Applying Coordination for Service Adaptation in Mobile Computing

Mobile computing devices vary in terms of display, memory, and battery life, and current network protocols aren't necessarily suited to the mobile computing environment. As computer networks' complexity increases, communication-related software must be able to adapt to mobile computing's constraints. This study aims to develop a software system that adapts data flows over dynamic wireless network conditions and various mobile devices. With their MobiGATE system, the authors adopt the principle of separation of concerns to support the system's service composition and reconfiguration. This approach can support ease of dynamic reconfiguration and the reusability of adaptation services across applications.

Yongjie Zheng
University of Florida

Alvin T.S. Chan and Grace Ngai
The Hong Kong Polytechnic University

Computer networks are increasingly complex and variable, with mobility dramatically exacerbating the issue. This complexity carries over to mobile computing devices, which often vary in terms of display characteristics, memory size, and battery lifetimes. Moreover, the existing network protocols that have enabled the Internet revolution aren't well suited to the mobile computing environment. TCP, for example, doesn't work well on many wireless links and often behaves poorly over satellite links due to long latencies.[1]

For mobile applications to operate effectively and optimally, communication-related software at the application level must be able to adapt to mobile constraints at runtime. One general solution is to allow adaptation of network traffic, which lets augmented network services perform aggressive computation and storage to alter protocols or the data content being transmitted, and to improve quality of service for users. Recently, researchers have built, tested, and in some cases validated systems embodying this idea,[2–5] demonstrating software-supported adaptation's benefits. Existing systems often implement service composition as static interactions of service entities by explicitly invoking procedures on named interfaces. As a result, the system-integration code becomes entangled with the application-specific code. Replacing or modifying a service entity requires updating to integrate not only the code for the new service entity but also the code of entities

directly related to the old one. This tight coupling of service entities in terms of a strong communication dependency translates into the need for manual modifications when we deploy transport service entities into new environments. In wireless networks, which exhibit highly dynamic network conditions, adapting service entities in the form of dynamic composition and reconfiguration should be the norm rather than the exception.

At the Hong Kong Polytechnic University, we developed the Mobile Gateway for the Active Deployment of Transport Entities (MobiGATE) architecture, a service-based system that adapts data flows over dynamic wireless network conditions and various mobile devices. A major goal of MobiGATE is to provide an environment in which programmers can develop new mobile applications by combining some adaptation services, while keeping the application's configuration structure completely separate from individual services' computational activities. Such functionality is particularly important for facilitating the dynamic reconfiguration and reuse of adaptation services. To achieve this functionality, we've adopted the *coordination concept*[6] for MobiGATE, which is composed of a series of *streamlets* — functional services that different applications can reuse or dynamically reconfigure at runtime without recompilation or redefinition. After developing the MobiGATE architecture, we conducted a case study to test its adaptation performance.

Background

We based our research study on concepts from the fields of coordination theory and service adaptation. Given that our study's main objective was to develop an adaptive software system, we focused more on service adaptation.

Coordination: Separation of Concerns

Coordination theory[6] is an emerging research area that focuses on the interdisciplinary study of coordination. Research in this area uses and extends ideas about coordination from disciplines such as computer science, organization theory, operations research, economics, linguistics, and psychology. Coordination theory defines coordination as the process of managing dependencies among activities. Its research agenda includes characterizing different kinds of dependencies and identifying the coordination processes we can use to manage them.

The greatest advantage of applying coordination theory to service adaptation is that it completely separates coordination from computational concerns — known as a *separation of concerns* — usually by defining a new coordination language to describe the composition's architecture. In particular, the coordination system generally consists of two kinds of processes: *computational* and *coordination*. Computational processes are treated as black boxes whose internal service logic is invisible to the external environment; the processes communicate with their environment via clearly defined interfaces, usually referred to as input or output ports. Coordination processes form producer–consumer relationships by setting up channel connections between the producer output ports and the consumer input ports.

Our research study is directly related to coordination theory, in that it views the process of developing mobile applications in MobiGATE as one of specifying architectures in which patterns of dependencies among adaptation services (streamlets) are eventually managed by coordination processes composed in the MobiGATE Coordination Language (MCL), which we defined to describe the composition of streamlets running in the MobiGATE system. (A detailed discussion of MCL is beyond this article's scope, but more information appears elsewhere.[7]) Streamlets in MobiGATE represent applications' main functional elements: they work as coordination units but don't require explicit knowledge about the coordination patterns in the application. Separating and externalizing the streamlets' interconnections greatly promotes their independence and reusability.

Service Adaptation

Because characteristics of mobile computing environments vary dramatically, doing service adaptation at the application level is necessary to optimize overlying mobile applications. We can adapt data flows over networks in many ways to shield clients from the effects of poor networks. The service entities involved include transformation (such as filtering or format conversion), aggregation (collecting and collating data from various sources), caching (both original and transformed content), and customization (maintaining a per-user preferences database).

Adaptive solutions to network problems have many interesting variations, including the University of California, Berkeley's TranSend,[3] the Odyssey system,[4] and the RAPIDware project.[5] Although these systems differ significantly in some respects, all offer ways to change the contents of

Table 1. A comparison of adaptive systems.

	TranSend	Odyssey	RAPIDware	MobiGATE
Adaptation location	Proxy	Client and server	Proxy and client	Proxy and client
Application awareness	Application-transparent	Application-aware	Application-transparent	Application-transparent
Dynamic reconfiguration	Yes	No	Yes	Yes
Adaptation compositions	Yes	No	Partial	Yes
Mechanism	Data-type-specific distillation	Resource management	Detachable stream objects	Separation of concerns
Description	Web acceleration through data-type-specific lossy compression	Application-aware adaptation by multiple applications using diverse data types	Web-based collaboration in in heterogeneous wireless environments	Applying coordination theory in service composition and system reconfigurations

transmitted data or the methods used to send that data. All adapt to changing conditions specific to the data transmission requested, prevailing network conditions, or users' needs. However, they differ from the MobiGATE system in some important aspects, ranging from the mechanism of service compositions to the resultant capability of dynamic reconfigurations and adaptation modes.

Table 1 compares the MobiGATE system with the three other systems. (For a more detailed discussion of the differences, see the "Related Work in Service Adaptation" sidebar.) Differences can occur in various areas:

- *Adaptation location* describes where the adaptation machinery resides — in the client, in the server, in one or more intermediate proxies, or in all of these.
- *Application awareness* in these systems can be application-aware or application-transparent, depending on whether the application knows that an adaptation is occurring and is perhaps expected to provide a response, or whether the system attempts to completely shield the application from this fact.
- *Dynamic reconfiguration* describes the system's ability to dynamically change its composition structure and create or destroy service instances at runtime.
- *Adaptation composition* refers to the possibility of composing adaptations in the adaptation machinery — in other words, it points out whether the adaptation can occur at multiple levels.
- The *mechanism* is the primary technology the adaptation uses. As far as the MobiGATE system is concerned, the notion of separation of concerns forms the underlying principle adopted in the adaptive system's design.
- The *description* summarizes the adaptive system.

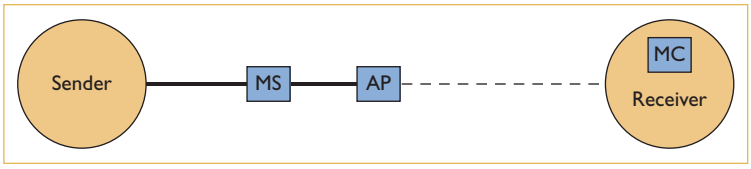

Figure 1. The MobiGATE system's working paradigm. The thick line represents the wired network and the dashed line suggests the wireless network. The access point (AP) at the wired network's edge supports communication between the fixed sender and its mobile receiver. At some point in the network, the MobiGATE server (MS) imposes various adaptation services on the data flow, which the MobiGATE client (MC) then reverse-processes at the receiver side.

This comparison with existing systems gives us a general idea about how MobiGATE works. To characterize the system to a much deeper extent, however, we must look at its architecture and evaluate its performance.

The MobiGATE Architecture

As we stated earlier, our main objective with the MobiGATE system is to adapt data flows over dynamic wireless network conditions and various mobile devices at the application level. Here, we focus on MobiGATE's design.

Working Paradigms

The MobiGATE system consists of two parts: a server and a client. The MobiGATE server, in which the system composes dataflow adaptations, resides in the intermediate proxy between the data sender and data receiver. The MobiGATE client, in most cases, acts as the data receiver and is responsible for reverse-processing received messages.

Figure 1 shows a simple data flow with a single sender (S) and receiver (R) — the data flows across various links and network nodes. The thick line represents the wired network and the dashed line suggests the wireless part. The access point

Related Work in Service Adaptation

The Mobile Gateway for the Active Deployment of Transport Entities (MobiGATE) is not the first system to examine service adaptation. UC Berkeley's TranSend Web accelerator proxy was one of the earliest projects to explore adaptation proxies aggressively.[1] TranSend intercepts HTTP requests from standard Web clients and applies data-type-specific lossy compression when possible. Unlike MobiGATE, however, TranSend doesn't have a client-side system to reverse-process data flows from the server, such as decompression and decryption. Thus, TranSend can't support applications that need client-side processing for adaptation.

Odyssey was built at Carnegie Mellon University to support challenging network applications on portable computers.[2] One interesting aspect of Odyssey with regard to the adaptation framework is that the applications interacting with the system do much of the adaptation. Odyssey itself doesn't decide to convert color video frames to black-and-white, for example; rather, it instructs the application that some action is required. This aspect highlights a big difference between Odyssey and MobiGATE, which completely shields applications from the adaptation work via a specially designed event-propagation mechanism (which we discuss in more detail in the main text). Significantly, Odyssey's approach to adaptation is to adjust the quality of individual services to match available resources. It doesn't support dynamic reconfigurations in the form of inserting or removing adaptation services at runtime.

Finally, the Michigan State University RAPIDware project addresses the design and implementation of middleware services for dynamic, heterogeneous environments.[3] RAPIDware is similar to MobiGATE in that both use the concept of filters and streams, have a lightweight client system, and can dynamically reconfigure service composition structures. However, RAPIDware can't fully support branch and multilevel compositions of adaptation services. Based on the MobiGATE Coordination Language descriptions, MobiGATE can enforce an analysis function to verify composition activities' correctness, which isn't possible in any of the other three projects.

References

1. A. Fox et al., "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *Proc. 7th Internet Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS-VII), 1996, pp. 160–170.
2. B. Noble et al., "Agile Application-Aware Adaptation for Mobility," *Proc. 16th ACM Symp. Operating System Principles*, ACM Press, 1997, pp. 276–287.
3. P.K. McKinley et al., "Composable Proxy Services to Support Collaboration on the Mobile Internet," *IEEE Trans. Computers*, vol. 52, no. 6, 2003, pp. 713–726.

(AP) is located at the wired network's edge and supports communication between the fixed sender and its mobile receiver. At some point in the network, the MobiGATE server (MS in the figure) imposes various adaptation services on the data flow, which the MobiGATE client (MC in the figure) then reverse-processes at the receiver side.

Figure 1 is, of course, a simplification of the real world — it shows a simple data flow and doesn't illustrate problems, such as delivery deadlines or security concerns. It also fails to suggest the level of complexity possible in even a single network flow. However, the figure does capture the root of the problem — a stream of data flows from a source to a destination across a network, using links of different conditions. Altering the data flows in various ways could lead to better overall results in terms of lowering bandwidth requirements, alleviating error conditions, encoding secured data, generic compression, and transcoding. However, MobiGATE doesn't aim to provide specific services or configurations of services, but rather a general platform to facilitate ease of deploying services across the wireless links by providing core mechanisms and system services.

The Server

There's a clear distinction in MobiGATE between coordination and computation. Figure 2 shows the MobiGATE server's architecture, which is organized into two executing planes: the *streamlet execution plane* schedules streamlet instances for computation, whereas the *stream coordination plane* maintains the interaction and relationship between the coordinated streamlets. The *coordination manager* maintains a configuration table for each instance of streamlet composition that contains meta-information on the initial composition of streamlets and reconfiguration actions in response to different events. All this information comes from the corresponding compiled MCL script that enforces those high-level policies. Based on the derived table, the *coordination manager* initializes and reconfigures related applications appropriately.

On another plane, the *streamlet manager* controls the execution of streamlet instances. During setup, the manager must locate classes of streamlets and allocate necessary computational resources for their execution. The *event manager* generates system events in reaction to different conditions; we discuss this component in more detail later. Finally, in the *streamlet directory*,

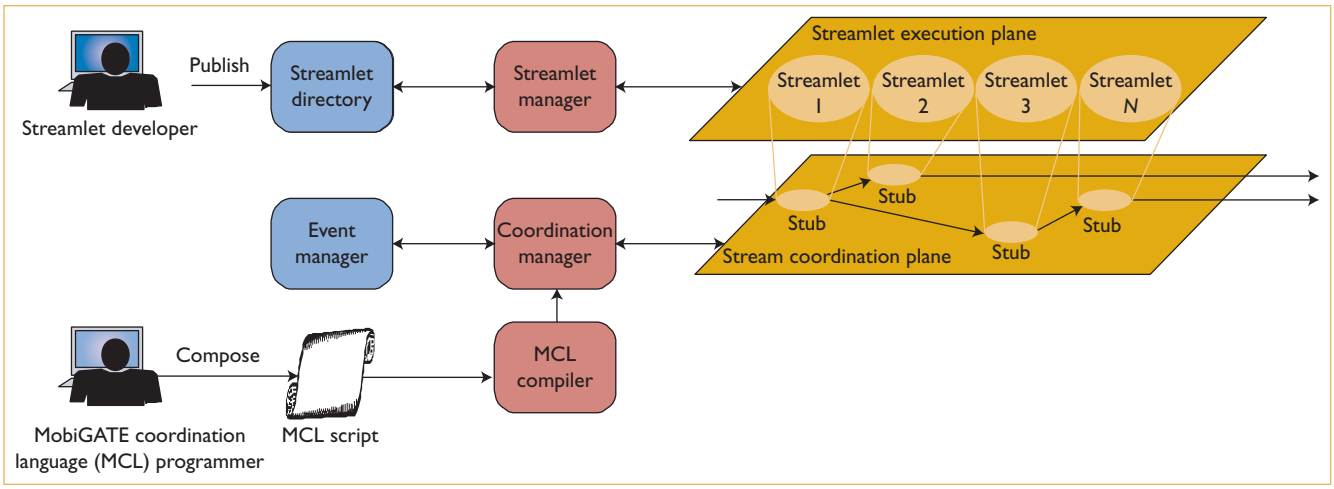

Figure 2. MobiGATE server architecture. The system consists of two execution planes: the streamlet execution and the stream coordination. The former schedules the computation of streamlets, whereas the latter is responsible for communications between different streamlets.

streamlet providers can advertise their services. This directory also provides code-level implementations of streamlets at runtime.

For the MobiGATE architecture, traditional application development is separated into two independent activities: streamlet development and MCL script composition; we can also regard these activities as two different levels of programming. For streamlet development, programmers don't know or care where their input comes from or where their output goes. They know nothing about the application's coordination pattern, and focus on imposing service logic on incoming messages. In contrast, in MCL composition, programmers know no details about the tasks the streamlets perform. The only concern is ensuring that streamlets receive the right input from the right source, deliver the results to the right sinks, and get reconfigured correctly in response to different system events. It's possible to use streamlets developed for one application in other concurrent applications, even with their different coordination patterns. This reusability results from the separation of concerns and is one of MobiGATE's most desirable properties.

The Client

In contrast to the server, the MobiGATE client system has no concept of channel or coordination. All the composition information is already recorded in the incoming message header. At the client side, the system simply needs to read the message headers and distribute the messages to corresponding client streamlets for reverse processing. It then sends the resultant messages to higher-layered applications. This asymmetry mechanism has greatly liberated MobiGATE client systems from heavy coordination logic and translates into much lower consumption of computing resources and energy on the client side.

The Event System

The generation and propagation of system events is another important issue we considered when designing MobiGATE. Today's Internet clients vary widely with respect to both hardware and software properties, including screen size, color depth, effective bandwidth, processing power, and the ability to handle different data formats. To build a dynamically adaptable system, we had to capture the various client variations and model them into a standard and recognizable form before the system could respond to them.

In the MobiGATE event system, we classified all client variations into four categories, each of which represents one axis along which clients can vary. *System command* is some system-generated event that manages running applications; *network variation* is related to underlying network conditions; and *hardware variation* and *software variation* correspond to different settings of mobile clients. In contrast to some existing adaptive systems, MobiGATE should completely shield overlying applications from system events and the resultant adaptation work. To achieve this, we designed a kernel entity, the event manager we mentioned earlier, to control the event system's operation, including event subscription, triggering, and monitoring. The event manager monitors the underlying client

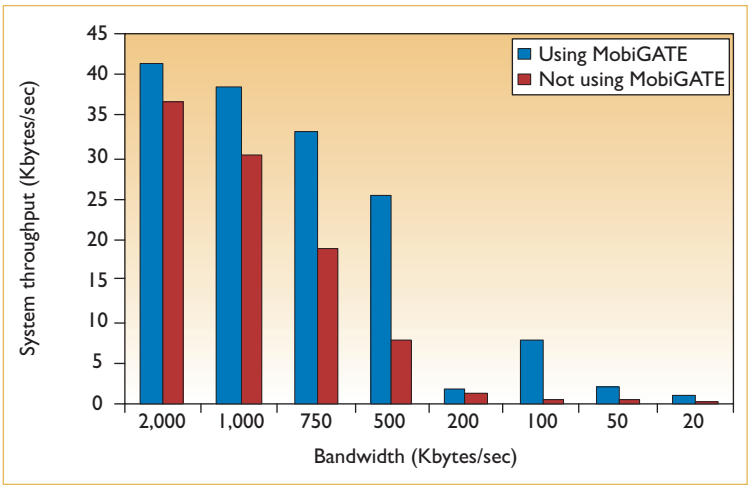

Figure 3. Experimental results. The MobiGATE system obtained improvement in system throughput over multiple bandwidths.

variations and composes corresponding events in response to various situations. Simultaneously, it multicasts events among different stream compositions, which will invoke the `onEvent()` method after they receive these events. To avoid incurring overheads from processing this flood of events, individual streams can subscribe to events of interest and react to only those by performing appropriate reconfiguration, while ignoring events that they consider superfluous. To support this function, the event manager is equipped with the `subscribeEvent()` method, which stream applications can use to register their events of interest.

Case Example

We wanted to fully exercise the MobiGATE system's components by setting up a realistic test bed in the form of a stream application operating over an emulated wireless network. To conduct this case study, we implemented the entire MobiGATE system and its runtime environment on a Java platform to promote maximum interoperability across heterogeneous systems. The testing setup included three PCs: one acted as the MobiGATE server residing on the wired departmental LAN; a second acted as the mobile node; and the third, installed with the NIST Net network emulator, we configured to act as a wireless router for emulating a wireless operating environment. Having a single server and a single client simplified the experimental scenario, while still retaining the experiments' end-to-end semantics. Importantly, the setup let us completely emulate a real system operation that exercises the interactions across all entities, including the proxies and client. We also prepared a pragmatic example of the composition of service entities. For this experiment, we used the following service entities, in the form of streamlets:

- `switch` divides incoming messages based on the data's semantic type;
- `image lower size` reduces the size of incoming images;
- `image down sampling` reduces the sample rate to create lossy compression of an image;
- `text compress` a generic text compressor; and
- `merge` integrates different types of information into a whole body.

In the application, a component continuously generated several real image and text messages. The system processed image messages via the `switch`, `image lower size`, `image down sampling`, and `merge` streamlets successively, from start to finish. Text messages encountered a different situation, however. Under normal conditions (bandwidth greater than 100 Kbytes per second), the text messages passed only through the streamlets `switch` and `merge`. However, when the bandwidth fell below 100 Kbytes per second, the system inserted the `text compress` streamlet between the other two to adapt to the poor bandwidth. After recording each message's sending and receiving time, we calculated the time incurred to transmit each message and obtained the overall system throughput.

We thoroughly measured system throughput for the experiment over bandwidths of 20, 50, 100, 200, 500, and 750 Kbytes, and 1 and 2 Mbytes per second, successively. Figure 3 shows the results. System throughput improved noticeably with the MobiGATE system as compared with a setup that directly transferred messages across the wireless link. The throughput gain increased as bandwidth decreased. We expected this because the effect of applying streamlet services to reduce the amount of required bandwidth begins to take prominence. When the bandwidth fell below 100 Kbytes per second, the system invoked a special reconfiguration mechanism to insert the `text compress` streamlet into the stream. The results indicate that the system throughput improved greatly (from 1 Kbyte to 4 to 7 Kbytes per second). The experiments clearly indicate MobiGATE's benefit and its ability to offset processing overheads that deploying the streamlet application might incur. This is particularly true if we deploy MobiGATE in an environment in which resource availability is dynamic and scarce.

Figure 4 shows MobiGATE's adaptation effects for transforming image messages in the experiment. Figure 4a presents the original image; Figure 4b shows the results of this image passing through the `image down sampling` streamlet; and Figure 4c shows the effects of the `image lower size` streamlet. Finally, Figure 4d shows the resulting image that made it to the client. By passing images through these streamlets, MobiGATE can greatly reduce transmission volume, while concurrently supporting mobile clients with smaller displays.

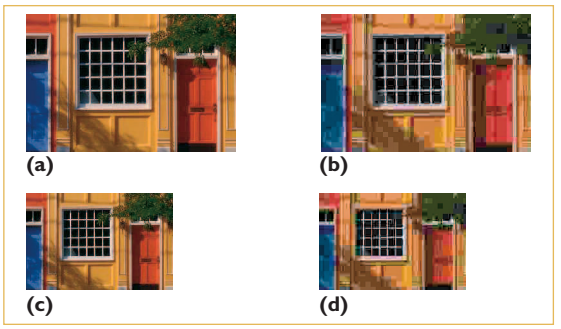

Figure 4. MobiGATE's adaptation effects on image data. (a) The original image (12.9 Kbytes) passes through (b) the `image down sampling` *streamlet (3.18 Kbytes) and c) the* `image lower size` *streamlet (8.63 Kbytes), resulting in (d) a smaller, lower-quality image (2 Kbytes), which the client receives on a smaller display.*

Our approach to service adaptation has several desirable properties, including reusability, ease of modification, and maintenance of an intuitive processing flow. Our empirical experimental results demonstrated the system's effectiveness in adapting data flows over an emulated wireless link while incurring insignificant computational overheads in its execution environment.

As an adaptive system running in the mobile world, MobiGATE needs to consider many system and deployment issues — far more than the separation of concerns we discussed in this article. Scalability, security, and heterogeneity are three such important topics necessitating future exploration before we can realistically deploy MobiGATE in an open and wide-area environment.

Acknowledgments

Hong Kong Polytechnic University Central Research Grant G-U154 and Internal Competitive Research Grant AP-F82 supported this work.

References

1. R. Caceres and L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," *IEEE J. Selected Areas in Comm.*, vol. 13, no. 5, 1995, pp. 850–857.
2. A.T.S. Chan and S.N. Chuang, "MobiPADS: A Reflective Middleware for Context-Aware Computing," *IEEE Trans. Software Eng.*, vol. 29, no. 12, 2003, pp. 1072–1085.
3. A. Fox et al., "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *Proc. 7th Internet Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS-VII), 1996, pp. 160–170.
4. B. Noble et al., "Agile Application-Aware Adaptation for Mobility," *Proc. 16th ACM Symp. Operating System Principles*, ACM Press, 1997, pp. 276–287.
5. P.K. McKinley et al., "Composable Proxy Services to Support Collaboration on the Mobile Internet," *IEEE Trans. Computers*, vol. 52, no. 6, 2003, pp. 713–726.
6. T.W. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," *ACM Computing Surveys*, vol. 26, 1994, pp. 87–119.
7. Y. Zheng and A.T.S. Chan, " Stream Composition for Highly Adaptive and Reconfigurable Mobile Middleware," *Proc. 28th IEEE Ann. Int'l Computer Software and Applications Conf.* (COMPSAC 04), IEEE Press, 2004, pp. 122–127.

Yongjie Zheng is a PhD student in the University of Florida's Department of Computer and Information Science and Engineering. His current research includes adaptive middleware, software architecture, and mobile computing. Zheng has a BE in computer science and technology from Tsinghua University and an MPhil in computer science from the Hong Kong Polytechnic University. Contact him at yzheng@cise.ufl.edu.

Alvin T.S. Chan is an associate professor at the Hong Kong Polytechnic University and founding member of a university spin-off company, Information Access Technology. He is an active consultant and has been providing consultancy services to both local and overseas companies. His research interests include computer networking, mobile computing, and context-aware computing. Chan has a PhD in computer engineering from the University of New South Wales, Australia. He is a member of the IEEE and the ACM. Contact him at cstschan@comp.polyu.edu.hk.

Grace Ngai is an assistant professor at the Hong Kong Polytechnic University. Her research interests include natural language processing, machine learning, and mobile compiling. Ngai has a PhD in computer science from the Johns Hopkins University. She is a member of the ACM. Contact her at csgngai@comp.polyu.edu.hk.