Programming Computers Embedded in the Physical World *

Liviu Iftode
Andrzej Kochut

Department of Computer Science
University of Maryland, College Park
College Park, MD 20740
{iftode, kochut}@cs.umd.edu

Cristian Borcea
Chalermek Intanagonwiwat
Ulrich Kremer

Division of Computer and Information Sciences
Rutgers University
Piscataway, NJ 08854
{borcea, intanago, uli}@cs.rutgers.edu

Abstract

Future ubiquitous computing environments will consist of massive, ad hoc networks of embedded systems deployed in the physical space. Programming such environments requires new abstractions and computing models. This paper presents Spatial Programming (SP), a novel paradigm for programming ubiquitous computing environments. SP offers access to data and services distributed on nodes spread across the physical space in a similar fashion to access to memory using references. SP programs access network resources using a high level abstraction, termed spatial reference, which names these resources using their spatial properties and content-based names. An underlying system takes care of mapping spatial references onto target nodes in the network. Thus, SP hides the complexity of the underlying network from the programmer, while offering a simple and intuitive programming model.

1. Introduction

Recent advances in technology made it feasible to create networks of embedded systems (NES). Such networks will represent the infrastructure for future ubiquitous computing environments [24, 20]. For instance, sensors monitoring the environment [12, 10, 11], robots with intelligent cameras collaborating to track a given object [1], or cars on a highway cooperating to adapt to traffic conditions [2] will become a daily reality. Although extremely heterogeneous, nodes in NES will have a common set of characteristics: (1) incorporate processing power that allows them to perform local computations, (2) are able to communicate with other devices using short-range radio, (3) are spatially distributed and mobile, and (4) work unattended and may fail frequently.

NES will consist of a huge number of small-size devices deployed in the physical space. These nodes may join or leave at any moment (becoming unreachable due to mobility, energy depletion, failures, or disposal) leading to dynamic and volatile network topologies. Therefore scale and volatility will be the main issues in NES. In many scenarios, it will not be feasible to deploy more powerful nodes that act as base stations, thus NES will have to function in a complete decentralized manner.

All these characteristics make NES programmability a challenging task. Traditional distributed programming models, such as message passing are not suitable for NES. Aspects of mobility, volatility and scale make it difficult to program applications as a series of interactions with devices identified by fixed addresses (as it is assumed in the message passing model). Instead, we want to gather information or perform actions on devices that have certain properties, content, or location. Thus, we are no longer interested in contacting a particular device, but any device that may be useful to perform the task. Simpler programming abstractions and more flexible programming models are needed to let the programmers design and write their applications without concerning about the underlying network details.

In this paper, we propose Spatial Programming (SP), a novel paradigm for programming ubiquitous computing environments that emphasizes the notions of space and content. In SP, space is a first order pro-

*This work is supported in part by the NSF under the ITR Grant Number ANI-0121416

gramming concept that needs to be exposed to applications. Also, SP shields the complexity of programming volatile, ad hoc networks by presenting a high level uniform abstraction, termed *spatial reference*, for naming and accessing network resources. A spatial reference names network resources using their geographical location and content, while an underlying system takes care of mapping it onto a target node in the network. The mappings between spatial references and nodes in the physical world are similar to the mappings from virtual memory to physical memory in a conventional computer system.

Although both geographical routing [14, 17, 18] and content-based naming and routing [4, 9, 13] have been extensively studied, a simple and intuitive programming model that allows the user to express the computation in terms of these abstractions is still missing. In our model, programmers write simple sequential programs and access transparently the network resources (i.e., using spatial references) in a similar fashion to the access to memory using variables. SP is independent of the underlying system, thus allowing for multiple implementations. Each implementation has to provide an SP compiler that is responsible for translating the high-level sequential program into a series of actions that will be performed by the underlying system.

The rest of this papers is organized as follows. Section 2 presents an example that motivates Spatial Programming. Section 3 describes the design principles of SP. Section 4 shows the basic programming constructs for SP and illustrates them with an example. Section 5 discusses the related work. The current status and future plans are presented in Section 6.

2. Motivating Example

As an example that motivates Spatial Programming, we present the task of performing object tracking over a large area using a set of spatially distributed intelligent cameras (they are battery powered and use wireless for communication). In Figure 1, we have two spaces, *campus* and *town*, a number of cameras presented as nodes marked with *camera*, and a black square that represents the object of interest. Each device is capable of determining its location (i.e., using GPS or other localization methods [16, 22, 7]) and remains static after deployment. However devices may fail, or be deployed far from other devices preventing them from participating in the computation. The user requires that the images should be taken from N different cameras that are present in the area of interest (*campus* in this example). The application has to dynamically determine the location of the camera that provides the best identification for the desired object. During this process, the application uses partial results computed at other nodes and it needs to be able to reach nodes already visited in order to execute various computations there.

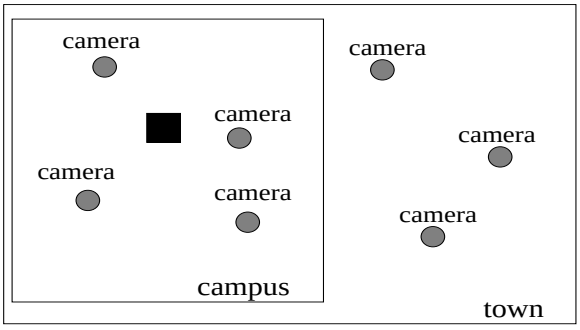


Figure 1. Motivating Example

The task stated above is difficult and tedious to program using traditional programming models. The programmer would have to explicitly manage the communication between devices and to program all the details involved in reaching the area of interest and contacting the target nodes located there. Additionally, the network dynamics (caused by node failures or new cameras being deployed) may cause the application to fail since fixed addressing schemes treat exceptions as failures.

Expressing the same task in our model is straightforward. SP paradigm defines a simple abstraction, called **spatial reference**, that presents a unified view of space and network to applications. A spatial reference is a reference to a node in SP, represented as a tuple **{space:tag}**. The **tag** represents a content-based name for the node. The **space** is a geographical scope for this content-based name of the node.

The source code for the program that realizes the desired task is presented in Figure 2. Cameras take images continuously, but their processor can execute a single identification task at a time by focusing on one object. The application computes the object identification on N free cameras (i.e., they are not currently focusing on any object) using spatial references and partial results accumulated during execution (lines 3-5). The expression {*campus:camera*} is a reference to a camera that is located in *campus*. The array subscript *[i]* is used to refer to different cameras with the same space-content description. Only the camera that provides the best partial result is instructed to focus on the desired object for a potentially complete identification. Once a new camera provides a better result, it will become the active camera (i.e., the one that focuses on the object) while the old one will become free (lines 6-10). The application may access the in-

```
1 object_tracking(object){
2   partialIdentification = 0; max_id = 0;
3   for(i=0; i<N; i++){
4     if ({campus:camera[i]}.focus == NONE){
5       result = identify({campus:camera[i]}, partialIdentification, object);
6       if (result > partialIdentification){
7         {campus:camera[max_id]}.focus = NONE;
8         {campus:camera[i]}.focus = object;
9         partialIdentification = result;
10        max_id = i;
11      }
12  }
13  if (completeIdentification({campus:camera[max_id]}))
14    return {campus:camera[max_id]}.location;
15 }
```

Figure 2. SP Code Example

formation and services on the target nodes using the *dot* notation (e.g., {campus:camera[i]}.focus). Detailed descriptions for all SP constructs used in this example are presented in Section 4. The application finishes by checking the images acquired by the camera that provided the best overall partial result. If a complete identification is found, the function returns the location of that camera (lines 13-14).

3. Design Principles

Networks of embedded systems (NES) are by definition very large, volatile, and spread across the physical space. To achieve their objectives, the applications running over such networks need to execute within certain geographical regions. These applications are interested in content and properties, not in the nodes that provide them. Also, they need to be able to adapt for the uncertainty encountered in NES, where the topologies and the resources available are unknown a priori.

Spatial Programming (SP) is a novel programming paradigm that attempts to address these issues. SP is based on four main design principles: (1) exposing space as a first order programming concept, (2) decoupling the access to network resources from networking details, (3) maintaining reference consistency, and (4) supporting programming for uncertainty.

Applications should be able to access resources located on the nodes participating in NES by referring to them in the same fashion a program addresses the memory using variables. The reference should involve spatial characteristics of the resource (i.e., the geographical region of the node containing the resource), and content characteristics of the resource (i.e., content that has to be present on the node). The application programmer however, should not have to perform any specific network-related operations to obtain the requested resources. An underlying system has to take care of name resolution, access to resources, routing, and communication.

The application that refers to a node with given spatial and content properties should be guaranteed to contact the same node each time it makes subsequent successful references to the same space-content description. Therefore, the underlying system needs to maintain bindings between these references and the nodes addressed by them. These bindings are maintained in a *per-application binding table*, which plays a similar role as a page table in a traditional computer system. Once a reference to a network resource is performed, the corresponding spatial reference is bound to a particular node in that space. The binding is persistent for the duration of the SP program execution. We call this feature reference consistency and it allows the programmer to visit repeatedly nodes and locations as long as the spatial reference is valid.

Since NES are extremely volatile, SP should make it easy for the programmer to deal with network configuration dynamics. This dynamics involves constant change in the location of nodes as well as intermittent network connectivity. For example, a reference made to a node may become invalid if this node has moved away from the area of interest. Another possibility is that a node failure leads to unavailability of requested resources. In such situations, the underlying system should allow the application to adapt to adverse network conditions.

For most types of applications it is important to guarantee execution completion within a predefined time period. In such a dynamic environment as NES,

where many operations depend on location-based and content-based routing, a lookup operation for a reference may take a substantial amount of time. To bound the execution time, each reference specifies a timeout value after which the attempt to access it is aborted and the application is informed. Thus, the application will be able to decide about its further actions.

4. Spatial Programming Constructs

Spatial Programming requires a set of programming constructs that can be added as extensions to any programming language. The main SP construct provides transparent access to network resources as well as an exception mechanism that allows applications to adapt to network configuration dynamics. The other constructs offer the possibility to create/remove network resources during execution, to define relative spaces dynamically, and to change the space for a given spatial reference.

4.1. Accessing Network Resources

In SP, network resources can be accessed using a quintuple {*space:tag[instance], timeout*}.*resource* that specifies: (1) the spatial information for a node of interest, (2) the content-based name for this node, (3) the instance of a particular node with these spatial and content-based properties, (4) an upper bound on the access time for the desired resource, and (5) the resource name at the node.

Spatial References. A *spatial reference* is a {*space:tag*} tuple that refers to a node located in *space* and named by content using *tag*. The *space* is a geographical scope for the content-based name of the node being addressed. Spatial references allow transparent access to resources located on the referenced nodes. In the example presented in Figure 1 we have two spaces, *campus* and *town*, and each node hosts a *camera* tag. The reference {*campus:camera*} represents one of the intelligent cameras that are in the campus.

Spatial Reference Instances. To make it possible to refer to more than one node with the same spatial and content properties we introduce the notion of *reference instance*. The reference instance denotes a particular index of the node that is referenced by the spatial-content description. For example, in Figure 1, {*campus:camera[0]*} and {*campus:camera[1]*} denote two different camera nodes located in *campus*.

Access Timeout. In such a volatile environment, it is difficult to estimate how long it takes to access a network resource (i.e., even the existence of a certain resource in a given space is unknown). Nevertheless, an application may need to achieve its objectives within a fixed time interval. Although we cannot offer real time guarantees in these highly dynamic networks, SP makes it possible to specify a soft deadline for looking up a spatial reference (i.e., the maximum time that may be spent for that operation). If the target node is not reached, the underlying system raises a timeout violation exception and the application will decide about further actions.

Resource Naming. In SP, applications can name and access any resource located on a node referenced by a spatial reference. In order to access these resources, the *dot* notation is used. The construct {*space:tag*}.*resource* instructs the system to access the *resource* located on the node referenced by {*space:tag*}. In the example from Figure 1, {*campus:camera[i]*}.*focus* represents the object observed by this camera. Similarly, {*campus:camera[i]*}.*location* may represent the location of this node in space.

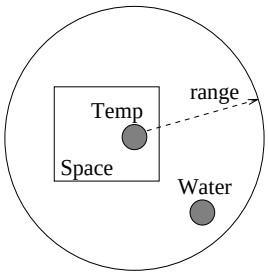
4.2. Creating/Removing Network Resources

Besides accessing resources that already exist at nodes, a program may need to dynamically create/remove its own resources. The primitives that offer this functionality are: *create(*{*space:tag[i], timeout*}.*resource)* and *remove(*{*space:tag[i], timeout*}.*resource)*. The binding for the spatial reference {*space:tag[i]*} must exist when these functions are called. For instance, an application may need to create new resources in order to store data in the network (i.e., similar to creating files in a file system), or it may even create new services on nodes of interest (e.g., an image recognition service on a *camera* node).

4.3. Defining New Spaces

To allow for flexible specifications of spaces, SP supports basic operations on spaces. The application may use statically defined spaces (such as *campus* or *town*) or create new *composed spaces* using union, difference or intersection operators. In our example from Figure 1, the reference {*(town - campus):camera*} returns a camera node located in *town*, but not in *campus*.

Defining relative spaces based on the position of a node referenced by a spatial reference can be useful for many applications. Therefore we define *rangeOf(*{*space:tag[i]*}*, range)* as the circular space with the center at the position of the node referenced by {space:tag[i]} (this position along with a unique identifier of the node are maintained in the associated binding) and the radius equals *range*. Using

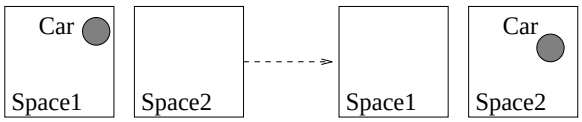


{rangeOf(Space:Temp, range):Water}

Figure 3. Relative Space Example

the same parameters we can define *northOf, southOf, eastOf,* and *westOf* as semi-circles relative to the position of the node stored in the binding associated with the given spatial reference. Figure 3 shows the use of *rangeOf* construct. To access a node that holds *Water* in the proximity (given by *range*) of the physical location associated with {*Space:Temp*}, we use {*rangeOf(Space:Temp, range):Water*}.

4.4. Space Casting



{Space1:Car} {Space2:(Space1:Car)}

Figure 4. Space Casting Example

If the user has knowledge about the mobility patterns of certain nodes, the space for a given spatial reference can be modified using space casting. The construct {*space2:(space1:tag)*} changes the geographical scope for the given spatial reference from *space1* into *space2*. Figure 4 shows how a binding was created for a given *car* in *Space1*, and how the same car is reached in *Space2* using the space casting construct.

4.5. SP Program Example

To illustrate some of the novel concepts and programming constructs introduced by SP, we present an application that monitors the state of fire sensors over a large area and activates a nearby water sprinkler once the fire is detected. Two kinds of embedded systems are deployed in the physical environment: fire sensors and water sprinklers.

```
 1 void MonitorAndExtinguish(Space area){
 2  for(i=0; i<10; i++)
 3    if ({area:Fire[i], timeout}.Fire){
 4      location = {area:Fire[i], timeout}.Location;
 5      try{
 6        newarea = rangeOf(area:Fire[i], Range);
 7        {newarea:Sprinkler, timeout}.Water = ON;
 8      }catch(Timeout e){
 9        {area:Firefighters}.FireLocation = location;
10      }
11    }
12 }
```

Figure 5. SP Program

Each sensor defines the tag *Fire*, which is *true* if the sensor observed fire, and *false* otherwise. Each sprinkler defines the tags *Sprinkler* and *Water*. A *Sprinkler* starts to pour water whenever its tag *Water* is set to *ON*. Additionally, the fire department may be alerted by setting the tag *FireLocation*, defined at the firefighters station. If it is impossible to contact one of the sprinklers, the fire department will be informed.

The source code of the SP program that implements the above task is presented in Figure 5. Reference consistency helps in obtaining the location of the sensor that detected fire when the same spatial reference instance is used(line 4). The *rangeOf* construct enables us to turn on a sprinkler located within the *Range* of the sensor that detected fire (lines 6-7). Also, if a *Timeout* exception is raised (line 8), which means that one of the sprinklers might have been destroyed, the fire department is notified (line 9).

5. Related Work

Recent projects [8, 5, 3] have presented programming models for ubiquitous/pervasive computing. SP shares some of their goals, but its main design goal is to define and implement a programming model that provides a simple way to program the physical spaces and to decouple the access to spatially distributed network resources from the networking details.

SP is closely related to Spatial Views, an iterative spatial programming model for networks of embedded systems [19]. Spatial Views provide a more restrictive, higher level programming model than SP. The Spatial Views programming model allows the specification of sets of nodes of interest, called views, together with a sequential program to be executed on each node in a view. In addition, language constructs are provided to group nodes according to their physical location,

and to specify constraints that have to be satisfied by a program execution, including resources, time, and quality of result constraints. SP can serve as a possible target language for a Spatial Views compiler, but not vice versa.

A research complementary to ours is TAG [21], which defines an SQL-like language for sensor networks. Both SP and TAG provide simple programming constructions that shield the programmer from the underlying network. The are two main differences between SP and TAG. First, SP focuses on a flexible abstraction that allows programming for uncertainty in highly dynamic networks, while TAG focuses on a set of queries executed efficiently in the network. Second, the programmer has the control over execution in SP, while TAG depends entirely on the compiler (i.e., essentially SP offers an imperative language and TAG offers a declarative language).

Spatial Programming shares the idea of using spatial information with various forms of geographical routing [17, 14, 18], but it differs from them in its main goal, transparent computing over networks distributed within the physical space.

Content-based naming has been recently presented for both Internet [4, 23, 9] and sensor networks [13]. The spatial references used in SP are similar abstractions to these content-based names, but they incorporate the spatial information and present a uniform view of space and network to applications. Another difference is that SP targets ubiquitous computing environments and is independent of the underlying implementation.

Recent work on large networks of embedded systems has focused on network protocols for wired and wireless sensor networks [11, 13], system architectures for fixed-function sensor networks [12], and energy efficient data collection for mobile sensor networks designed to support wildlife tracking [15]. Sensor networks can represent a platform for SP. Since they are deployed across large geographical regions, SP provides a viable solution to alleviate the task of writing programs for them.

6. Status and Future Work

At this time, we are in the process of implementing SP using Smart Messages (SM) [6], which represent a suitable platform for SP. The main advantage of using SMs is that they overcome the scale, heterogeneity, and connectivity issues encountered in NES by placing the intelligence in migratory execution units, while requiring only a minimal system support from nodes. Another advantage of an SM implementation is the possibility to dynamically install new services at nodes. SMs consist of code and data sections as well as a lightweight execution state. They migrate through the network, searching for nodes of interest, and execute at each node in the path. The SM execution is embodied in tasks described in terms of migration and computation phases. SMs name the nodes of interest by their properties and self-route to them using other nodes as stepping stones. Nodes in the network support SMs by providing (1) a name-based memory for inter-SM communication, synchronization and interaction with the local host, and (2) a virtual machine for SM execution.

SP constructs can be added either as language extensions or as library calls. The main component of this implementation is an SP to SM translator, which essentially will translate every access to a spatial reference into an SM migration and will generate code to handle the bindings created during execution. The binding table will be carried by SMs as mobile data. Necessary components of the implementation are geographical and content based routing that can be developed using the flexible self-routing mechanism provided by SM.

In the following, we briefly outline future ideas for improving the SP design and implementation. We plan to further investigate what other features should be added to the SP to make it even more flexible and easier to use in such complex environments as NES. We also plan to analyze the tradeoffs between various design choices that we face in implementing SP.

For networks of resource constrained devices, such as sensor networks, a more traditional implementation may yield better performance. Therefore we plan to have an SP implementation on top of Directed Diffusion [13].

The decision whether to implement the SP constructs as programming language extensions or as library calls has to be evaluated both in terms of performance and ease of programming.

There are situations when a parallel access to spatial references is possible (there are no dependencies among them) and desirable in order to increase the performance. Therefore a future additions to SP might be constructs that allow the user to specify such parallel activities (i.e., parbegin/parend, parfor).

In our current design, if an access to a bound spatial reference fails, an exception is raised and the application regains the control. Two improvements are possible: (1) a re-bind operation can be performed if the application accepts a similar node (same space and content-based name) and, (2) a transparent tracking of the mobile node that moved into another space.

References

[1] Sandia National Laboratories. http://www.sandia.gov /media/NewsRel/NR2000/avalanch.htm.

[2] Sensoria Corporation. http://www.sensoria.com.

[3] S. Adhikari, A. Paul, and U. Ramachandran. D-Stampede: Distributed Programming System for Ubiquitous Computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS).*, pages 209–216, July 2002.

[4] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The Design and Implementation of an Intentional Naming System. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 186–201, 1999.

[5] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. Challenges: An Application Model for Pervasive Computing. In *Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 266–274, August 2000.

[6] C. Borcea, D. Iyer, P. Kang, A. Saxena, and L. Iftode. Cooperative Computing for Distributed Embedded Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS).*, pages 227–236, July 2002.

[7] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin. Locating Tiny Sensors in Time and Space: A Case Study. In *Proceedings of the International Conference on Software/Hardware Codesign (ICCD 2002). Invited Paper*, October 2002.

[8] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. Systems Directions for Pervasive Computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), May 2001.*

[9] M. Gritter and D. Cheriton. An Architecture for Content Routing Support in the Internet. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2001.

[10] J. Heideman, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building Efficient Wireless Sensor Networks with Low-Level Naming. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 146–159, October 2001.

[11] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of the Fifth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 174–185, August 1999.

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 93–104, November 2000.

[13] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensors Networks. In *Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 56–67, August 2000.

[14] Jinyang Li, John Janotti, Douglas De Couto, David R. Karger and Robert Morris. A Scalable Location Service for Geographic Ad Hoc Routing. In *Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 120–130, August 2000.

[15] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2002.

[16] E. Kaplan, editor. *Understanding GPS: Principles and Applications.* Artech House, 1996.

[17] B. Karp and H. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, August 2000.

[18] Y.-B. Ko and N. H. Vaidya. Location-Aided Routing(LAR) in Mobile Ad Hoc Networks. In *Proceedings of the Fourth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 66–75, October 1998.

[19] U. Kremer, L. Iftode, J. Hom, and Y. Ni. Spatial Views: Iterative Spatial Programming for Networks of Embedded Systems. Technical Report DCS-TR-493, Rutgers University, June 2002.

[20] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, August 2001.

[21] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI). To Appear.*, December 2002.

[22] N. B. Priyantha, A. K. L. Miu, H. Balakrishnan, and S. Teller. The Cricket Compass for Context-Aware Mobile Applications. In *Proceedings of the 7th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 1–14, 2001.

[23] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active Names: Flexible Location and Transport of Wide-Area Resources. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 151–164, October 1999.

[24] M. Weiser. The computer for the twenty-first century. *Scientific American*, September 1991.