ContextPhone : a prototyping platform for context-aware mobile applications

Raento, Mika

2005

Raento , M , Oulasvirta , A , Petit , R & Toivonen , H 2005 , ' ContextPhone : a prototyping platform for context-aware mobile applications ' , IEEE Pervasive Computing , vol. 4 , no. 2 , pp. 51-59 .

http://hdl.handle.net/10138/143998

publishedVersion

ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications

ContextPhone was developed using an iterative, human-centered design strategy. It thus helps developers more easily create applications that integrate into both existing technologies and users' everyday lives.

Mika Raento, Antti Oulasvirta, Renaud Petit, and Hannu Toivonen
University of Helsinki and Helsinki Institute for Information Technology

As a computing platform, mobile phones are both pervasive and personal. They're almost always on and tend to have an intimate relationship with their owners, who store private information on them and often personalize their appearance or ring tones, for example. This personal nature suggests that mobile phones are well suited for context-aware computing. On one hand, mobile phones follow the user and have clues about the current situation. On the other, their various usage contexts will likely benefit from context awareness.

Smart phones are a particularly tempting platform for building context-aware applications because they're programmable and often use well-known operating systems. There's a gap, however, between the operating systems' functionality and the features that application developers need. A smart phone knows, for example, how to connect to a Bluetooth device—such as a Bluetooth-enabled Global Positioning System receiver—but applications need the actual GPS coordinates.

To fill this gap, we've designed and developed *ContextPhone*, a software platform consisting of four interconnected modules provided as a set of open source C++ libraries and source code components. ContextPhone runs on off-the-shelf mobile phones using Symbian OS (www.symbian.com) and the Nokia Series 60 Smartphone platform (www.series60.com). To develop ContextPhone, we followed a human-centered research strategy[1] that included field studies of application use. As a result, our platform offers several useful capabilities and functions that existing platforms don't.

Design goals and philosophy

The current ContextPhone version is the result of several development iterations and repeated evaluations of real-world use. Our development experiences led to several design goals.

The first is to *provide context as a resource*. The platform represents the user's machine-sensed context in a way that humans can understand and communicates it to the user's environment when appropriate. The context thus becomes a resource for social interaction, not just input to machine adaptation. Humans can thus construct new meanings from the contextual information.[2]

The second is to *incorporate existing applications*. Most previous work, including our own, has implemented custom-built hardware systems or isolated applications. To accommodate the deeply intertwined nature of human mobility practices, ContextPhone interfaces and integrates with existing Smartphone applications, particularly messaging and calling functions.

The third is to *offer fast interaction and unobtrusiveness*. In mobility, cognitive-interaction resources are seriously fragmented and interrup-

1536-1268/05/$20.00 © 2005 IEEE ■ *Published by the IEEE CS and IEEE ComSoc*

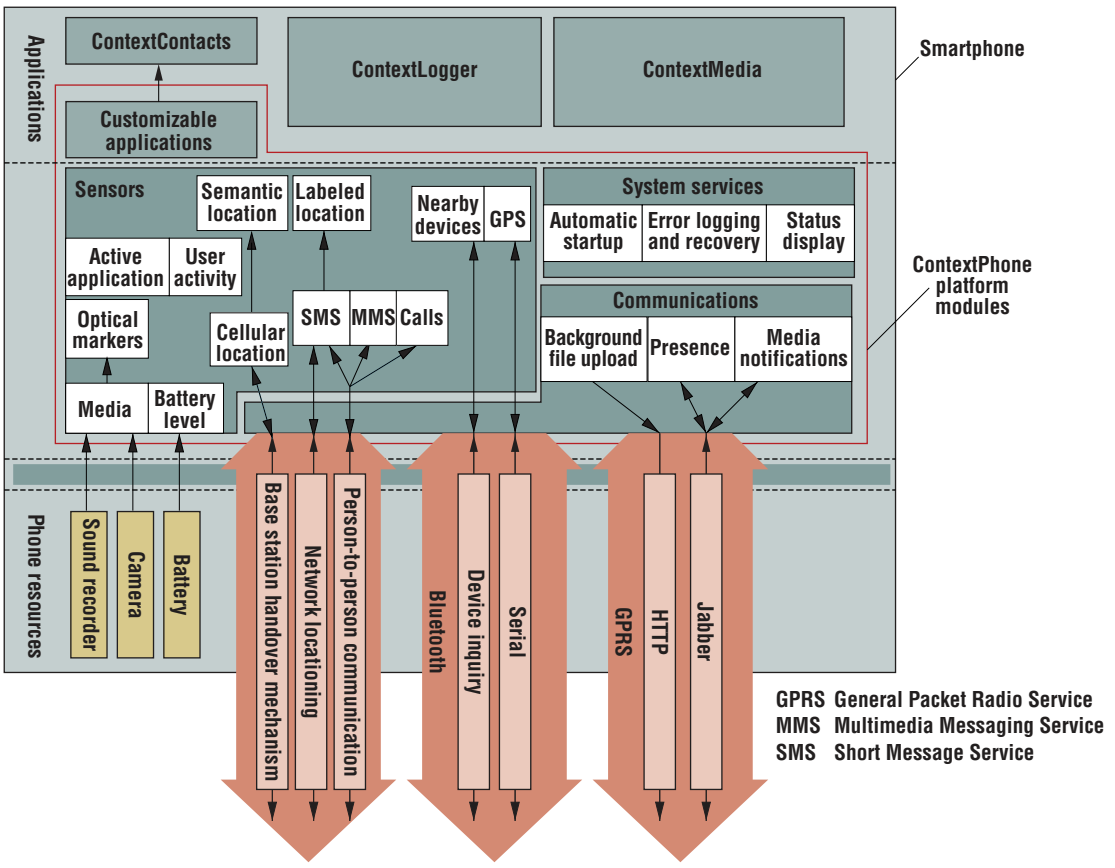

Figure 1. The ContextPhone platform. Four interconnected modules—sensors, system services, communications, and customizable applications—facilitate communication with the outside world.

tions are ubiquitous.[3] The platform must therefore enable fast interaction when needed and otherwise run in the background without intruding on other applications' use.

The fourth is to *ensure robustness*. The platform should automatically recover from lost power or connectivity and from internal logic and system component failures without user correction. All significant data should be automatically saved to nonvolatile memory. These features build trust and prevent negative social consequences.

The fifth is to *let users control seams*. Battery life, technical-platform heterogeneity, and network connectivity gaps create seams in interaction.[4] Empowering users to compensate for and control these seams is desirable in many applications.

The sixth is to *emphasize timeliness*. Response latency is crucial for applications that support turn-taking-based communications. Social events are often significant only at a particular moment. Several applications and terminals might require quick access to socially significant context information.

The final goal is to *enable rapid development*. Developers should be able to easily add new context data sources, sensors, and sinks, and build new applications without rebuilding the entire system. At best, data types and components should be runtime extensible; at least, developers should have a unified structure for adding new types and components at compile time. These features support rapid, iterative development that involves users in every stage of design.

These goals are generally applicable in helping developers create applications that are useful to people in everyday life, as opposed to building stand-alone tools for limited settings.[5]

The ContextPhone platform

Figure 1 shows ContextPhone's four modules:

- *Sensors* acquire context data from different sources, such as location (Cell ID and GPS) or phone use.
- *Communications* connect to external services via standard Internet protocols using General Packet Radio Ser-

vice (GPRS), Bluetooth transfers, Short Message Service (SMS), and Multimedia Messaging Service (MMS). The communication channels can, for example, share presence information (using Jabber, as we describe later) or obtain sensor data (using GPS over Bluetooth).
- *Customizable applications*—such as ContextLogger, ContextContacts, and ContextMedia—can seamlessly augment or replace built-in applications such as the Contacts and Recent Calls lists.
- *System services* automatically launches background services, error logging and recovery, and the Status display.

Sensors

ContextPhone can be used to sense, process, store, and transfer context data. The sensed or inferred context can then trigger actions within the phone or be communicated to the outside world. Although current Smartphones contain few physical sensors, their ability to monitor phone internal state and usage, observe the cellular network, and use wireless connections enables a rich contextual environment.

Our software supports four sensor types:

- *location*, including Global System for Mobile Communications (GSM) cell identifier, cell-based semantic location,[6] naming of cells via network location services, and GPS via a Bluetooth GPS receiver;
- *user interaction*, including active application, idle/active status, phone alarm profile, charger status, and media capture;
- *communication behavior*, including calls and call attempts, call recording, sent and received SMS, and SMS content; and
- *physical environment*, including surrounding Bluetooth devices, Bluetooth networking availability, and optical marker recognition (using the built-in camera).

Some of these sensors seem almost trivial, but we've found uses for them in specific applications. For example, whether the phone is idle or active indicates the user's availability in our ContextContacts application. Also, while using off-the-shelf hardware limits the number of physical sensors, developers can add support for external hardware for specific applications.

Because ContextPhone can send and receive SMS and MMS, developers can incorporate any mobile service that can be used through text messages.

Communications

A phone's most important use isn't to run applications but to communicate with the outside world. ContextPhone supports both local (infrared and Bluetooth) and wide-area (GSM and GPRS) communications. Our communications package also offers protocol implementations and service abstractions on top of these.

To gather data (ContextLogger) and share media (ContextMedia), we need a way to upload files. Typically, to support our unobtrusiveness goal, ContextPhone automatically uploads files in the background through an `HTTP POST` request. However, a manually triggered local transfer is also possible. Because ContextPhone can send and receive SMS and MMS, developers can incorporate any mobile service that can be used through text messages, such as network location services. To distribute presence information and notifications, we use the Jabber extensible messaging and presence protocol (see www.jabber.org).

Customizable applications

Series 60's built-in applications have at best limited customizability. Developers can use our customizable versions of its built-in call-making applications (Contacts and Recent Calls) to add new features to person-to-person communication. Our applications work exactly the same way as the more familiar built-in ones, but developers can extend them. They might, for example, use our ContextContacts application to add presence information.

The customizable applications also support extensive user-interaction logging. This lets developers and researchers study usage patterns (including usability) in users' everyday lives without requiring human observers. Figure 2 shows a sample from ContextLogger.

System services

Unobtrusive applications require minimal user interaction and run without disturbing ongoing activities. Although background services should start and run automatically, Series 60 doesn't offer automatic startup. ContextPhone services add this feature. If a crash occurs, a watchdog process automatically restarts services and applications.

Robustness is a crucial feature of pervasive Smartphone applications. All ContextPhone components support disconnected execution, queuing operations—such as queuing file uploads when network connectivity is interrupted—and storing the latest network information locally. Most components have a retry and recovery strategy for

Scenario: Sending a short message to a contact to request a call back.

10h51m48s	Open Contact application
10h51m51s	Select the contact (Mika)
10h51m52s	Open the message composer and write the message
10h52m48s	Send the message
10h53m00s	Reception of delivery report
11h30m27s	Incoming call (Mika)
11h30m33s	Answering the call (recording starts)
11h33m59s	End of call

```
// Interaction log
20040623T105148   To foreground
20040623T105148   Showing contacts
20040623T105151   Items: [Antti 0 0 0]//Mika, loc: Exactum (1:00) 22 11 17//
                  Renaud 0 0 0
20040623T105151   Items: Antti 0 0 0//[Mika, loc: Exactum (1:00) 22 11 17]//
                  Renaud 0 0 0
20040623T105152   Sending SMS to: Mika, loc: Exactum (1:00) 22 11 17
20040623T105152   To background

// Context log
20040623T103507   profile:0 General (0 7 Off)
20040623T103629   area, cell, nw: 19000, 1952, RADIOLINJA
20040623T104620   devices: 0060579a6f70 [Janne] 0002eea07729 [Antti]
20040623T105148   UserActivity: active
20040623T105148   ActiveApp: [101fbad0] contextbook
20040623T105152   ActiveApp: [100058c5] mce
20040623T105248   SMS : sent msg #1053236 to Mika:"Please call me asap!"
20040623T105352   ActiveApp: [100056cf] ScreenSaver
20040623T105552   UserActivity: idle
20040623T113027   app event: STATUS: call
20040623T113027   app event: STATUS: call status 3
20040623T113027   ActiveApp: [100058b3] Phone
20040623T113033   UserActivity: active
20040623T113033   app event: STATUS: call status 4
20040623T113033   app event: STATUS: recording call
20040623T113359   app event: STATUS: recorded

// Communication log
20040623T105248   EVENT ID: 2268 CONTACT: -1 DESCRIPTION: Short message
                  DIRECTION: Outgoing DURATION: 0 NUMBER: +123456789
                  STATUS: Sent REMOTE: Mika
20040623T105300   EVENT ID: 2269 CONTACT: -1 DESCRIPTION: Short message
                  DIRECTION: Incoming DURATION: 0 NUMBER: +123456789
                  STATUS: Delivered REMOTE: Mika
20040623T113033   EVENT ID: 2270 CONTACT: -1 DESCRIPTION: Voice call
                  DIRECTION: Incoming DURATION: 207 NUMBER: +123456789
                  STATUS: REMOTE: Mika
```

Figure 2. An example ContextLogger log. The application logs details of the communication's nature, timing, and participants, along with an approximation of the current location (the current cell) and hints about the people present (the nearby Bluetooth devices). In addition to this data, ContextLogger can record the call.

other transient errors, such as memory exhaustion.

The system is useful in various circumstances. However, because it's continuously updated and runs on several OS versions, unexpected errors do occur. In such cases, components can propagate errors up to a level where they can be logged. Because some Symbian errors (called *panics*) are not trappable, we've added a persistent stack trace mechanism that can inspect a crashed program's state from the outside. In practice, we can therefore pinpoint the source and preceding call sequence of all errors. ContextPhone sends the logged errors to a server through the background file upload.

Architecture

Our architectural decisions were guided by our design goals, especially the need for a robust and extensible system that supports unobtrusiveness and rapid development. As the "Related Work: Platforms for Context-Aware Systems" sidebar describes, to rapidly produce actual applications we've built on the experiences of other context-aware platforms. We've also followed the Extreme Programming maxim: "Build the simplest thing that could possibly work." Our intention is that both the software's functionality and the architecture evolve over time.

Our first goal was for the software to log contextual information; the architecture thus focuses on context-event provision and storage. The components use a publish-subscribe model within a single process. This logical architecture mirrors the Context Toolkit's context widgets,[7] although only within a single system. Developers can easily extend the event sources and event sinks to provide simple context-triggered events.

Context-aware systems must also deal with *data type* extensibility. Our goal was to be able to easily add new contextual variables (both sensor based and inferred) to the system. ContextPhone has many built-in data types—such as GSM Cell ID, phone profile, and Bluetooth device—with defined serialization formats that the system can use to store and transfer the data. Developers can add new data types fairly easily, but only at compile time.

Applications and research tools

Research groups have used the ContextPhone software for many different applications. According to many of

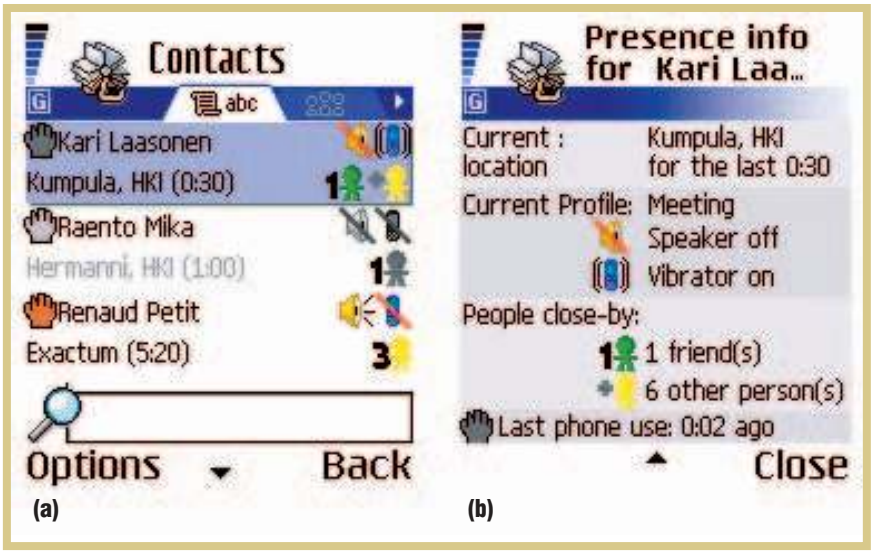

Figure 3. The ContextContacts context representation. (a) ContextContacts replaces the Smartphone Contacts list with one that includes information on current and past location, phone use activity (indicated by hand color), people present, and phone alarm profile. (b) By clicking on the contact in the list, the user sees more details and an explanation of the system's shorthand and icons.

these researchers, using our software has saved them months of work and, in some cases, enabled research that would have otherwise been infeasible owing to budget constraints. For example, the MIT Media Lab's Reality Mining group (http://reality.media.mit.edu) was able to give users a version of ContextLogger after only a few days of modification and testing.

Although adding sensors or changing the user interface requires some knowledge of Symbian programming, developers have built many applications simply by getting context information and sending feedback through ContextPhone's communication channels. Because these channels use standard Internet protocols, applications can be implemented quickly using off-the-shelf components such as Jabber clients or Web server scripting. Following are the three most important ContextPhone-based applications to date, which are all included with the basic ContextPhone package.

ContextLogger: Studying mobility patterns

ContextLogger records mobility data. Our goal with it is to give researchers a robust, reliable tool that requires minimum control and maintenance and lets them acquire rich data unobtrusively.

ContextLogger receives notifications of context changes from the sensors and customizable applications, writes this data in a local file, and periodically uploads the files to the researchers' server via the background file upload. Because ContextLogger requires no user interaction and isn't typically visible to users, it's unobtrusive. Figure 2 shows a sample of logged data.

We used the first version of ContextLogger to gather the data needed to build our adaptive location model for cellular data.[6] During this study, we realized the importance of unobtrusiveness: we started with manual uploads of the logs every few days, but this was unacceptably laborious. Also, we had to manually restart the software periodically. To remove these obstacles we added automatic data upload and automatic logger restart.

Unobtrusiveness isn't only important to users—it also lets researchers decrease the observation effect on their subjects. We're using ContextLogger to discover correlations between context data and a user's availability. Also, the Technical Research Centre of Finland (VTT) is using ContextLogger to collect data on Bluetooth device proximity for modeling the recurring patterns of a work group's person-to-person interactions. The project's goal is to learn and cluster group-collaboration patterns based on when people physically meet. In addition, the MIT Media Lab's Reality Mining project is collecting communication and proximity data from students to model social-network dynamics. The ability to measure social relationships' evolution and strength can "revolutionize the field of social network analysis," according to David Lazer,[8] a social scientist at Harvard University.

ContextContacts: Automatic context sharing

ContextContacts lets users automatically represent and exchange context information. Rather than build agents that decide whether the callee is interruptible (and block calls accordingly), our working hypothesis is that offering the callee cues about the caller's context might be more constructive. These cues should facilitate decisions about whether to call, and if so, which communication channel to use, thus supporting social awareness in the group. Figure 3 shows our context representation.

Given mobile users' fragmented attention,[3] the time it takes to make a phone call must remain extremely short. This rules out using a separate application to display the context. By using the customizable applications, ContextContacts integrates presence with normal calling and messaging. It provides the same response time as the built-in Contacts application by storing the callee's current context locally, using the Jabber channel's push-based presence notification.

ContextContacts has three components:

- the *presence publisher*, which gathers relevant sensory data from the sensors

Related Work: Platforms for Context-Aware Systems

Over the past few years, researchers have built many platforms for pervasive and context-aware systems that support rich contextual features. Several of these systems are openly available. However, no such systems are available for devices that are inexpensive, available off-the-shelf, and widely accepted by users in their everyday life—such as smart phones, the first real-world pervasive platform.

Foundational work

Mark Weiser's vision for ubiquitous—or pervasive—computing was first truly embodied in the Xerox PARC's ParcTab project. ParcTabs were mobile computing devices with limited context awareness, focusing mainly on location. The system's applications were built in a client-server fashion, but the context information was made available via a *blackboard*.[1] The blackboard pattern defines a data noticeboard that all components can access: they can write new data on the board, and read and act upon any data they're interested in. The blackboard unifies sensor-based context data and higher-level inferred context, and provides a loosely coupled, extensible component interface. ParcTabs ran on proprietary hardware and weren't generally available outside Xerox.

One of context awareness's defining works was Anind Dey, Daniel Salber, and Gregory Abowd's seminal paper on the Context Toolkit,[2] which was the first coherent statement about a context-aware systems architecture. Their software follows Weiser's vision of distributing intelligence in the environment. The Context Toolkit is a set of software components that developers can use to build distributed context-aware applications using a *widget* pattern. The widget pattern is a variation of publish-subscribe: context data sources and sinks are connected to each other in a point-to-point manner. The Context Toolkit is available as source code (http://contexttoolkit.sourceforge.net), and developers have used it in many real-world contextual systems (such as Georgia Tech's Aware Home, www.cc.gatech.edu/fce/ahri). The toolkit enables a larger variety of applications than a Smartphone-based system, but it requires a special lab-like environment and specific hardware.

Recent advancements

Ian MacColl and his colleagues describe the Equip (Equator Universal Platform) system, built by the Equator Interdisciplinary Research Collaboration.[3] Equip is based on CORBA (Common Object Request Broker Architecture) and built using the blackboard pattern. Equip supports many different sensors, it can be used from C++ and Java, and researchers have field-tested it in numerous Equator IRC projects. The system is available as open source (www.crg.cs.nott.ac.uk/~jym/ect/ect.php) and is probably the most advanced, generally available context-aware platform in existence. Although Equip's resource requirements and CORBA communication make it unsuitable for current smart phones, it does run on portable general-purpose computers, such as the Hewlett-Packard iPAQ.

Panu Korpipää and Jani Mäntyjärvi[4] have described an architecture for context-aware systems running on Series 60 Smartphones built by Nokia and the Technical Research Centre of Finland (VTT).

and sends this to other users via the Jabber channel;
- the *presence listener*, which receives sensory data from others through the Jabber channel and integrates it into the applications' user interface; and
- *application customizations* (the ContextContacts and Call Log).

Researchers can combine ContextContacts with ContextLogger to study the effects of the presence service. It's important that users know whether the service is delivering accurate context information—and thus become aware of the technology's seams. Representing inaccurate or old information as new or accurate information would undermine users' trust in the system. Our working solution is to store the context data's time stamp and slowly gray out items if new data hasn't been received.

In our first field study, the ContextContacts application had two shortcomings. First, it showed the context in the Contacts list but not in the Recent Calls list, which meant that the caller didn't always have the information available. Second, the people near the callee provide a strong contextual cue about the situation, but our service didn't account for this. To address these issues, we added a replacement Recent Calls list to the customizable applications and implemented the Bluetooth environment sensor. Because many people have Bluetooth-enabled phones, this sensor is a good indirect indicator of social context.

We're running field studies to investigate how ContextContacts affects in-group awareness and calling practices. We're especially interested in how users can manage privacy with such applications. The ability to tailor the sensors, interactional patterns, and context representation are crucial to experimenting with privacy management. The system services' Status display keeps users aware of whether they're revealing personal information without having to navigate to a specific application.

ContextMedia: Sharing mobile media

Smartphones let users not only capture media but also annotate and share it. The idea of *locative media*,[9] which is media attached to a physical location, is evolving into the more generic idea of *situated media*, which is media that

Researchers have used the blackboard-based architecture to prototype content adaptation and study sensor-based context inference, and it even allows end-user-defined adaptation rules. Researchers have also field-tested many of the prototyped applications. The platform has a rich set of sensors but has fewer communication channels than our ContextPhone platform. Nokia owns the system, which can't be extended, improved, or used by outside researchers.

Nokia's Multi-User Publishing Environment (www.mupe.net) is an open-source client-server system for building context-aware applications. MUPE has a thin Java MIDP (Mobile Information Device Profile) client that presents the user interface. The server drives the user interaction, providing storage and processing resources. MUPE's thin client is both a strength and a weakness. Applications can be prototyped more rapidly than with our thick client, which requires a recompilation and installation after changes. However, a Java client has limited access to the phone features, and response times can grow because actions often require at least one network round-trip. The MUPE platform is ideal for building stand-alone applications for collaborative work and play, such as locative games. In contrast, our platform enables much tighter integration to existing applications, doesn't require constant network availability, and has much better response times for many interaction patterns. None of the applications we describe in this article could have been built with MUPE. Also, MUPE provides no sensors out-of-the-box, whereas we provide an extensive set.

Development approaches

These different approaches to building context-aware systems can be characterized along three axes: widget versus blackboard, thin-client versus native application, and smart phone versus specialized hardware versus general-purpose computers. To support our goals of timeliness, robustness, high integration to platform features, and simple implementation, as well as our aim to make the system a part of users' everyday lives, we chose to build a native application running on a smart phone and using a widget architecture. Our approach's limitations are fewer sensors and less processing power than with special hardware, longer development cycles than with thin clients, and perhaps a lack of flexibility compared to the more mature platforms.

REFERENCES

1. B.N. Schilit, *A System Architecture for Context-Aware Mobile Computing,* PhD thesis, Graduate School of Arts and Sciences, Columbia Univ., 1995.
2. A. Dey, D. Salber, and G. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human Computer Interaction*, vol. 16, nos. 2–4, 2001, pp. 97–166.
3. I. MacColl et al., "Shared Visiting in Equator City," *Proc. 4th Int'l Conf. Collaborative Virtual Environments*, ACM Press, 2002, pp. 88–94.
4. P. Korpipää and J. Mäntyjärvi, "An Ontology for Mobile Device Sensor-Based Context Awareness," *Modeling and Using Context: Proc. 4th Int'l and Interdisciplinary Conf.* (Context 2003), LNCS 2680, Springer-Verlag, 2003, pp. 451–458.

includes a description of the situation it was taken in (see http://tinyurl.com/4g2uz). ContextPhone can approximate this idea by providing additional contextual clues.

ContextMedia uses multiple sensors to annotate media, the media sensor to notice capture, and the background file upload to share the annotated media. Users can use the Jabber message channel to notify other users of sharing in real time, which also lets researchers and artists test different sharing patterns. To ensure that no data is lost even if the upload isn't immediately successful, ContextMedia stores the user input and uploader state in the phone's nonvolatile memory during the process. The media publisher uses the phone's standard, built-in features to capture media, after which an upload prompt automatically appears.

Integrating ContextMedia into existing practices reduces the learning time required for users to interact with the system. Also, phone manufacturers have extensively tested the built-in capture applications and carefully designed their user interfaces. By reusing available applications, we can add novel features in a robust manner. In comparison, a Java application such as the MUPE (Multi-User Publishing Environment) client must implement its own capture process.

ContextMedia is the result of our collaboration with the Aware project (http://aware.uiah.fi) at the University of Art and Design Helsinki's Media Lab. The Aware group is using ContextMedia to explore collective publication and syndication of mobile media. Figure 4 shows several photos with automatic and manual annotation.

The University of California at Berkeley's Garage Cinema Research Group (http://garage.sims.berkeley.edu) is studying automatic annotation and sharing of mobile media. The group originally used client-server software developed with a commercial company, but server interaction over a low-bandwidth, high-latency GPRS connection took too long. Starting the browser and fetching a simple Web page typically required about 30 seconds, whereas uploading media using ContextMedia requires as little as three seconds. The group is conducting a ContextMedia field study with 60 users.

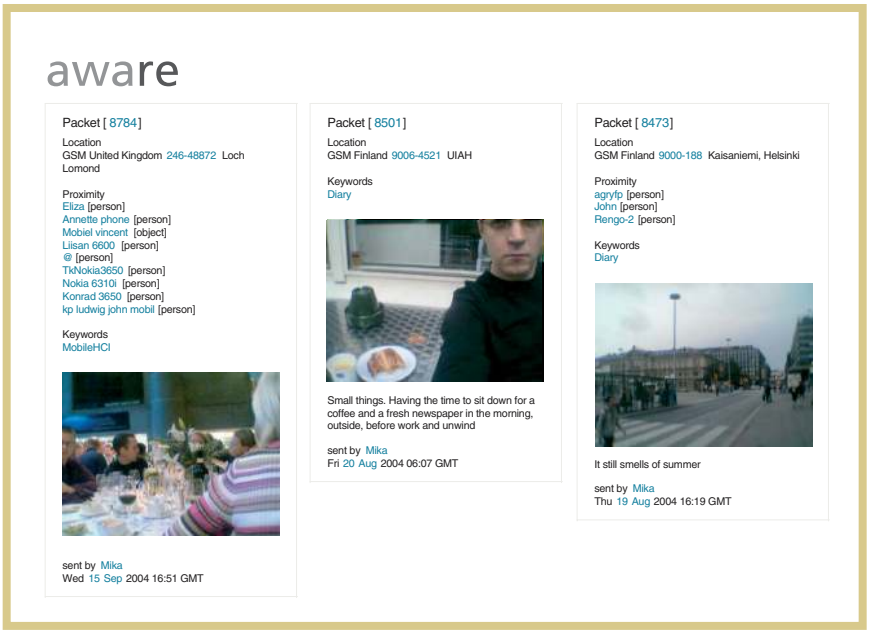

Figure 4. Mobile Weblogging—or *Moblogging*—with ContextPhone and Aware. Photos, text, and other media are annotated with context and easily uploaded and gathered. The automatic annotation includes features such as location and nearby Bluetooth devices.

By focusing on human-centered design and evaluating applications and more general mobility studies,[3] we've prioritized several key design goals for ContextPhone, including emphasizing context, unobtrusiveness, truthfulness, seamfulness,[5] timeliness, and fast interaction. Both our user-focused development process and the resulting design principles should be applicable to future context-aware systems.

Our choice of the context widget architecture[7] has proven sufficient, although not necessarily optimal. We're rewriting the communication as a blackboard, which will fully separate the concerns of what data is used and who provides it. Generic support for context-data persistence will let applications more quickly recover their world state after a crash.

We've spent much of our efforts resolving issues arising from the poor availability of interface documentation and the heterogeneity of related technologies. The resulting components are part of our concrete contribution to other researchers and practitioners in the area. ContextPhone isn't a sealed stand-alone solution. Its components can be connected via proxies to distributed, context-aware platforms such as MUPE and Equip (Equator Universal Platform), or wrapped for higher-level environments such as Java and Python on the Smartphone. Developers unfamiliar with Symbian can therefore also benefit from our work.

Because we believe that openness is necessary for the success of pervasive context-aware technologies, we've made ContextPhone freely available and published it under an open source license for anyone to use (see the "Obtaining ContextPhone" sidebar). The entire Smartphone ecosystem would benefit from opening up the platform's possibilities—not necessarily by open-sourcing, but by providing better documentation and limiting access only with respect to the phone's security-critical features. P

Obtaining ContextPhone

ContextPhone is freely available under the GNU General Public License. It can be redistributed or modified, provided that modified versions are available under the same terms.

The software works on Nokia Series 60 Smartphones (versions 1 and 2), running Symbian OS 6.0 and 7.0s (examples include the Nokia 7650, 3650, 6600, and 7610). Binaries, source code (currently approximately 50 KLOC), installation notes, and a building guide are available at www.cs.helsinki.fi/group/context/.

ACKNOWLEDGMENTS

The Academy of Finland funded the ContextPhone project under the PROACT research program. We thank John Evans and Andrew Paterson for working with us in integrating ContextPhone with the Aware platform, Beat Gfeller and Michael Rohs at ETH Zürich for the code we used for optical marker recognition, and Anthony Joseph for providing valuable feedback for this article.

REFERENCES

1. A. Oulasvirta, "Finding Meaningful Uses for Context-Aware Technologies: The Humanistic Research Strategy," *Proc. Conf. Human Factors in Computing Systems*, ACM Press, 2004, pp. 247–254.

2. G.D. Abowd and E.D. Mynatt, "Charting Past, Present, and Future Research in Ubiquitous Computing," *ACM Trans. Computer-Human Interaction*, vol. 7, no. 1, 2000, pp. 29–58.

3. P. Dourish, "What We Talk about When We Talk about Context," *Personal and Ubiquitous Computing*, vol. 8, no. 1, 2004, pp. 19–30.

4. A. Oulasvirta et al., "Interaction in Four-Second Bursts: The Fragmented Nature of

Attention in Mobile HCI," to be published in *Proc. 2005 Conf. Human Factors in Computing Systems* (CHI 2005), ACM Press, 2005.

5. M. Chalmers and A. Galani, "Seamful Interweaving: Heterogeneity in the Theory and Design of Interactive Systems," *Proc. Conf. Designing Interactive Systems* (DIS 2004), ACM Press, 2004, pp. 243–252.

6. K. Laasonen, M. Raento, and H. Toivonen, "Adaptive On-Device Location Recognition," *Proc. 2nd Int'l Conf. Pervasive Computing* (Pervasive 2004), LNCS 3001, Springer-Verlag, 2004, pp. 287–304.

7. A. Dey, D. Salber, and G. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Human Computer Interaction*, vol. 16, nos. 2–4, 2001, pp. 97–166.

8. C. Biever, "Cellphones Turn into Smart Personal Assistants," *New Scientist*, no. 2475, 27 Nov. 2004; www.newscientist.com/article.ns?id=mg18424753.100.

9. Ben Russell, *redux*, number 3 in *headmap* series, Ben Russell, London, 2003; available at www.headmap.org.

the AUTHORS

Mika Raento is a researcher in the University of Helsinki's Department of Computer Science. He received his MSc in computer science from the University of Jyväskylä and is working on his doctoral dissertation on privacy management in ubiquitous computing, focusing on social-awareness applications. He's a member of the IEEE Computer Society and ACM. Contact him at the Dept. of Computer Science, PO Box 68, FI-00014 Univ. of Helsinki, Finland; mika.raento@cs.helsinki.fi.

Antti Oulasvirta is an human-computer interaction researcher for the Helsinki Institute for Information Technology's Context Recognition project. He's completing his doctoral dissertation in cognitive science at the University of Helsinki on how interruptions are cognitively managed in mobile interaction. His other research interests include interaction design and development of field experimentation methodology for mobile and ubicomp interfaces. He's a student member of ACM SIGCHI, the European Association of Cognitive Ergonomics, and the European Society for Cognitive Psychology. Contact him at the Advanced Research Unit, Helsinki Inst. for Information Technology, PO Box 9800, FI-02015 HUT, Finland; antti.oulasvirta@hiit.fi.

Renaud Petit is a researcher in the Helsinki Institute for Information Technology's Basic Research Unit. His main research interest is information retrieval within context-aware systems. He holds an MSc in computer science from Institut National des Sciences Appliquées de Lyon, France. Contact him at the Basic Research Unit, Helsinki Inst. for Information Technology, P.O. Box 68, FI-00014, Univ. of Helsinki, Finland; renaud.petit@cs.helsinki.fi.

Hannu Toivonen is a professor of computer science at the University of Helsinki. His research interests include knowledge discovery, data mining, and analysis of scientific data with applications in genetics, bioinformatics, ecology, and mobile communications. He received his PhD in computer science from the University of Helsinki. Contact him at the Dept. of Computer Science, PO Box 68, FI-00014 Univ. of Helsinki, Finland; hannu.toivonen@cs.helsinki.fi.