Spatial Views : Iterative Spatial Programming for Networks of Embedded Systems*

Ulrich Kremer Liviu Iftode† Jerry Hom Yang Ni

DCS-TR-493
Department of Computer Science
Rutgers University
June 2002

Abstract

Programming a large network of embedded systems (NES) is an enormous challenge since all or subsets of nodes cooperate to achieve a particular goal. Recently proposed Spatial Programming makes networking transparent to the programmer by providing a consistent spatial and content-based naming scheme for spatially distributed network resources. In Spatial Programming, NES applications appear as simple sequential programs using spatial variables. However, for many NES applications the spatial programming model can be further abstracted, by factoring out the spatial scope from resource naming and expressing the execution iteratively within that scope.

In this paper, we propose Spatial Views, a high-level abstraction for spatial programming which allows programs (iterators) to be executed over a dynamic set of nodes with similar properties and location. The execution of a spatial view iterator can be constrained by multiple conditions such as time, energy, etc., and hence, can terminate prematurely with a certain quality of result (QoR). In this paper, we also show how spatial view iterators can be translated into Smart Messages, a distributed computing programming environment for NES based on execution migration and self-routing.

1 Introduction

Programming large networks of embedded systems (NES) will be a challenge comparable to that of programming massively parallel machines. In contrast to massively parallel machine, NES are inherently unreliable and volatile. They consist of a large number of heterogeneous nodes distributed over a physical space that communicate mostly over ad-hoc networking channels. Most nodes will have limited resources and are specialized to provide particular services, for instance, temperature measurements or motion sensing. Many nodes may rely on battery power alone, and therefore have severe power constraints. Executing a program on such a "target system" will decrease the life times of its nodes, eventually leading to partial system failure. Quality of result and resource usage, in particular energy consumption, are program execution characteristics that need to be expressed at the application level.

Most applications that will execute on a NES are spatial in nature. The spatial distribution of nodes across a large physical space is a key feature of a massive network of embedded systems. You would not expect to have 10,000 embedded systems piled up in a corner of a machine room. What you would expect are sensor networks that span buildings, large facilities such as a university campus or airport, or even across state and country boundaries. As a result, location of nodes are first order programming concepts which need to be expressed and used by the application.

Spatial Programming is a novel paradigm for programming dynamic networks of embedded systems distributed in the physical space [13]. In Spatial Programming, space is a first-order programming

*This research was partially supported by NSF ITR/SI award ANI-0121416. Authors' email addresses: uli@cs.rutgers.edu, iftode@cs.umd.edu, {jhom, yangni}@cs.rutgers.edu

†Author's affiliation: Department of Computer Science, University of Maryland, College Park, MD 20742

concept that is exposed to the application and used to name network resources. Using spatial variables programmer can completely ignore the complexity of the ad-hoc networking protocols used to access the corresponding resources. A naming scheme based on spatial references can be further simplified when the application targets a set of nodes with similar properties and location. In this case, what is necessary is to define (i) the virtual network with the nodes of interest (view), and, (ii) the program to be executed on each node of the view (iterator).

In this paper, we define a simple and flexible spatial programming model for NES called Single Program Multiple Locations (SPML) based on Spatial Views and View Iterators. In SPML a single program travels across the network, gathering information or initiating computation as it visits network nodes. Since the network is volatile, nodes may become unreachable, or new nodes may become available during program execution. As a result, any programming model for such a network has to have the notion of "best effort" semantics. This means that if a node cannot be reached within a given time constraint, the node does not exist for the purpose of program execution. Applications should not end up in a state where they indefinitely wait for a node to respond, or indefinitely travel along the network to find a node with a desired characteristic or property.

The best effort semantics may lead to a program execution that produces an answer of unacceptable quality. For instance, a program that gathers temperature readings across a physical area may only succeed in reaching a single temperature sensor. The programming model should allow the specification of a range of semantically acceptable answers of a program execution, with the underlying understanding that the best possible answer should be computed. It will be the application programmers responsibility to express this range of acceptable answers, together with a set of constraints that should not be violated for any program execution. Such constraints may include resource usage (energy, CPU, memory) and soft execution deadlines.

The goal of the Spatial Views design is to identify the characteristics of a network of embedded systems that need to be exposed in a programming model in order to write expressive and efficient programs at a high level of abstraction. The key features that are included in our proposed high-level Spatial Views programming model are

- the description of a physical space, and the specification of absolute or relative node locations within this physical space,
- the specification of network topologies and network properties based on their location within the physical space, for instance, as *stable*, *mobile*, *volatile*, *dense*, or *sparse*,
- the specification of virtual networks, called views, that contain nodes of interest for the application due to their location or the services that they provide,
- the specification of resource constraints that should not be violated, such as response time, power consumption, and monetary budget constraints, and
- the specification of services (including APIs) used by the Spatial Views program.

Each of these features will be discussed in more detail in the next sections. It is the responsibility of a Spatial Views compiler to map programs from this high-level abstraction into efficient Smart Messages or Spatial Programming code.

2 Spatial Views

Our Spatial Views programming model targets application that run a spatial distributed heterogeneous network of embedded systems. The programming model assumes that each node knows its physical location, and makes this information available to an application through a predefined API (node service). Geographical routing ([5, 16, 19, 17, 14]) capabilities are assumed to be supported in the target NES. This is important for program injection, result reporting, and virtual network refinements through spatial constraints.

The physical location of a node within the network may determine whether the node is of interest to a particular application or not. In addition, a node's location often indicates the characteristics of the underlying communication network, allowing communication, process migration, and routing optimizations. As a result, the notion of space is a central feature in our programming model since a node's location within a physical space has semantic and performance implications.

Spatial Views uses the tag concept as introduced by Smart Messages [18, 7]. The availability of a specific node service is indicated through a corresponding "tag", and accesses to the services use this unique tag. In addition, tags may be created by the Spatial Views program, allowing application specific information to be computed and stored at network nodes.

Spatial Views uses a two-level spatial mapping scheme. At the first level, abstract spaces are defined and are associated with absolute physical locations. Nodes within abstract spaces may have a common network topology and characteristic which can be defined by the user. At the second level, nodes are grouped according to their logical properties and relative physical locations. This grouping is called a view. view iterators process each node in a view exactly once. They may be subject to user specified resource constraints such as response time, energy consumption, and monetary budget. A user writes a single SPML node program that defines data objects as well as views and view iterators. A program execution is initiated by injecting the program into an abstract space, restricting its activities to the physical boundaries of the abstract space. If the program computes an answer, the user needs to specify a node or set of nodes where the answer should be reported.

2.1 Abstract Spaces

The particular characteristics of the network topology can help in the selection of efficient routing strategies [8]. For instance, the massive network may have stationary nodes (for instance within a building), mobile nodes (for instance in moving cars), volatile nodes (for instance if solely battery powered), or regions of dense or sparse node population. Some of the network characteristics may be transient, but some may not. Hints about the network topology and characteristic can help the compiler or underlying runtime system to select efficient routing or migration strategies. For instance, if the program knows that it is within a dense and stable region of the network it may choose a different routing strategy than in a volatile, sparse network region.

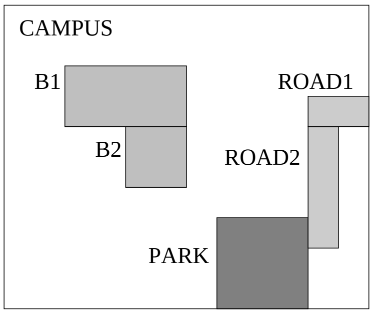
For simplicity of discussion, we describe SPML programming using only two-dimensional abstract spaces of rectangular shapes. Reference points are used to align different spaces relative to each other, which allows scaling and positioning. The textual order in which spaces are specified allows a hierarchical space definition, with later specifications overwriting former ones. Every abstract space can be annotated with attributes that reflect the nature of the network within the space. Once all abstract spaces have been defined and aligned, the resulting collection of spaces is mapped to a corresponding physical space. The precision of the physical location coordinates may depend on the available information, with resolutions in the order of millimeters to meters. Figure 1 shows the outline of a physical space of a campus consisting of buildings, a parking lot, and an access road, together with its corresponding abstract space specifications.

We expect that the same abstract space definition is used by many different SPML node programs that perform computations and information acquisition tasks within the corresponding physical space. An SPML node program is injected into a location within the abstract space, allowing the compiler and runtime system to take advantage of the specified properties of the space in order to optimize tasks such as routing and process migration. It is important to note that our programming model does not require the specification of an abstract space. However, without an abstract space, applications that require physical information, for instance the computation of the outside temperature on campus, cannot be easily expressed. In addition, more generic routing and process migration schemes may have to be used, leading to a degradation in performance.

2.2 Views and View Iterators

A view describes a set of network nodes with a given property, i.e., specifies a virtual network of embedded systems. A view can specify all nodes with a particular tag, for example, all nodes that have a "temp" tag indicating that they have a temperature sensor. In addition, a description of the physical locations of all nodes in the view relative to a reference location may be specified. The combination of property constraints and relative spatial constraints define a view. Our current model supports three types of relative spatial constraints,

- **Grid** constraint: Divides the virtual network into equally sized sections. The view contains only a single node for each section.
- **Range** constraint: Only nodes within a physical range of the reference node are part of the view.
- **Cone** constraint: Only node within a directional cone with the reference node as the source are included in the view.




```
// PHYSICAL SPACES
space B1(2,4) // 2 units in x, 4 in y dim.
space B2(2,2) // 2 units in x, 2 in y dim.
space CAMPUS(10,12)
space ROAD1(1,2)
space ROAD2(4,1)
space PARK(3,3)

// ALIGNMENTS (RELATIVE LOCATIONS)
align B1(0,0) with CAMPUS(2,2)
align B2(0,0) with CAMPUS(4,4)
align ROAD1(0,0) with CAMPUS(3.10)
align ROAD2(0,0) with CAMPUS(4,10)
align PARK(0.0) with CAMPUS(7,7)

// PHYSICAL LOCATION ASSIGNMENTS
map CAMPUS(0,0) to (...)
  . . .
map CAMPUS(10,12) to (...)

// SPACE GROUPINGS
define BUILDINGS = B1+B2
define ROADS = ROAD1+ROAD2
define OPENSPACE = CAMPUS-(BUILDINGS+ROADS+PARK)

// ATTRIBUTE ASSIGNMENTS
attribute stable, dense for BUILDINGS
attribute mobile, sparse for ROADS
attribute volatile, dense for PARK
attribute volatile, sparse for CAMPUS
```

Figure 1: Abstract space example: A campus with buildings, a parking lot, and an access road.

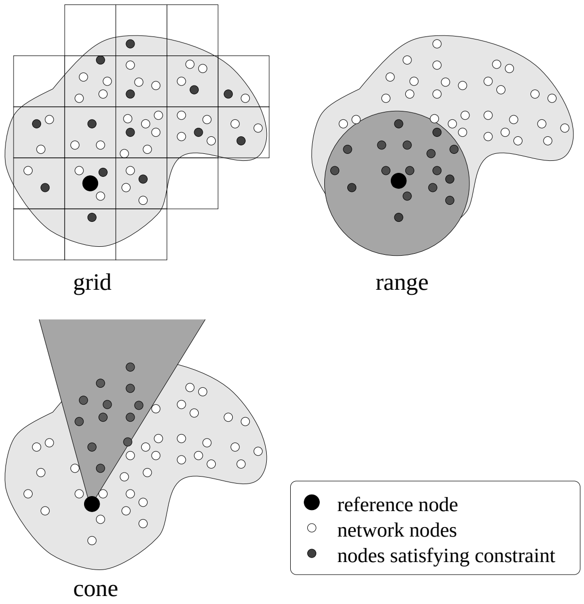


Figure 2: Relative spatial constraints: Grid, range, and cone.

Figure 2 shows different relative spatial constraints and their semantics. Grid constraints can be used to express quality of result (QoR) tradeoffs of an application. For instance, a program that collects temperature readings across a space may not want to visit every node that has a temperature sensor. Instead, a representative subset should be visited, in this case a subset of nodes evenly distributed across the physical space. Limiting the number of visited nodes is not sufficient since clustering may occur. For example, only visiting sensors on the same floor of a building in order to compute the average temperature across the entire building is not acceptable. A grid constraint avoids such a clustering. Range constraints allow the expression of locality relations between network nodes of interest. For instance, an application that tries to locate the origin of a recorded sound may first want to query other nearby microphones to confirm or locate the source of the sound. Cone constraints are useful in applications that track objects or events. In the microphone example, the search for the source of the sound may be direction oriented.

Each view is associated with at least one iterator that specifies the program to be executed on each node of the view. The iterator may be subjected to constraints, for instance resource constraints (e.g.: do not use more than 10 KJoules of Energy) time constraints (e.g.: terminate within less than 10 seconds), or QoR constraints (e.g.: visit no more than five nodes). In a "best-effort" semantics, there are in general no firm constraints. The system tries to do its best to respect the constraints, i.e., the system only supports soft constraints. Any statistics that the user needs to verify the soft deadline needs to be gathered explicitly by the programmer through program data structures. However, the view iterator construct keeps track of the number of nodes that have been visited so far in the current iteration. A view has a scope with a corresponding lifetime. View iterators may themselves contain views and can be nested.

2.3 Tag Semantics and Services

The specification of tags and their semantics is optional in our Spatial Views programming model. All name resolution will occur at program execution time, i.e., is dynamic in nature. In general, no semantic support is provided, i.e., it is assumed that there is an implicit understanding between the application and the NES what particular tags mean, and what particular services do. If a name of a particular service cannot be resolved, a runtime error will occur.

The specification of services and their corresponding APIs within a Spatial Views program allows "standardized tags" with a fixed semantics. Typically, nodes will have to be certified according to this semantics before they are deployed as part of a NES. A certification checking process is used during dynamic name resolution. This will ensure that, for instance, if a node provides a camera tag, a camera

```
define PROG1 (int average){
 constraint TEMP-VISIT (at most 50 nodes);
 view Temps (temp, TEMP-VISIT);//temp is a tag
 viewiterator TempsIter (Temps);
 average = 0;
 do {
  node currentT = TempsIter.current.temp;
  average = average + currentT.getTemperature;
 } while (TempsIter++);
 average = average / TempsIter.numNodesVisited;
}

define PROG2 (set X){
 constraint TEMP-VISIT (at most 10 nodes);
 constraint  GRID (2d-grid (2m, 2m));
 view Temps (temp, TEMP-VISIT, GRID);//temp is a tag
 viewiterator TempsIter (Temps);
 do {
  node currentT = TempsIter.current.temp;
  if (currentT.getTemperature > 70) {
   // temperature is measured in Celsius
   constraint CAMERA-RANGE (currentT.myLocation, 10m);
   constraint CAMERA-VISIT (at most 3 nodes);
   constraint CAMERA-TIME (at most 2 secs);
   view Cameras (camera, //camera is a tag
                 CAMERA-RANGE, CAMERA-VISIT);
   viewIterator CamerasIter(Cameras) (CAMERA-TIME);
   int confidence = 0;
   do {
    node currentC = CamerasIter.current.camera;
    if (currentC.mostlyRed(currentC.getPicture))
     confidence++;
   } while (CamerasIter++);
   if (CamerasIter.numNodesVisited < 1)
    X.insert(currentT.myLocation, ``no camera'');
   else
    X.insert(currentT.myLocation, confidence);
  }
 } while (TempsIter++);
}
```

Figure 3: Example SPML node programs: PROG1 computes the average temperature; PROG2 detects fires using temperature sensors and cameras.

```
prog SV1 {
  int averageTemperature;
  inject PROG1(averageTemperature) into OPENSPACE;
  averageTemperature.print();
}

prog SV2 {
  constraint TIME (at most 30 secs);
  set SensorFireReadings;
  inject PROG2(SensorFireReadings) into B2
    with constraints TIME report at B1(1,1);
  SensorFireReadings.print();
}
```

Figure 4: Example Spatial Views programs: SV1 computes the average outside temperature on campus. SV2 detects potential fires within building B2, and reports result at location in building B1 within 30 seconds. Both programs use abstract space definitions introduces in Figure 1.

is present at the node and a picture may be accessed through a known `picture` tag API. A discussion of possible strategies to enforce tag semantics and services is beyond the scope of this paper.

3 Program Examples

Figure 3 shows two example SPML node programs similar to those described in [18, 7, 13]. A complete Spatial Views program consists of at least one SPML node program and an `inject` statement that may either directly specify the physical location at which the SPML program is activated, or a location within an abstract space where the activation will occur. In the latter case the compiler and runtime system will translate the abstract space location into a physical location. Abstract spaces were discussed in Section 2.1. Figure 4 shows different Spatial Views programs based on the SPML nodes programs `PROG1` and `PROG2`. In addition to a list of constraints, the `inject` statement also may contain a `report` clause which allows the specification of the location of a node at which the results should be delivered. If the `inject` clause is omitted, the results are returned to the node that initiated the injection process.

The SPML node program `PROG1` shown in Figure 3 computes the average temperature based on at most 50 nodes with temperature sensors. Nodes with temperature sensors are identified by the `‘‘temp’’` tag. The average temperature reading of the visited sensor nodes is returned. SPML node programs `PROG2` visits at most 10 temperature sensors in order to detect the presence of fires. Once a sensor reports a high temperature, at most three nearby cameras are queried to confirm that there is a fire close to the location of the sensor. This is done by defining a view that contains at most three cameras. The presence of a fire is indicated by an image that is mostly red. Visiting the cameras that may have the temperature sensor in its range should not take longer than two seconds, as required by the time constraint specified for the camera iterator. The results are recorded in a `set` data structure, which is a formal parameter of the SPML node program. Our Spatial Views programming model uses a *call by value/result* parameter passing semantics [3]. The additional spatial constraint `GRID` specifies the granularity at which nodes with temperature sensors should be visited. The desired resolution is two by two meters. This is a quality of result (QoR) constraint in order to achieve an equal spacing of the visited temperature sensors. Although not shown in the example, constraints may be parameters to SPML node programs.

Translation into Smart Messages

The Spatial Views program has to be translated into executable code on the target network of embedded systems. In this section, we present the translation of SPML in Smart Messages (SM) [18, 7]. A Smart Message is a distributed application that executes in a NES by migrating to the nodes of interest. SMs name the nodes of interest by their properties(content) and self-route to them using other nodes as stepping stones. Nodes in NES cooperate to support SM by providing: (1) a name-based memory(Tag Space) and (2) an architectural independent execution environment (Virtual Machine). SM provides a small set of programming primitives that is expressive and flexible enough to implement a wide range of

```
bool inOPENSPACE(location) {
  return (location >= ( , ) &&
          location <= ( , ) && ...)}

mobile location reportingLocation;
mobile int numNodesVisited = 0;
mobile int average = 0;

reportingLocation = readTag(myLocation);
do {
 // volatile and sparse network topology
 if (space_migrate_volatile_sparse(//SM library call
          temp,inOPENSPACE(),timeout)) {
  average += readTag(temp);
  numNodesVisited++;
 } else
  break;
} while (numNodesVisited < 50);
average = average / numNodesVisited;
geo_migrate(reportingLocation);// SM library call
print(average);
```

Figure 5: Translation of Spatial Views program SV1 from Figure 4 into SM code.

distributed applications [18, 8, 7]. The SM runtime library contains routines that implement different self-routing and migration strategies, and functions to modify admission policies on single nodes in the network. Figure 5 shows a possible translation of the Spatial Views program `SV1` of Figure 4 into SM code. The compiler generates the `inOPENSPACE` predicate based on the abstract space definition. A SM migration library call is inserted that takes advantage of the particular properties of the target network. Geographical routing is used (`geo_migrate`) to return the result to the node that initiated the program execution. If less than 50 temperature sensors can be reached, the migration routine will time out, and return the partial result.

4 Compiler Optimizations

The Spatial Views model allows several compiler optimizations. For instance, although the program model has the underlying notion that a single program "walks" across the network, an actual implementation may use several program instances to improve the response time of an application, thereby satisfying a possible response time constraint. Other optimizations are based on the knowledge of physical space and the services and network characteristics available in the space. For instance, instead of using a general routing strategy, specialized routing can be used in stable networks.

Process migration is only performed at iteration boundaries. The current language does not allow views to be specified within procedures. This avoids the problem of moving the runtime stack across the network. We are currently investigating less restrictive view usages and their impact on optimized program performance.

5 Related Work

Recent projects [9, 6, 1] have presented programming models for ubiquitous/pervasive computing. Spatial Programming shares some of their goals, but its main design goal is 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 network.

Spatial Programming shares the idea of using spatial information with various forms of geographical routing [5, 16, 19, 17, 14], 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 [2, 4, 10] and sensor networks [15].

Recent work on large networks of embedded systems has focused on network protocols for wired and wireless sensor networks [11, 15] and system architectures for fixed-function sensor networks [12]. Sensor networks are a main target for Spatial Views , which can be implemented using either Smart Messages , Spatial Programming , or a system built on top of the above protocols.

6 Summary and Conclusion

We believe that programming massive networks of embedded systems is one of the main future challenges. New programming models need to be developed to deal with the complexity and volatile nature of such networks which only support a "best effort" semantics. This paper discusses the design of Spatial Views , a new high-level spatial programming abstraction for networks of embedded systems. Spatial Programming assumes a target network where each node knows about its physical location, allowing geographical addressing and routing strategies. The programmer is able to specify abstract spaces together with their physical locations and attributes. The attributes are used by the compilation and runtime system to improve program performance. An Single Program Multiple Location (SPML) program is injected into the abstract space, or directly into a physical location. Spatial Views provides network views that allow the specification of a subset of network nodes with particular logical and spatial properties. An iterator construct provides the means to visit and process each node in a view. Iterators themselves may be subject to resource, time, and quality of result (QoR) constraints. A compiler can translate a Spatial Views program into a low-level program representation with explicit process migration and routing. Using a simple Spatial Views program as an example, the translation process into an efficient Smart Messages program representation is illustrated. A prototype implementation of a Spatial Views compiler is currently underway.

References

[1] Sameer Adhikari, Arnab Paul, and Umakishore Ramachandran. D-stampede: Distributed programming system for ubiquitous computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS), July 2002. To Appear.*

[2] 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*, pages 186–201, 1999.

[3] A. V. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, second edition, 1986.

[4] Thomas Anderson Amin Vahdat, Michael Dahlin and Amit Aggarwal. Active names: Flexible location and transport of wide-area resources. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS)*, October 1999.

[5] Karp B. and Kung H.T. Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 243–254, 2000.

[6] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Deborra Zukowski. Challenges: an application model for pervasive computing. In *Proceedings of the 6th ACM MOBICOM, August 2000.*

[7] C. Borcea, D. Iyer, P. Kang, A. Saxena, and Iftode. Cooperative computing for distributed embedded systems. In *International Conference on Distributed Computing Systems (ICDCS2002)*, July 2002.

[8] C. Borcea, A. Saxena, D. Iyer, P. Kang, R. Sarker, and L. Iftode. Self-routing in networks of embedded systems using Smart Messages. Technical Report DCS-TR-477, Department of Computer Science, Rutgers University, March 2002. submitted for publication.

[9] Robert Grimm, Janet Davis, Ben Hendrickson, Eric Lemar, Adam MacBeth, Steven Swanson, Tom Anderson, Brian Bershad, Gaetano Borriello, Steven Gribble, and David Wetherall. Systems directions for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), May 2001.*

[10] Mark Gritter and David Cheriton. An architecture for content routing support in the internet. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01)*, March 2001.

[11] W. Rabiner 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*, pages 174–185, 1999.

[12] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[13] L. Iftode, C. Borcea, D. Iyer, P. Kang, U. Kremer, and A. Saxena. Spatial programming with Smart Messages for networks of embedded systems. Technical Report DCS-TR-490, Department of Computer Science, Rutgers University, May 2002.

[14] T. Imielinski and J. C. Navas. Geographic addressing, routing, and resource discovery with global positioning system. *Communication of the ACM*, 1999.

[15] 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*, 2000.

[16] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, Boston, Massachusetts, August 2000.

[17] D. Niculescu and B. Nath. Trajectory based forwarding and its applications. Technical Report DCS-TR-488, Department of Computer Science, Rutgers University, May 2002.

[18] P. Stanley-Marbell, C. Borcea, K. Nagaraja, and L. Iftode. Smart Messages: A system architecture for large networks of embedded systems. Technical Report DCS-TR-430, Department of Computer Science, Rutgers University, January 2001. Position summary at Hot OS-VIII, May 2001.

[19] Young-Bae Ko and Nitin H. Vaidya. Location-Aided Routing(LAR) in Mobile Ad Hoc Networks. In *ACM/IEEE MobiCom*, pages 66–75, October 1998.