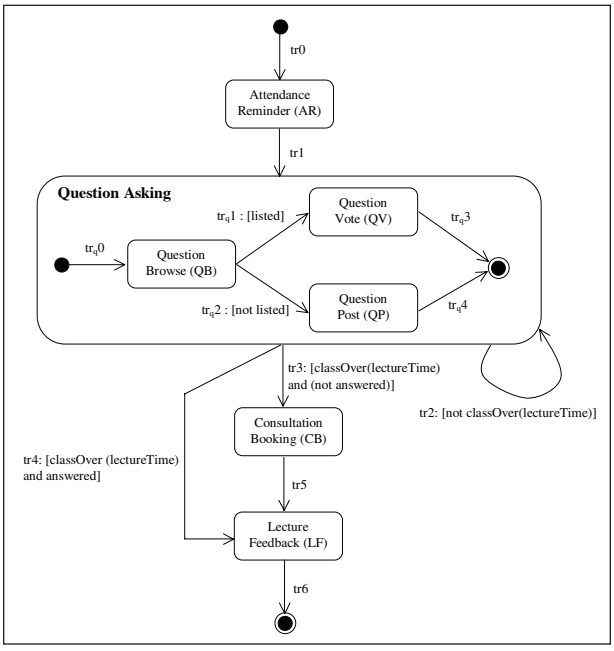
User Layer. The user layer gives mobile users access to service provisioning environment through two main components, namely *client* and user agent. The client is an application that can be downloaded and runs on mobile devices. It provides users with an interface for specifying user activities, and interacting with the user agent. The client interacts with the user agent via access points in the wireless networks (for example, IEEE 802.11b).

Users' activities (for example, class attendance) are usually complex. The fulfilment of an activity may call for multiple services executed in a specific chorology. It is too tedious to specify activities from scratch through small devices like PDA, which have limited input capabilities. To ease the activity specification process, we introduce the notion of *process templates*.[10] A process template is a reusable business process skeleton that corresponds to recurrent user needs (for example, managing class activities). It is made up of tasks (for example, attendance reminder), data/control flow dependencies between tasks, and exception handling policies.

We specify process templates with statecharts.[a] Succinctly, a statechart is made up of states and transitions. States can be basic or compound. A basic state (also called task) corresponds

a Process templates developed in statecharts can be adapted to other process definition languages like BPEL4WS (http://dv2dev.bea.com/techtrack/BPEL4WS.jsp).

Figure 2.



to the execution of a Web service. Compound states enclose one or several statecharts within them.

Figure 2 is a simplified classAssistant process template that models the common routine about student class attendance. Firstly, an attendance reminder notifies students about a lecture's time and place so that the class will not be missed. During the lecture, when a student wants to submit questions via her PDA, she browses the questions asked by other students and decides to either vote for a posted question (if a similar question was already asked), or post her question (if no one has asked yet). The student may ask several questions during the lecture. After the class, a consultation booking might be requested and a feedback on the lecture is provided by the student.

The client allows users to define their activities by specifying *personal preferences* (for example, temporal/spatial constraints, data supply/delivery preferences) over the tasks of process templates, thereby defining *personalized composite services*. A user can specify *temporal* and *spatial* constraints for each task, which respectively indicate *when* and *where* the user wants to have a task executed. For example, a student can specify that the question asking services (for example, Question Vote service) can be executed only during the lecture time and when she is in the classroom. Meanwhile, data supply and delivery preferences are related to supplying values to the input parameters and delivering the values of output parameters of a task. A user can specify that the value of a task's input parameter should be obtained from her profile so that she does not have to provide the value manually. Similarly, she can also specify that the value of a task's output parameter should be sent to her user agent in case she wants to know the results.

A user agent (UA) acts on behalf of a specific mobile user. The UA maintains profile information including the user's contact information, personal preferences, and mobile device characteristics. Based on the profile, the UA can identify data formats and interaction protocols for communicating with the client that runs on the mobile device.

The UA subscribes to the template repository where all the process templates are stored, which in turn notifies the UA about the availability of each new process template. Upon receiving the notification, the UA prepares a short description outlining the functionalities and potential charges of the process template and sends it to the user in the notification message (for example, SMS). If the user is interested in the process template, the UA will contact the template repository to deliver an XML document of this process template, which is going to be stored in the user's mobile device upon reception for later configuration.

The client submits a configured process template (i.e., user's personalized composite service) to the UA, which is responsible for executing the service, collecting service results, and delivering results to the user. The UA accomplishes this by interacting with other agents in the architecture.

Service Layer. This layer consists of a collection of services that service providers develop, deploy, and maintain. For example, a university may provide services like class attendance reminder, question vote, and consultation booking, for its students. These services are typically proprietary/native applications (for example, J2EE or CORBA based applications, legacy applications), and are exposed as Web services for dynamic location, composition, and flexible invocation.

There exist tools for the automatic generation of WSDL descriptions from proprietary services. For example, the Java2WSDL utility in Apache Axis automatically generates WSDL descriptions from Java class files. The services publish their descriptions in a UDDI registry. The registration includes the URLs for communicating with the services and pointers to their WSDL descriptions. Services communicate (such as, advertise, discover, and invoke) via SOAP messages.

Context Layer. The context layer contains a context agent (CA) that collects and disseminates context information. Currently, the CA maintains three kinds of context, namely *user context, device context,* and *service context*. The user context includes information related to a user (for example, preferences, calendar, and location). The device context includes hardware and software characteristics of the user's devices. While the service context includes information related to a service. For example, executionStatus is a service context showing the current execution status of a Web service.

The CA collects context information from *context providers*, which can be a user, a service, or a third party entity. The CA consists of a set of configurable *context collectors*. Each context collector handles one type of context information and encapsulates the details of interactions with the context provider for that information (for example, the context collector pulls the context information periodically from the context provider).

Orchestration Layer. The orchestration layer consists of a set of agents that collaborate among each other for the robust and context-aware execution of composite services in wireless environments.

Orchestration Tuples. The orchestration of a composite service is encoded in the form of *orchestration tuples*. Orchestration tuples are expressed as event-condition-action (E[C]/A) rules specifying the actions that must be per-

formed when specific events occur and when the conditions hold. It should ne noted that process templates developed in statecharts can be adapted to other prcess definition languages like BPEL4WS (http://dev2dev.bea.com/techtrack/BPEL4WS.jsp). We introduce three types of orchestration tuples to coordinate the execution of personalized composite services:

- *Precondition tuples* specify the conditions that need to be satisfied before the execution of a service,
- *Postprocessing tuples* specify the actions that need to be performed after the execution of a service, and
- *Exception handling tuples* specify the instructions that dynamically react to run-time exceptions (for example, mobile device disconnection and services failures).

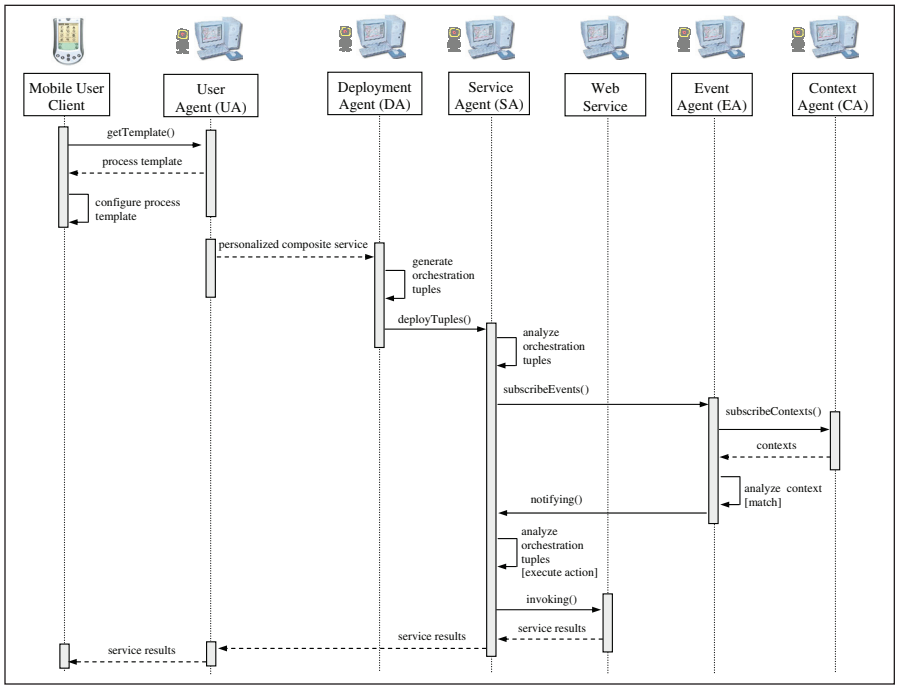
For example, an exception handling tuple unpresentable(r,d)[true]/transform(r,TS,d) indicates that if the service result r can not be displayed in the users current device d, the result will be sent to TS, a transformation service, for adaptation.

The orchestration tuples of a composite service are statically derived by analyzing the information encoded in the statechart of the service (for example, control flow and data dependencies, exception handling policies, and personal preferences).

Orchestration Agents. The orchestration layer consists of a set of agents, facilitating the asynchronous, distributed, and context-aware execution of composite services. The agents include *deployment agent, event agent,* and *service agents*. The deployment agent (DA) generates orchestration tuples from the specification of personalized composite services, which are submitted from the user agent (UA). The DA then deploys (uploads) these orchestration tuples into the *tuple spaces* of the corresponding services.

Service agents (SAs) act as proxies for Web services, monitoring and coordinating service executions. The knowledge required by an SA is a set of orchestration tuples, stored in a tuple space associated with the SA. The orchestration tuples are generated and deployed by the DA. There is one SA per service. For each Web service in the service layer, the service administrator needs to download and install an SA, together with a tuple space.

Figure 3-a



The event agent (EA) is responsible for disseminating events. The EA maintains the information of events supported by the platform, i.e., for a specific event, what context attributes are relevant to this event and what condition should be satisfied to fire the event. For example, event failed(s) indicates that an execution failure of service s has occurred. The related context of this event is executionStatus and the condition, executionStatus="failed", should be satisfied in order to fire the event.

Orchestration Interactions. Figure 3 (a) is a sequence diagram showing the process of the orchestration of personalized composite services. Firstly, the DA takes as input the specification of a personalized composite service from the UA, generates orchestration tuples for each task of the personalized composite service, and injects these orchestration tuples into the tuple spaces of the corresponding SAs. Then, SAs parse the orchestration tuples and retrieve relevant information (for example, events, conditions, and actions). The events (for example, lowBattery) are registered to the EA, which in turn subscribes relevant conditions (for example, batteryRemaining < 15%) to the CA. The EA fires and distributes events if the corresponding conditions are matched (for example, when the current battery capacity of the device is less than 15% of its full battery power). Upon receiving the notifications (i.e., the occurrence of the events) from the EA, the SAs extract the corresponding orchestration tuples from the associated tuple spaces, evaluate the conditions, and perform the proper actions (for example, service invocation in the diagram).

It should be noted that our current design is targeted at relatively small scale domains like university campuses, smart workplaces, and hotspots. To support a larger scale environment, multiple CAs and EAs need to be deployed. Mechanisms for managing events and contexts among such agents are topics of our ongoing research.

Implementation and Evaluation

To test the applicability of our architecture, we implemented a prototype system. We developed a process template builder, which assists template providers or mobile users in defining and editing process templates. The template builder offers a visual editor (an extension of Sheng et al.[11]) for describing statechart diagrams of templates. The client was implemented using J2ME Wireless Toolkit 2.[b] kXML 2[c] is used to parse XML documents on mobile de-

b http://java.sun.com/products/sjwtoolkit.
c http://kxml.enhydra.org.
d http://ksoap.objectweb.org.

Figure 3-b-1

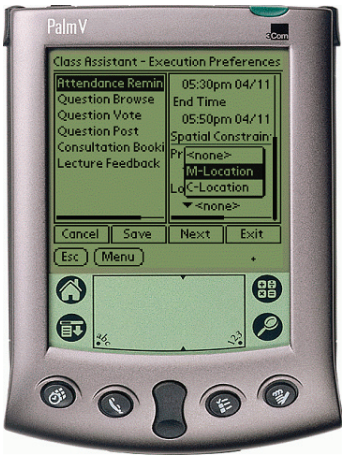


Figure 3-b-2

vices and kSOAP 2.0[d] is used by the client to handle SOAP messages. Figure 3 (b) shows the screenshots of process template configuration.

Currently, the functionalities of agents are realized by a set of pre-built Java classes. In particular, the class *deployAgent* (for the deployment agent) provides method called deploy() that is responsible for generating orchestration tuples from composite services. The input is a personalized composite service described as an XML document, while the outputs are orchestration tuples formatted as XML documents as well. The orchestration tuples are then uploaded into the tuple spaces of the corresponding service agents. IBM's TSpaces[e] is used for the implementation of tuple spaces.

To validate our design of the system architecture, we conducted a usability study. We presented our system to 18 people. The presentation includes a powerpoint show of the architecture overview, a demonstration of the usage of the system, and classAssistant, the prototype application built on top of the architecture. The participants were then asked to use the system and given a questionnaire to report their experience in using the system.

Table 1 shows some of the questions from the questionnaire and the participants' responses. The responses actually validate and highlight some important design principles of the architecture: users should avoid data entry as much as possible, the interactions during service execution should be asynchronous, and the bandwidth consumption should be minimized. More experimental results (for example, performance study) are not reported here due to space reasons. Readers are referred to Sheng[10] for a detailed description of the system evaluation.

e http://www.alphaworks.ibm.com/tech/tspaces.

Related Work

Very little work is being done on Web services orchestration for the benefit of mobile users. A middleware named Colomba[1] handles the dynamic binding for mobile applications. This binding addresses multiple issues such as frequent disconnection, limited supply of battery power and absence of network coverage. A personal process-oriented system for mobile users is presented in Huang[7], which investigates the specification and querying of processes that involve personal tasks and data. The objective of these approaches is to support simple applications for mobile users, rather than providing personalized specifications and adaptive orchestrations of composite services, fulfilling *complex* user activities.

PerCollab[3] is a middleware system that integrates multiple communication devices within workflow systems. Relying on a centralized BPEL engine and a context service, tasks can be proactively pushed to users. However, PerCollab does not consider the personalized specification of business processes, nor distributed orchestration of the processes.

The emerging semantic Web efforts such as OWL-S[f] and WSMF (Web Service Modeling Framework),[5] promote the use of ontologies as a means for reconciling semantic heterogeneity between Web resources. Such efforts focus on designing rich and machine understandable representations of service properties, capabilities, and behavior, as well as reasoning mechanisms to select and aggregate services. The issues addressed in this area are complementary to those addressed in our work.

Finally, we also note that some industrial efforts in mobile Web services such as IBM's Web Service Toolkit for Mobile Devices[g] and Microsoft and Vodafone's Mobile Web Service initiative.[h] The former provides tools for developing Web services on mobile devices while the latter plans to create new standards to integrate IT and mobile worlds through Web services.

Conclusion

Our system builds upon the building blocks of Web services, agents, and publish/subscribe systems and provides a platform through which services can be offered to mobile users. The work is one step toward simplifying the design and implementation of process-based mobile applications like personal daily activities management and work support in critical workplaces like hospitals. Ongoing work

Table 1.

Questions	Responses		
	A	B	C
Suppose you are invoking a remote service using a PDA and the invocation will take some time, which action you prefer to take: (A) wait with the handheld on till receive the result; (B) turn off the handheld and catch the results in another time	6	12	N/A
Suppose you are invoking a service using a PDA and the service needs some inputs, which strategy is your favourite to supply values for the service inputs: (A) manually input using stylus; (B) automatically collect the data (e.g., from user profile) (C) does not matter	1	15	2
Suppose you are receiving the results of a service using a PDA, you would like to receive: (A) all of them; (B) only important and necessary ones (C) does not matter	6	11	1

f http://www.daml.org/services/owl-s.
g http://www.alphaworks.ibm.com/tech/wstkmd.
h http://www.microsoft.com/serviceproviders/resources/bizresmwsroadmap.mspx.

includes extending the architecture to support large-scale environments, and building more mobile applications on top of the architecture to further study its performance.

REFERENCES

1. Bellavista, P., Corradi, A., Montanari, R., and Stefanelli, C. Dynamic binding in mobile applications: A middleware approach. *IEEE Internet Computing* 7, 2 (Mar/Apr 2003), 34–42.
2. Caporuscio, M., Carzaniga, A., and Wolf, A.L. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Transactions on Software Engineering 29*, 12 (Dec. 2003), 1059–1071.
3. Chakraborty, D., and Lei, H. Extending the reach of business processes. *IEEE Computer 37*, 4 (Apr. 2004), 78–80.
4. Chen, Y.-F. R. and Petrie, C. Ubiquitous mobile computing. *IEEE Internet Computing 7*, 2 (2003), 16–17.
5. Fensel, D. and Bussler, C. The Web service modeling framework WSMF. *Electronic Commerce Research and Applications, 1*, 2 (2002), 113–137.
6. Harel, D. and Naamad, A. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology 5*, 4 (Oct. 1996) 293–333.
7. Hwang, S.-Y. and Chen, Y.-F. Personal workflows: Modeling and management. In *Proc. of the 4th International Conference on Mobile Data Management*, (Melbourne, Australia, Jan. 2003).
8. Keidl, M. and Kemper, A. Towards context-aware adaptable web services. In *Proceedings of the 13th International World Wide Web Conference*, (New York, May 2004).
9. Papazoglou, M. P. and van den Heuvel, W.-J. Service-oriented architectures: Approaches, technologies, and research issues. *The VLDB Journal 16*, 3 (July 2007), 389-415.
10. Sheng, Q. Z. *Composite Web Services Provisioning in Dynamic Environments*. Ph.D. thesis, The University of New South Wales, Sydney, Australia, 2006.
11. Sheng, Q. Z., Benatallah, B., Dumas, M., and Mak, E. SELF-SERV: A platform for rapid composition of Web services in a peer-to-peer environment. In *Proc. of the 28th International Conference on Very Large Databases*, (Hong Kong, China, Aug. 2002).
12. Zambonelli, F., Jennings, N.R., and Wooldridge, M. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology, 12*, 3, 317–370, (July 2003).

Quan Z. Sheng (qsheng@cs.adelaide.edu.au) is a lecturer in the School of Computer Science at the University of Adelaide, Adelaide, Australia.

Boualem Benatallah (boualem@cse.unsw.edu.au) is a professor in the School of Computer Science and Engineering at the University of New South Wales, Sydney, Australia.

Zakaria Maamar (zakaria.maamar@zu.ac.ae) is an associate professor in the College of Information Technology at Zayed University, Dubai, U.A.E.