PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

SARANA: language, compiler and run-time system support for spatially aware and resource-aware mobile computing

Pradip Hari, Kevin Ko, Emmanouil Koukoumidis, Ulrich Kremer, Margaret Martonosi, Desiree Ottoni, Li-Shiuan Peh and Pei Zhang

Phil. Trans. R. Soc. A 2008 **366**, 3699-3708
doi: 10.1098/rsta.2008.0127

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A

Phil. Trans. R. Soc. A (2008) **366**, 3699–3708
doi:10.1098/rsta.2008.0127
Published online 31 July 2008

SARANA: language, compiler and run-time system support for spatially aware and resource-aware mobile computing

By Pradip Hari[1], Kevin Ko[2], Emmanouil Koukoumidis[2], Ulrich Kremer[1], Margaret Martonosi[2,*], Desiree Ottoni[1], Li-Shiuan Peh[2] and Pei Zhang[2]

[1]*Department of Computer Science, Rutgers, The State University of New Jersey, Piscataway, NJ 08854, USA*
[2]*Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, USA*

Increasingly, spatial awareness plays a central role in many distributed and mobile computing applications. Spatially aware applications rely on information about the geographical position of compute devices and their supported services in order to support novel functionality. While many spatial application drivers already exist in mobile and distributed computing, very little systems research has explored how best to program these applications, to express their spatial and temporal constraints, and to allow efficient implementations on highly dynamic real-world platforms. This paper proposes the SARANA system architecture, which includes language and run-time system support for spatially aware and resource-aware applications. SARANA allows users to express spatial regions of interest, as well as trade-offs between quality of result (QoR), latency and cost. The goal is to produce applications that use resources efficiently and that can be run on diverse resource-constrained platforms ranging from laptops to personal digital assistants and to smart phones. SARANA's run-time system manages QoR and cost trade-offs dynamically by tracking resource availability and locations, brokering usage/pricing agreements and migrating programs to nodes accordingly. A resource cost model permeates the SARANA system layers, permitting users to express their resource needs and QoR expectations in units that make sense to them. Although we are still early in the system development, initial versions have been demonstrated on a nine-node system prototype.

Keywords: mobile computing; programming interfaces; run-time systems

1. Introduction

Today's mobile systems are reaching a turning point. Until recently, common operation has involved one mobile device (e.g. a laptop) seeking to remain connected with mostly fixed infrastructure, or sometimes two mobile devices

* Author for correspondence (mrm@princeton.edu).

One contribution of 19 to a Discussion Meeting Issue 'From computers to ubiquitous computing, by 2020'.

(e.g. cellular telephones) seeking to communicate or share data with each other. Increasingly, however, new mobile applications are emerging in which larger confederations of mostly mobile devices might collaborate to share information or resources. For example, many social networking applications involve cooperative information or resource sharing across a collection of cellular telephone or mobile device users (Shen & Li 2007). Distributed gaming scenarios often involve people with handheld devices communicating data and game status while ranging widely across a region (Wipeer 2007). While collaborative applications on wireless devices already exist (Aalto *et al.* 2004; Wipeer 2007), there still remain considerable difficulties in programming them to work resiliently across a large range of conditions.

These collaborative applications have several key characteristics in common.

— *Dynamic confederations of devices.* The applications represent loose collaboration between a fairly large number (tens or hundreds) of mobile devices. By loose collaboration, we mean that (much like a peer-to-peer system in other domains) the devices have chosen to come together to share resources or information, but each device is entitled to drop out at any time, or to refuse to share some of its resources. These collaborative groups have very dynamic membership, and each device in the group may have independent policy goals.
— *Resource awareness.* For distinct nodes to be willing to collaborate, there must be mechanisms for allowing them to estimate resource usage and account for the time and energy costs involved in offering a service to another node.
— *Spatial awareness.* Another key attribute is that spatial information (i.e. current positions of the nodes and their relationship to each other) is usually needed as a 'first-class' citizen in the system design.

Given the predominance of these mobile applications, and the difficulty in programming them using traditional languages, this paper argues for language and run-time system constructs specifically aimed at supporting them. In particular, this paper introduces SARANA, currently being developed to provide support for a spatially aware, resource-aware networking architecture.

At the language level, SARANA's key features are (i) support for programmers to express geographical regions of interest and computation/communication patterns based on such regions, and (ii) support for programmers to express policies to guide trade-offs of quality of result (QoR), performance, energy costs and other resource costs. Programmers need only to specify abstractly the QoR rather than hardwiring application-specific attributes such as the number or addresses of devices to be involved in the computation. Likewise, SARANA allows applications to avoid including explicit resource-cost management logic, while still ensuring that program executions adhere to a predetermined cost target.

SARANA's run-time system guides the decisions on how many mobile devices to contact, based on programmer-specified preferences about QoR trade-offs between result quality, latency and cost. While individual nodes can always opt out of the confederation, SARANA allows the computation to adapt resiliently to variations in neighbour count, battery levels, communication latencies and other 'facts of life' in mobile computing. SARANA's run-time cost management determines which and how many remote devices will be used in the computation

so as to meet (if possible) the programmer's QoR requirement within the imposed cost constraint. In addition, the run-time system brokers relationships between nodes. It selects possible candidate nodes for execution or sharing, based on both geography and resource availability. It also performs dynamic cost feedback management; that is, it dynamically adjusts cost estimates and remote node candidates as an application is in progress. This allows SARANA to get the best possible QoR within a cost budget, while abiding by each node's choice to opt out of remote computations.

The remainder of this paper is organized as follows. Section 2 gives more details of the SARANA system, couched in the context of a driver example patterned after the 'Amber Alert' emergency system in the USA. Section 3 describes the SARANA programming language. Section 4 gives an overview of the SARANA run-time system. Section 5 offers comparisons with related work, and §6 gives our conclusions.

2. Driving example: Amber Alert

In order to illustrate the different features of SARANA, we describe here an example of its use for an Amber Alert system. Amber Alert is a system in the USA that is used to notify the population when a child is kidnapped. It creates a cross-medium announcement targeting specific geographical regions to aid the search for the kidnapper, the child, or a particular vehicle or license plate number. Typically, notifications are made via television or radio broadcasts and highway information signs. In addition, individual cellphone users may 'opt in' to receive Amber Alerts as SMS messages. Citizens with information concerning the situation are urged to call the police. In our case, we show how SARANA could be used to create a more dynamic and intelligent version of the Amber Alert system. The application combines spatial awareness, in-network processing and cost/QoR trade-offs. A partial listing of SARANA code is shown in figure 1.

(*a*) *Amber Alert problem overview*

A typical Amber Alert search profile might be an announcement that people in a particular region should be looking for a red four-door sedan, with a particular license plate number. A photograph of the kidnapped child, or a sketch of the likely suspect, may also be provided.

— *Spatial awareness*. An Amber Alert is issued for a particular physical region, such as a city, county or state. Our goal is to harness resources within this region, such as cameras and cellphone users, to help identify the missing child or car. Spatial awareness allows Amber Alert to focus within the region of interest, not using resources far from the incident.
— *Resource awareness*. Any given target region is likely to have many distributed cameras and compute devices. These cameras may be fixed, such as traffic surveillance cameras, or mobile, such as those in nearly every cellphone today. Our goal is to launch code on several cellphones and camera nodes in order to harness these cameras as distributed search resources. Since launching everywhere would be very resource hungry, we must abide by cost or credit constraints.

```
1  import SpaceDefs.*; // includes absolute space definition for TargetCity

2  public class ImageAquisisitionAndUnderstanding {
3      public static void main(String[ ] args) {
4          SearchAttributes searchAttr = new SearchAttributes();
5          searchAttr.parse(args); // record attributes of search target
6          Container carInfo = new Container();
7          spatialview sv1 = camera @ TargetCity;
8          visiteach cam : sv1 every 30 secs within 3 hours {
9              Reduction boolean successfulMatch = false;
10             Image img = cam.getImage(); // acquire image
11             Time time = System.currentTime(); // time of image acquisition
12             Location loc = System.currentLocation(); // location of camera
13             spatialview sv2 = imageUnderstandingCode @ new Circle(loc, 200m);
14             visitone imageAnalysis : sv2 {
15                 if (imageAnalysis.processImage(searchAttr, img) = match) {
16                     successfulMatch = true; } }
17             if (successfulMatch) {
18                 carInfo.addElement(loc, time, img);
19                 spatialview sv3 = AmberAlertParticipant @ new Circle(loc, 100m);
20                 visiteach participant : sv3 {
21                     participant.display(loc, time, img) } }
22         }
23         carInfo.displayAll(); // display all images on injection node
24         obtain( #participant.display / #successfulMatch = 10) ; // QoR specification
25         } }
```

Figure 1. Sketch of Amber Alert driver application.

— *Quality-of-result trade-offs.* The Amber Alert application illustrates the sorts of QoR trade-offs that many such distributed applications face. A broadcast to every relevant node in the region of interest might be low latency, but high cost. A parsimonious sequential query of each camera in the city would be lower cost, but with longer latency and lower likelihood of success. SARANA's run-time system will balance these goals by deciding how many nodes to launch and by sampling these nodes across the region of interest.

In our example, we assume that the entire program terminates after 3 hours, i.e. the search is aborted. At this point, if the suspect has not yet been found, police could relaunch the program with a new larger region of interest.

(*b*) *SARANA code example*

The code for Amber Alert is shown in figure 1. Every 30 s, a picture is taken across a sampled subset of cameras (line 8). Determining which cameras to use (and how many) is a key aspect of SARANA's QoR-based run-time system. Each camera image is then processed by a computation node within a physical neighbourhood, if available, which has the needed image understanding code installed (line 14). This local in-networking processing reduces communication costs and avoids contention.

The image understanding code receives and processes the image using particular Amber Alert search attributes. If the search attributes match the image content, the camera image is distributed to a set of Amber Alert participants near the

camera (line 19). (To simplify the example, the physical neighbourhood has been hard-coded into the program, but it could be adaptive or a dynamic parameter.) Here, individuals in close proximity to the potential target sighting can confirm or refute the positive identification of the target, and give additional information, if possible (e.g. the license plate number). In order to keep the example small, we omit the code that allows Amber Alert participants to annotate images with additional information. All potential target matches are collected in a data structure (`carInfo`) and displayed at the emergency centre that originally initiated program execution (line 23).

The `obtain` construct (line 24) in our Amber Alert code allows the user to specify an acceptable quality of a program execution. In this example, the programmer wishes to contact 10 Amber Alert participants for each image that has been identified as showing the search target. The rationale here could be that getting information from 10 people will help to confirm a positive identification. Having too many responses to the displayed image would produce redundant information and would lead to a significant waste of precious resources, such as battery life and an Amber Alert participant's time (social cost). The SARANA run-time system seeks to satisfy all `obtain` QoR requests while not exceeding the specified overall budget.

(*c*) *QoR possibilities for Amber Alert*

To accomplish the QoR goal, the run-time system has to pick a number of cameras (line 8) and a number of Amber Alert participants (line 20) for each successfully matched camera image. All this must be done in a dynamic environment where the costs for services such as taking or analysing a photograph may vary and are not known at program invocation time. As described further in §3, the SARANA run-time system is novel in its use of incremental execution of nested spatial iterations, together with a probing strategy that determines the overall costs of a single iteration, or a small set of iterations. Based on this probe information, the overall execution cost is limited to within a preset budget while honouring the quality constraints specified in the `obtain` construct.

3. Language and compiler overview

SARANA's main program abstraction is a space–time region of virtual network nodes that provide specific services within a physical space and within a time interval. This space–time region is called a *spatial view*. The spatial view definition is declarative in nature, i.e. it does not instantiate any mapping between virtual and physical nodes. This is done through the *iterator* (`visiteach`).

Conceptually, the body of a `visiteach` statement is executed on every network node in a given space. In practice, the actual number of remote nodes that can be used will be limited by the availability of various system resources. In SARANA, resources are assigned *costs* and applications are given *credits* with which to 'purchase' them. Proper management of a credit budget is not an easy task, owing to the inherent unpredictability and volatility of ad hoc networks. The programmer may attempt to estimate the numbers and prices of remote

nodes through performance testing and tuning, but the actual environment faced by the application after deployment may differ radically from that of the test-bed. An 'iteration controller' is intended to mediate between the application and the policy controller to handle the dynamic trade-offs of QoR and cost within SARANA budgets. In a SARANA program, QoR is defined with the `obtain` construct. QoR expressions may contain quantities such as the sizes of collections produced by the program or the number of times that a particular remote service is invoked. In the current implementation, a time constraint can be imposed as an overall deadline. If QoR is given as an expression, the run-time system will attempt to maximize the value of that expression. We are exploring incremental execution strategies in which some of the budget is spent on initially sampling nodes across the space; depending on the results of this initial probe, additional budget may be spent to get improved quality.

We are currently implementing a prototype of the SARANA compiler as an extension of the JAVA compiler in the JAVA Development Kit v. 1.7, but have already explored our language and run-time features with modest, hand-built SARANA programs.

4. SARANA's run-time system

The key idea behind SARANA is to hide from application programmers the low-level issues related to communication, resource management and dynamic resource discovery/optimization, while still allowing them to express abstract QoR preferences regarding how these issues affect the quality of their program output.

The execution of a SARANA application begins at the launch node. Based on its available credit budget and local energy resources, the application will contact a certain number of remote nodes. To decide which nodes to contact, it queries a 'directory server' to find nodes that meet the desired criteria in terms of resources available as well as spatial position. It then issues launch requests and waits to gather the results.

On the other side, the 'launchees' have the same set of active components, but they function differently. When the launchee receives a request to locally execute a task, it first checks with the local policy controller to see if it has sufficient resources to satisfy the request. If permission is granted, it next determines whether or not necessary code modules reside on the node; if any are missing, the code module distribution manager must request them. Finally, it spawns the task as a new SARANA process/application and monitors its execution. The spawned task may further wish to contact remote nodes to launch subtasks on them. If this happens, the launchee becomes at the same time a launcher, with activities as previously described.

(*a*) *Directory server*

The run-time QoR libraries query the directory server for discovering nodes that meet the location and resource constraints for execution. SARANA therefore includes a directory service that is updated periodically by active nodes with their location, the services that they might provide (e.g. camera, image analysis), and costs for each service. Thus far, we have implemented two directory services. One is based on a centralized scheme and the other is a

distributed hash table approach. The directory server is, on its own, a rich research problem with interesting issues related to implementing the 'range queries' needed to support SARANA's service model. We are currently exploring distributed search structures to support these queries.

Of significant interest is that the directory only needs to be approximately correct, in terms of service availability or node communication address. If node addresses are stale (e.g. owing to mobility), packets will be dropped, but the QoR approach allows the system to be resilient to this. Service availability can also be somewhat stale, because once a node is contacted for SARANA execution, it can still choose not to participate at that time. This resilience to staleness offers more flexibility in how the directory server is implemented and kept consistent.

(*b*) *Policy controller*

Applications running on SARANA track resource usage for accounting purposes. Given our execution structure, we use a three-tiered approach to track available resources, with increasing accuracy as the application approaches the execution location. The directory server, described above, gives a rough estimation of resources. The policy controller provides an *estimate* of device cost before execution. Finally, the policy controller provides the *true* cost after task completion. It also updates system information regarding the node's location and resource availability. This approach provides a reasonable trade-off between accuracy and overhead.

The policy controller predicts the cost of the task before device execution in order to provide an updated estimate for the system. Since the correlation between resource usage and different executions of the same task is strong, we choose a history-based approach to learn the cost of each task. In our approach, a cache holds previous credit usages of the same task, with histories of old tasks evicted if left unused for a certain interval.

5. Related work

Being a broad effort encompassing language, systems and resource monitoring issues, SARANA draws from many prior research topics. We discuss the most closely related projects here.

(*a*) *Languages and compilers*

In recent years, programming support for sensor networks has become a hot research area (Hill *et al.* 2000; Levis 2002; Blum *et al.* 2003; Boulis & Han 2003; Gay *et al.* 2003; Liu 2003; Welsh 2004; Whitehouse *et al.* 2004; Kothari *et al.* 2007; Newton & Morrisett 2007). Several prior works such as Regiment, Hood and others have similar objectives as SARANA, but focus on the context of tightly connected sensor networks, where nodes are specifically deployed to collaborate, rather than SARANA's loose confederation of nodes that occasionally choose to share services. Spatial Programming (Borcea *et al.* 2004) and SpatialViews (Ni *et al.* 2005) are related spatial programming efforts. One key distinction between them and SARANA is that they do not allow the cost-based adaptations of code execution that SARANA does. The high-level

SARANA model is built on SpatialViews for convenience, but introduces novel QoR specifications previously unavailable. The use of a cost model as part of the language has also been proposed previously by Mainland *et al.* (2004) at Harvard.

(*b*) *Mobile computing infrastructure*

The mobile phone industry is well known for closed hardware and proprietary software that make it difficult for third parties to develop software for these devices. Nonetheless, significant prior work has offered software development kits (SDKs) for such platforms. These include the Open Handset Alliance's Android (Google Inc. 2007), Qualcomm's BREW (Qualcomm 2007), Apple iPhone's SDK (Apple 2007) and OpenMoko (2007). The key difference between SARANA and these SDKs is that SDKs simplify the programming of a single mobile phone, while SARANA eases distributed programming of a mobile computing substrate consisting of many mobile phones. SARANA can be developed on top of such SDKs, to simplify the development of SARANA's run-time system itself.

Various other systems support features similar to those of SARANA. For example, Combine (Ananthanarayanan *et al.* 2007) is an infrastructure for supporting collaborative downloading on wireless mobile devices. Like SARANA, it uses credits for tracking execution and energy costs of collaborative applications. However, it is targeted towards the specific application of collaborative downloading, while SARANA is broader in scope. Likewise, Slingshot (Su 2005) supports remote code distribution and replication, with goals similar to that of the code module distribution manager of SARANA, but at a coarser granularity of entire applications.

(*c*) *Cluster and volunteer computing*

Moving from the wireless domain into more high-infrastructure wired environments, there have been other prior works on run-time system support for automatically scheduling tasks onto large clusters. These include Google's MapReduce (Dean 2004) and Microsoft's Dryad (Isard *et al.* 2007). Clearly, similarities exist between SARANA and these run-time systems: they all schedule execution of tasks onto compute nodes and manage the transfer of input and output data. Since SARANA targets mobile devices, it tackles problems that are not major issues in a data centre environment: diverse geographical locality, bandwidth constraints, extreme heterogeneity of systems and user budgets, and failures being the norm rather than a rarity.

Likewise, volunteer computing is where the general public volunteer their computing and networking resources for scientific research projects (Bayanihan Computing Group 2007; Berkeley Open Infrastructure for Network Computing 2007; Seti@Home 2007), creating a distributed computing substrate. Similar to SARANA, volunteer computing is enabled by a run-time system infrastructure that autonomically schedules and launches applications onto volunteered computers. An incentive mechanism is also employed to promote the continued donation of resources. At a superficial level, SARANA's deployment scenario (i.e. mobile, handheld nodes in a geographical region versus largely fixed PCs across countries) cultivates a differing set of design choices for its run-time system. A stronger contrast, however, lies in the tight coupling between

SARANA's programming language, compiler and run-time system. SARANA's integration with the programming language provides an opportunity to ease distributed programming beyond what is possible with a development library.

6. Status and conclusions

Overall, this paper has argued for a holistic approach in supporting spatially aware, resource-aware mobile distributed applications. A key aspect of our approach is support for QoR trade-offs. The SARANA language supports programmer abstractions to specify spatial regions, as well as trade-offs between quality, resource costs and latency. Underneath this abstraction, the SARANA run-time system provides best-effort results subject to cost, energy and connectivity constraints.

Although our system development is still under way, we have tested our ideas using a preliminary nine-node test-bed, as well as simulations of larger-scale deployments. Our preliminary results show that SARANA's ability to respond dynamically to measured execution costs and to adjust execution choices dynamically is much more efficient than that of other less abstract approaches. In particular, across a range of possible cost distributions, SARANA is better able to use up available credits without exceeding the budget and is able to produce higher-quality results in turn.

Overall, this work demonstrates the value of language-level QoR abstractions for mobile systems, paired with run-time optimizations. While 'hard-wired' application-specific approaches remain an option, they rarely prove as agile in finding high-quality solutions across a range of dynamic settings. This work represents an important step towards easing the programming of truly robust and portable mobile distributed applications.

This work has been supported in part by the US National Science Foundation through grants CNS-0615175 and CNS-0614949. In addition, we acknowledge further partial support from Nokia Research.

References

Aalto, L., Göthlin, N., Korhonen, J. & Ojala, T. 2004 Bluetooth and WAP push-based location-aware mobile advertising system. In *MobiSys'04: Proc. 2nd Int. Conf. on Mobile Systems, Applications, and Services*, pp. 49–58.

Ananthanarayanan, G., Padmanabhan, V. N., Ravindranath, L. & Thekkath, C. A. 2007 Combine: leveraging the power of wireless peers through collaborative downloading. In *MobiSys'07: Proc. 5th Int. Conf. on Mobile Systems, Applications and Services*, pp. 286–298.

Apple 2007 iPhone Development Center. See http://developer.apple.com/iphone/devcenter/.

Bayanihan Computing Group 2007. See http://bayanihancomputing.net/.

Berkeley Open Infrastructure for Networking Computing 2007. See http://boinc.berkeley.edu/.

Blum, B., Nagaraddi, P. Wood, A., Abdelzaher, T., Son, S. & Stankovic, J. 2003 An entity maintenance and connection service for sensor networks. In *MobiSys'03: Proc. Int. Conf. on Mobile Systems, Applications and Services.*

Borcea, C., Intanagonwiwat, C., Kang, P., Kremer, U. & Iftode, L. 2004 Spatial programming using smart messages: design and implementation. In *Int. Conf. on Distributed Computing Systems* (*ICDCS'04*), *Tokyo, Japan, March 2004.*

Boulis, A., Han, C. & Srivastava, M. 2003 Design and implementation of a framework for efficient and programmable sensor networks. In *MobiSys'03: Proc. Int. Conf. on Mobile Systems, Applications and Services*.

Dean, J. & Ghemawat, S. 2004 Mapreduce: simplified data processing on large clusters. In *Symposium on Operating Systems Design and Implementation* (*OSDI'04*), pp. 137–150.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E. & Culler, D. 2003 The nesC language: a holistic approach to networked embedded systems. In *Proc. ACM SIGPLAN 2003 Conf. on Programming Language Design and Implementation* (*PLDI*).

Google Inc. 2007 Android—an open handset alliance project. See http://code.google.com/android/.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. & Pister, K. 2000 System architecture directions for network sensors. In *Int. Conf. on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*).

Isard, M., Budiu, M., Yu, Y., Birrell, A. & Fetterly, D. 2007 Dryad: distributed data-parallel programs from sequential building blocks. In *European Conf. on Computer Systems* (*EuroSys*), *Lisbon, Portugal, 21–23 March 2007*, pp. 59–72, also as MSR-TR-2006-140.

Kothari, N., Gummadi, R., Millstein, T. & Govindan, R. 2007 Reliable and efficient programming abstractions for wireless sensor networks. In *The ACM SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI'07*), *San Diego, CA, June 2007*, pp. 200–210.

Levis, P. & Culler, D. 2002 Maté: a tiny virtual machine for sensor networks. In *Int. Conf. on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS*).

Liu, T. & Martonosi, M. 2003 Impala: a middleware system for managing autonomic parallel sensor systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (*PPoPP*).

Mainland, G., Kang, L., Lahaie, S., Parkes, D. C. & Welsh, M. 2004 Using virtual markets to program global behavior in sensor networks. In *Proc. 11th ACM SIGOPS European Workshop, Leuven, Belgium*.

Newton, R., Morrisett, G. & Welsh, M. 2007 The regiment macroprogramming system. In *IPSN'07: Proc. 6th Int. Conf. on Information Processing in Sensor Networks*, pp. 489–498.

Ni, Y., Kremer, U., Stere, A. & Iftode, L. 2005 Programming ad-hoc networks of mobile and resource-constrained devices. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI'05*), *Chicago, IL, June 2005*.

OpenMoko 2007. See http://www.openmoko.com/.

Qualcomm 2007 BREW. See http://brew.qualcomm.com/brew/en/.

Seti@Home 2007. See http://setiathome.berkeley.edu/.

Shen, G. & Li, Y. & Zhang, Y. 2007 Mobius: enable together-viewing video experience across two mobile devices. In *MobiSys'07: Proc. 5th Int. Conf. on Mobile Systems, Applications and Services*, pp. 30–42.

Su, Y.-Y. & Flinn, J. 2005 Slingshot: deploying stateful services in wireless hotspots. In *MobiSys'05: Proc. 3rd Int. Conf. on Mobile Systems, Applications, and Services*, pp. 79–92.

Welsh, M. & Mainland, G. 2004 Programming sensor networks using abstract regions. In *Symposium on Networked Systems Design and Implementation* (*NSDI*), *March 2004*.

Whitehouse, K., Sharp, C., Brewer, E. & Culler, D. 2004 Hood: a neighborhood abstraction for sensor networks. In *MobiSys'04: Proc. 2nd Int. Conf. on Mobile Systems, Applications, and Services*.

Wipeer 2007. See http://www.wipeer.com/.