Spatial Views: Space-Aware Programming for Networks of Embedded Systems *

Yang Ni, Ulrich Kremer, and Liviu Iftode

Department of Computer Science
Rutgers University
Piscataway, NJ 08854
{yangni, uli, iftode}@cs.rutgers.edu

Abstract. Networks of embedded systems, in the form of cell phones, PDAs, wearable computers, and sensors connected through wireless networking technology, are emerging as an important computing platform. The ubiquitous nature of such a platform promises exciting applications. This paper presents a new programming model for a network of embedded systems, called Spatial Views, targeting its dynamic, space-sensitive and resource-restrained characteristics. The core of the proposed model is iterative programming over a dynamic collection of nodes identified by the physical spaces they are in and the services they provide. Hidden in the iteration is execution migration as the main collaboration paradigm, constrained by user specified limits on resource usage such as response time and energy consumption. A Spatial Views prototype has been implemented, and first results are reported.

1 Introduction

The possibility of building massive networks of embedded systems (NES) has become a reality. For instance, cell phones, PDA's, and other gadgets carried by passengers on a train can form an ad hoc network through wireless connection. In addition to those volatile and dynamic nodes, the network may contain fixed nodes installed on the train, for instance public displays, keyboards, sensors, or Internet connections. Similar networks can be established across buildings, airports or even on highways among car-mounted computers. Any device with a processor, some memory and a network connection, probably integrated on a single chip, can join such a network. The application of such a network is limited only by our imagination, if we had the right programming models and abstractions.

Existing programming models do not address key issues for applications that will run on a network of embedded systems.

Physical Locations: An application has a physical target space region, i.e., a space of interest in which it executes. The semantics of a program executing outside its target space in not defined. For instance, it makes a difference if an application collects temperature reading

* This research was partially supported by NSF ITR/SI award ANI-0121416.

within a building or outside a building, and whether all or only a subset of temperature sensors are to be polled. A motion sensor reading may trigger the activation of other sensors but only of those which are in the spatial proximity of the motion sensor. A programmer must be able to specify physical spaces and location constraints over these spaces.

Volatile and Dynamic Networks: Nodes may join and leave at any time, because of movements or failure. Portable devices or sensors, carried by a person or an animal[16], may go out the space of interest while they are moving with the carriers. Battery powered small devices may go out of power at any point. A node available at time t can not be assumed available at time $t + \Delta t$ or time $t - \Delta t$, where Δt can be very small relative to the application execution time.

Resource Constraints: Resources like energy and execution time are limited in a network of embedded systems, due to the hardware form factor and application characteristics. Graceful degradation of quality of results is necessary in such an environment. Instead of draining the battery of the sensors, you might want to limit the total energy used by a program and accept a slightly worse answer. Or you may limit the response time of a query for traffic information 10 miles ahead on the highway, so you will have enough time to choose a detour after getting the answer. In those cases, energy-wasting or late answers are not better or even worse than no answer. Programmers should be able to specify the amount of resource used during a program execution, so trade-offs between quality of results and resource usage can be made.

In this paper, we introduce Spatial Views, a novel programming model targeting networks of embedded systems. Spaces, services and resource constraints are explicit programming elements in Spatial Views. Spaces and services are combined to define dynamic collections of interesting nodes in a network, called *Spatial Views*. *Iterators* and *Selectors* specify code to execute in a view under a specified time constraint, and possibly additional user specified resource constraints. These high level program constructs are based on a migratory execution model, guided by the space and service of interest. However, Spatial Views does not exclude an implementation using other communication mechanisms, such as remote procedure calls, message passing or even socket programming, for performance or energy efficiencies.

Network or node failures are transparent to the programming model. However, there is no guarantee that the execution of an application will be able to complete successfully. Our proposed model is not fault tolerant, but allows answers of different qualities. In contrast, in a traditional programming model for a stable target system, any answer is considered to have perfect quality. In our programming model, it is the responsibility of the programmer to assess the quality of an answer. For example, if a user wants the average temperature calculated from readings of at least ten network nodes, he or she should report the average temperature together with the number of actually visited nodes to assess the quality of the answer. A best-effort compiler and run-time system will try to visit as many nodes as possible, as long as no user defined constraint is

violated, assuming that visiting more nodes will produce a potentially better answer. A target space and a time constraint have to be specified for each program to confine its execution, including node discovery, to a space×time interval.
Security and privacy issues are also important in a NES but are not currently part of our programming model. The same application will run on a secure network as well as on a insecure network. We assume that security-sensitive hosts will implement authentication and protection policies at a lower level than the programming model.
Smart Messages[3] and Spatial Programming[12] are possible implementation platforms for our proposed Spatial Views programming model. A programming environment for execution migration that includes protection and encryption for Smart Messages is currently under investigation[26], which could be used as an secure infrastructure to implement our programming model. However, in this paper we describe an implementation of Spatial Views on top of Sun's K Virtual Machine (KVM) independent of Smart Messages.
In the rest of this paper, we will present a survey of related works (section 2), the programming model (section 3), a discussion of the implementation of a prototype system (section 4) and experimental results (section 5).

2 Related Work

Our work is correlated to recent work on sensor networks[11, 19, 5, 15, 18] in that they all target ad hoc networks of wireless devices with limited resources. However, we broaden the spectrum of network nodes to include more computing powerful devices like PDA's, cell phones and even workstations or servers in addition to sensors. An ad hoc network including more powerful devices with versatile IO capabilities would enable more interesting applications. Our vision is that fixed nodes ,including sensors, displays, speakers, workstations and servers, work as an infrastructure in the environment. PDA's, cell phones, intelligent watches or other gadgets play the role as personal terminals. These "personal terminals" not only interact with the environments, but also interact among themselves through wireless ad hoc network. In one word, our work targets different applications and assumes different hardware than sensor networks.
TinyOS[11] and nesC[5] provide a component-based event-driven programming environment for Motes, small wireless devices that have processors of a couple of MHz, about 4KB RAM and 10Kbps wireless communication capability. TinyOS and nesC use Active Messages as a communication paradigm. Active Messages has a similar flavor to execution migration of Spatial Views, but use non-migrating handlers instead of migrating code. Mate[19] is a tiny virtual machine built over TinyOS for sensor networks. It allows capsules in bytecode to forward themselves through a network with a single instruction, which bears the resemblance to execution migration in Spatial Views. Self forwarding enables on-line software upgrading, which is important in large-scale sensor networks.
Our work can also be considered a research effort on ubiquitous computing[25] (also called pervasive computing[23] or context-aware computing

in recent literature). In this broad research area, we are only investigating certain relevant issues from the perspective of programming language and compiler designing. Our focus is on proper programming abstractions for elements such as services, locations and resource limitations, which are very important in ubiquitous computing.
Next, we are going to discuss related work about services and locations. We will also discuss related work about execution migration, which is used in the reference implementation of our programming model.

2.1 Service Discovery

Service discovery is a research area with a long history. Service is usually specified either as an interface (like in Jini,)[24] or as a tuple of attribute-and-value pairs (like in INS.)[2] Attribute-and-value pairs describe a hierarchical service space by adding new attributes and corresponding values in a describing tuple. The same goal can be achieved through interface sub-typing.
In this work we assume that service discovery is a basic function provided by the operating system. Spatial Views programming model specify services as interfaces. Applications and service providing nodes agree on the semantics of the methods of the services.

2.2 Location Technology and Space Modeling

Space-awareness is crucial for NES computing. Spatial Views tries to provide a general space model in a high-level programming language, by uncoupling space knowledge from specific location technology.
GPS[6] is the most developed positioning technology. It has a history of more than 30 years. It is all-weather world-wide available with very high accuracy regarding its scale, 16 meters for absolute positions and 1 meter for relative positions. With the radio signals from 4 satellites out of 24 in the system, a user with a receiver can calculate the distances from each of them and thus solve equations to get his or her longitude, latitude, altitude and, as a side-effect, the time to an accuracy of 100ns. In spite of its many advantages, GPS is only available outdoor and its accuracy is still not satisfactory for many mobile computing applications. In recent years, more accurate in-door positioning technologies have been developed by the mobile computing community. Active Badges and Bats[1, 10, 9] are tracking systems as accurate as to a few centimeters. Each object is attached a radio frequency tag, called Active Badge or Bat, whose signal can be detected by a grid consisting of hundreds of receivers installed on the ceiling of a building. Receiver readings are stored in a central machine and analyzed to track the object associated with a specific tag.
Although accurate, Active Badges and Bats are costly and hard to deploy. User privacy is not protected since everyone with a tag exposes his/her position by sending out radio signals. The central machine in charge of analyzing each user's position causes scalability problem and represents a single point of failure.

Cricket[21, 20] tries to address those issues by using a distributed and passive architecture similar to GPS. Beacons are installed in every space of interest in a building, like offices, meeting rooms and hallways. A beacon emits radio and ultrasound signals simultaneously. A user receiver listens. After a radio signal is received from a beacon, the receiver times the delay until the ultrasound signal is received from the same beacon. The distance from the beacon can be calculated with the delay. Among all the beacons whose distances are calculated, the user select the location of the nearest beacon as his own. Beyond addressing the cost, scalability and privacy problems, Cricket provides an accuracy to a few meters.
Embedded in the above location technologies are two categories of space models, which provides an abstract representation of locations and spaces. Cricket uses a symbolic model, while GPS uses a geometric model. In a symbolic model, a space is represented using a human readable symbolic name, usually hierarchical, such as "room301.core.busch.rutgers". In a geometric model, a space is represented as a set of locations (2- or 3-tuples). The set is usually described analytically, by describing the shape of the space with necessary parameters. For efficiency, instead of analytical representation, a quadtree can be used to represent the maximal cover of a space[10].

2.3 Migratory Execution

Spatial Views is part of the Smart Messages project[4, 3]. The goal of Spatial Views is to build a high-level space-aware programming language over Smart Messages. We had a simple implementation of the migratory execution feature of Smart Messages for fast development and evaluation of Spatial Views.
Migratory execution has been extensively studied in the literature, especially in the context of mobile agents[8, 7]. Mobile agents are programs autonomously migrating from one node to another. So, considering their migratory execution feature, Spatial Views and Smart Messages fall into the category of mobile agent programming models. However, Spatial Views only supports implicit transparent migration hidden in its iteration operation, and names a node based on the services that it provides. D'Agents[8], once known as Agent Tcl[7], allow programmers to write mobile agents in Tcl, Scheme and Java. D'Agents provide strong migration operations, named "jump", using IP addresses or domain names to address hosts. Transport is assumed a task of the underlying TCP/IP network by D'Agents. Spatial Views/Smart Messages is different from D'Agents in terms of the design goal. We are designing a programming tool and infrastructure for cooperative computing on networks of embedded systems. The major network connection is assumed wireless. In Spatial Views/Smart Messages, we take a content naming approach, and a migrating program is responsible for its own routing. All those features are in an extension to KVM instead of a standard JVM. Experimental results showed that our simple implementation of execution migration for Spatial Views has similar performance to the performance of D'Agents. We expect that using the Smart Messages implementation currently un-

der development[3] will further improve the performance of a Spatial Views program.

3 Programming Model

To program a network of embedded system in Spatial Views, a programmer specifies the nodes in which he or she is interested based on the properties of the nodes. Then he or she specifies the task to be executed on those nodes. The properties used to identify interesting nodes include the services of the nodes and their locations.
A program starts running on one node. Whenever it needs some services which the current node does not provide, it discovers another node that does, and migrates there to continue its execution.
Spatial Views provides necessary programming abstractions and constructs for this novel programming model. Node discovery, ad hoc network routing, and execution migration are transparently implemented by the compiler, runtime system, and the operating system. A programmer is freed from dealing directly with the dynamic network. Figure 1 shows an example of Spatial Views program. We will walk through this example in Section 3.3.

3.1 Services and Virtual Nodes

NES computing is cooperative computing[4]. Nodes participate in a common computing task by providing some service and using services provided by other nodes at the same time. A service is described or named with an *interface* in Spatial Views. Nodes provide services which are discovered at run-time, and are provided as objects implementing certain interfaces. In our programming model, discovery is assumed a basic function provided by the underlying middleware or OS. But we provide a simple discovery implementation based on the "random walk" technique in Section 4. The discovery procedure looks for nodes hosting classes implementing the interface. When such a node is found, an object of the class is created. The program is then able to use the service through the object. The discovery may be confined to certain physical space as we will discuss in Section 3.2.
The basic programming abstraction in Spatial Views is a *virtual node*, which is denoted as a pair (service, location), representing a physical node providing the service and locating in the location. Concrete physical nodes with IP addresses or MAC addresses are replaced by virtual nodes. Depending on how many services it provides, a single physical node may be represented by multiple virtual nodes. More interestingly, if a physical node is mobile, it may be used as different virtual nodes at different points during the application execution. Uniquely identifying a particular physical node is not supported in Spatial Views. In case that an application needs to do so, the programmer can use some application-specific mechanism, for example, using MAC addresses.

3.2 Spatial Views, Iterators and Selectors

A *spatial view* is a dynamic group consisting of virtual nodes that provide a common service and locate in a common space. Here a space is a set of locations, which can be a room, a floor, or a parking lot. Iterators and selectors describe actions to be performed over the nodes in a view. The instructions specified in the body of an iterator are executed on all or as many nodes as possible of the view. In contrast, the body of a selector is executed on only one node in the view if the view is not empty.
The most important characteristics of a spatial view is its dynamic nature. It is a changing set of virtual nodes. A physical node may move out, or run out of power. So a virtual node may just disappear at an arbitrary point. On the other hand, new nodes may join at any time. For this reason, two consecutive invocations of the same iterator over the same view may lead to different results.
A spatial view is defined as follows:

SpatialViewDefinition →
`SpatialView` *SV_id* `= new SpatialView(` *Service* `,` $Space_{opt}$ `)`

where *Service* is the name of an interface and *Space* is the space of interest. If the space is omitted, any node providing the interesting service is included in the view no matter where it is.
A spatial view is accessed through an iterator or selector.

Iterator →
`foreach` *node_id* `in` *SV_id* `do` *TimeConstraint* $ConstraintList_{opt}$
Statement
Selector →
`forany` *node_id* `in` *SV_id* `do` *TimeConstraint* $ConstraintList_{opt}$
Statement
TimeConstraint →
`within` *NumberOfMilliseconds*

ConstraintList gives a list of energy, monetary or other constraints applied to an iterator or a selector. *TimeConstraint* gives a time constraint, which is mandatory. At this point, only time constraints are supported. A time constraint demands an iterator or selector finish in *NumberOfMilliseconds*. Time constraints are enforced following a best-effort semantics with the iteration body as the minimal atomic unit of constraint control. This means an iteration will never be partially executed even when a time constraint is violated. A time constraint in Spatial Views is a soft deadline, and is a time budget rather than a real-time deadline. In other words, the time constraint does not ensure that a program terminates successfully within the deadline, but ensures no further execution after the budget is exhausted.

```
 1: // Import space definitions.
 2: import SpaceDefinition.Rutgers.*;
 3:
 4: public class SVExample {
 5:   public static void main(String args[]) {
 6:     // Define a Spatial View containing cameras on the 3rd floor
 7:     // of the CoRE building
 8:     SpatialView cameraView = new SpatialView("Camera",
 9:                                              BuschCampus.CoRE.3rdFloor);
10:
11:     // Iterate over camera view in 30 seconds
12:     foreach camera in cameraView do within 30000 {
13:       Picture pic = camera.getPicture();
14:       Rectangle redRegion = pic.findRegionInColor(Color.Red);
15:
16:       if (redRegion != null) {
17:         Rectangle face;
18:
19:         // Define a Spatial View of nodes providing face
20:         // detection service. The default space is anywhere.
21:         SpatialView detectorView = new SpatialView("FaceDetector");
22:
23:         // select a detector and finish face detection in 10 seconds
24:         forany detector in detectorView do within 10000
25:           face = detector.detectFaceInPicture(pic);
26:
27:           Location loc = camera.getLocation();
28:
29:           // Check if the the red region is close to the
30:           // face so that we can think it is a person in red
31:           if (face != null && face.isCloseTo(redRegion))
32:             System.out.println("A person in red is found at " + loc);
33:           else
34:             System.out.println("Something red is found at " + loc);
35:       }
36:     }
37:   }
38: }
```

Fig. 1. Spatial Views example application of locating a person in red

3.3 Example

The example shown in Figure 1 illustrates a Spatial Views application that executes on a network that contains nodes with cameras and nodes that provide image processing services such as human face detection[22]. The program tries to find a person with a red shirt or sweater on the third floor of a building. An answer is expected back within 30 seconds (soft deadline). A time limit is necessary because the computed answer may become "stale" if returned too late (the missing person may have left the building at the time the successful search result is reported).
Static physical spaces such as buildings and floors within buildings may be defined as part of a Spatial Views space library. In the example, we assume that the package "SpaceDefinition.Rutgers.*" contains such definitions for the Rutgers University campuses.
Line 8 defines a spatial view of cameras on the third floor of a building named CoRE (a building at Rutgers University.) Line 11-36 define the task to be performed on the cameras in the spatial view defined in line 8. It is an iterator, so the task will be executed on each camera discovered within the time constraint, 30 seconds as defined in line 12. In line 13, a picture is taken using a just discovered camera. Line 14 tries to find a region in the picture that is mostly red. If such a red region is found, another spatial view consisting of face detectors is defined (line 21.) Line 24 and 25 use a face detector in the view defined in line 21 to find a face in the picture. (Because it is a selector, line 24 and 25 finishes as soon as one face detector is discovered.) If the face is close to the red region in the picture, the program concludes it is a person in a red shirt, and reports the location of the camera where the picture is taken.

4 Implementation

The implementation itself is not the major contribution of this paper. The programming model is. The purpose of this implementation is to justify the programming model, and to provide an opportunity to study the abstractions and constructs proposed in the model. It is part of our on-going work to make this implementation faster, scalable, secure and economic acceptable. However, the current implementation has shown the feasibility of our programming model.
Our prototype is an extension to Java 2 Platform, Micro Edition (J2ME)[13]. Figure 2 shows the basic structure of the Spatial Views compilation system. We are currently investigating optimization passes that improve the chances of a successful program execution in a highly volatile target network. The compiled bytecode runs on a network, each node of which has a Spatial Views virtual machine and a Spatial Views runtime library. Figure 3 shows the architecture of a single node.
We build the Spatial Views compiler, virtual machine and runtime library based on Sun's J2ME technology[13]. J2ME is a Java runtime environment targeting extremely tiny commodities. KVM[14] is a key part of J2ME. It is a virtual machine designed for small-memory, limited-resource and networked devices like cell phones, which typically contain 16- or 32-bit processors and a minimum memory of about 128 kilobytes.

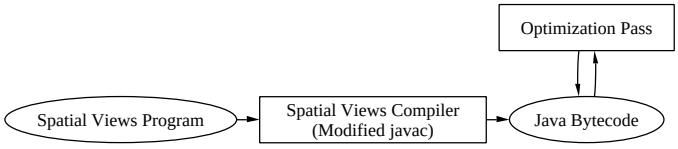


Fig. 2. Compilation of Spatial Views Programs

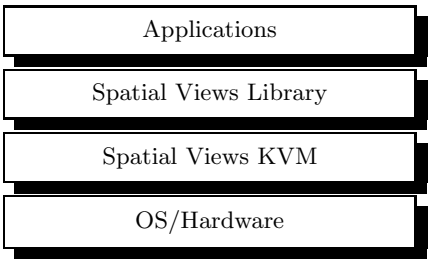


Fig. 3. Architecture of a Node

We modified `javac` in Java 2 SDK 1.3.1 to support the new Spatial Views language structures, including the `foreach` and `forany` statement and space definition statements. We modified the KVM 1.0.3 to support transparent process migration. And we extended CLDC 1.0.3 with new system classes to support Spatial Views language features. We ported our implementation to x86 and ARM architectures, running Linux 2.4.x.

4.1 Spatial Views Iteration and Selection

At the beginning of an iteration, a new thread is created to discover interesting nodes and to migrate the process there. We call the new thread *Bus Thread*. The Bus Thread implements a certain discovery/routing algorithm and respects the user-specified constraints.
The Bus Thread migrates from one interesting node to another. An interesting node is a node that provides the service locates in the space specified in the spatial view definition. On such a node, the Bus Thread blocks and switches to the user task thread the code of which is specified in the iteration body. When an iteration step finishes, the user task thread blocks and switches back to the Bus Thread. The Bus Thread finishes when no more interesting nodes can be found or the time budget is used out. In the case of selectors, the Bus Thread finishes after the first interesting nodes is found. When the Bus Thread finishes, the corresponding spatial views iteration ends. The Bus Thread is like a bus carrying passengers (user task threads in our case), running across a region and stopping at certain interesting places, hence the name.
This implementation with a Bus Thread provides a simple framework to iterate a spatial view as a dynamic set of interesting nodes. Node

discovery is transparent to the programmer and performed by the underlying middleware or by the OS using existing or customized discovery and routing algorithms.
Notice that such a framework does not limit the search algorithm a program uses to discover an interesting node. In the current implementation. We use "random walk" technique, which randomly picks a neighbor of the current node and migrates there. On each node the bus thread checks for the service and location. If the interesting service is found in the specified space, it switches to user task. The Bus Thread remembers the nodes that it has visited by recording their using a unique ID (IP addresses and port numbers) and avoid visiting them again.
Such an algorithm may be slow and not scalable, but one can hardly do better in an unstructured, dynamic network. However, if the network is not changing very fast or not changing at all, then a static directory of services can be maintained and used to find interesting nodes directly than moving from node to node to look for them. Such a structure can also be used by the Bus Thread to implement fast discovery algorithm. Another possible improvement is to allow the Bus Thread to fork itself and search the network in parallel. This optimization is currently under investigation.
As to the constraints, so far we have implemented the time constraint. The Bus Thread times each single iteration step, and checks the remained time budget after each single iteration step finishes. If the budget drops below zero, the iteration is stopped. So the time constraint is a soft deadline implemented with "best-effort" semantics. This soft deadline provides effective trade-offs between quality-of-results and time consumption as shown in section 5.3.

4.2 Transparent Process Migration

Transparent process migration is implemented as a native method, `migrate`, in a Spatial Views system class. It is used in the implementation of `foreach` and `forany` operations. `migrate` takes the destination node address as its parameter. When `migrate` is called, the Spatial Views KVM sends the whole heap to the destination, as well as the virtual machine status, including the thread queue, instruction counter, the execution stack pointer and other information.
The KVM running on the destination node receives the heap contents and the KVM status and starts a new process. Instead of ordinary process initialization, the receiving KVM populates its heap with the contents received from the network and adjusts its registers and data structures with the KVM status received from the network. To make `migrate` more efficient, we enforce a garbage collection before each migration.

5 Experiments

We use 10 Compaq iPAQ PDA's (Model H3700 and H3800) as our test bed, 2 of which are equipped with camera sleeves developed as part of the Mercury project at HP Cambridge Research Laboratory (CRL)

Fig. 4. Mercury Backpaq from HP CRL

(Figure 4). The iPAQ's were connected via 802.11b wireless technology. Since we have not implemented a location service based on GPS or other location technology, all node locations were statically configured in these experiments.

5.1 Application Example

We implemented the person search application discussed in Section 3.3. We timed the execution of the application on 10 iPAQ PDA's connected by a 802.11b wireless network. The network topology is shown in Figure 5. [1]

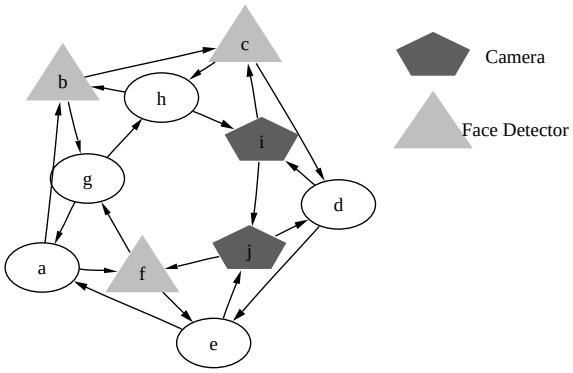


Fig. 5. Network for the Person Search Application

Node "i" and "j" have cameras, shown as dark gray pentagons in the figure; node "b", "c", and "f" provide the face detection service, shown

[1] In this paper, "network topology" refers to the network topology observed by one program execution. Another execution is very likely to observe a different topology, because the network is changing.

as light gray triangles in the figure. The program starts from node "a" and eventually visits all the nodes in the network in the depth-first order. Once it finds a node with a camera, it takes a picture and checks if there is a red region in the picture. If there is, the program will look for a node providing face detection service. It stops on the first node with the service and looks for a face in the picture. If a face is found, and it is close to the red region in the picture, the program records the location where the picture is taken. Once the program finishes all the nodes, it migrates back to the starting node.
We experimented with two situations. Situation 1: A red region is detected on both node "i" and "j", but a face is found only in the picture from node "j". Situation 2: No red region is detected on either node "i" or "j", so no face detection is triggered. We timed the executions in both situations. The program took on average 23.1 seconds in situation 1 and 10.0 seconds in situation 2. In both cases, the time constraint was not violated. It is important to note that all the iPAQ's use SA-1100 StrongARM processors running at 206MHz. But the nodes that provide face detection service offload the face detection computation to a PC. The execution times for the first situation was dramatically reduced as suggested in [17].

5.2 One-Hop Migration Time

To assess the efficiency of execution migration, we measured the one-hop migration time. We measured the overall execution time of two consecutive migrations, one migrating to a neighbor, followed by another one migrating back. The time taken by those two consecutive migrations is the round-trip time for one-hop migration, which is twice the migration time. We measured the time for different live data size (The heap size is 128KB, but only live data are transfered.) The result is shown in Figure 6 using a wired 100Mbps and a wireless 11Mbps (802.11b) Ethernet connection.
In the KVM heap, there is a permanent space which is not garbage collectible. For our test program, the size of the permanent space is 65KB (66560 bytes). This includes garbage (e.g. one-time used strings) as well as Java system classes which are available on all the nodes. The current implementation transfers the entire 65KB in a migration operation. We are making efforts to modify the KVM module for garbage collection and memory management to avoid transferring the whole permanent space in a migration. We expect a resulting speed up of the migration by an order of magnitude.

5.3 Effects of Timeout Constraints

To evaluate the effects of timeout constraints, we fake failures with a probability p for the network links. The test program iterates over "temperatures sensors" and reads the temperatures to calculate the average temperature. After finishing on each node, the program tries to connect to a neighbor. If none of the neighbors is reachable, the program waits for

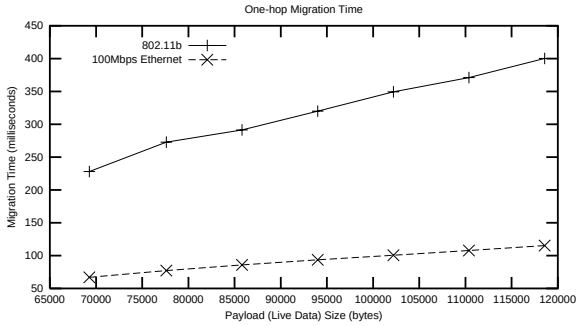
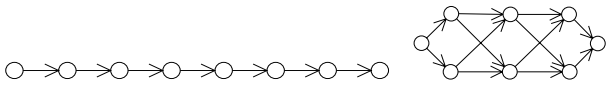


Fig. 6. One-Hop Migration Time



(a) (b)

Fig. 7. Topology

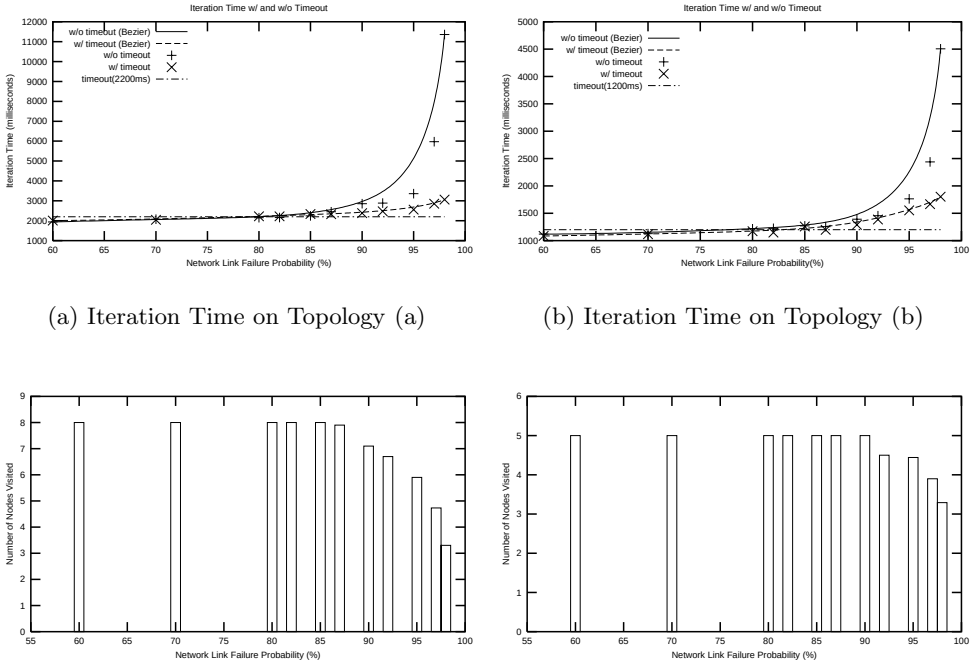
10ms and tries again. And it keeps trying until it successfully migrates to a neighbor.

If the network link failure probability is high, the iteration time might be very long. In that case, the timeout constraints can significantly reduce the iteration time and still get some result. We did the experiments with two different topologies shown in Figure 7(a) and 7(b), with the experimental results shown in Figure 8.

The time to wait before a successful migration is $10ms \times \frac{1}{1-p}$, where p is the failure probability of all the links to the neighbors. In Topology (a), $p = p_l$, where p_l is the failure probability of a single link. In Topology (b), $p = p_l^2$. Then the time of a single iteration step is $10ms \times \frac{1}{1-p} + 400ms$, where 400ms is the maximum one-hop migration time(see Figure 6).

If no time constraint is imposed, the expected execution time is $(n-1) \times 10ms \times \frac{1}{1-p} + (n-1) \times 400ms$, where n is the number of nodes visited. We omit the task execution time on each node, because the temperature reading is so fast that the time it takes is much less than migration and waiting time.

If a timeout $t_{timeout}$ is specified, the expected program execution time will be $\leq t_{timeout} + 10ms \times \frac{1}{1-p} + 400ms$. For Topology (a), link failure probability p_l=98% and $t_{timeout}$ =2200ms, that upper bound is 3100ms , which is verified by the experimental result, 3095ms (see Figure 8(a)). For Topology (b), p_l=98%, and $t_{timeout}$=1200ms, that upper bound is



(a) Iteration Time on Topology (a)

(b) Iteration Time on Topology (b)

(c) Number of Nodes Visited with Timeout Constraint (Topology (a))

(d) Number of Nodes Visited with Timeout Constraint (Topology (b))

Fig. 8. Effects of Timeout

1850ms. which is also verified by the experimental result, 1802ms (see Figure 8(b)).
Using time constraints, a programmer is able to keep a decent quality of result of the program, while significantly reducing the execution time. Instead of producing no answer (as it happens when a user presses "Ctrl-C" in a traditional programming environment,) the program reports a result of reduced quality (e.g. only two temperature readings.) when the time budget is used out. The number of nodes visited in our experiments, as the criterion for quality of result, is shown in Figure 8(c) and 8(d).

6 Conclusion

Spatial Views is a programming model that allows the specification of programs to be executed on dynamic and resource-limited networks of embedded systems. In such environments, the physical location of nodes is crucial. Spatial Views allows a user to specify a virtual network based

on common node characteristics and location. Nodes in such a virtual network can be visited using an iterator or selector. Execution migration, node discovery, or routing is done transparently. Time and other resource constraints allow the programmer to express quality of result trade-offs and to manage the inherent volatility of the underlying network.
The Spatial Views programming model is simple and expressive. A prototype of Spatial Views including a compiler, a runtime library and a virtual machine, has been implemented as an extension to J2ME. Experimental results on a network of up to 10 iPAQ's handheld computers running Linux are very encouraging for a person search application. In addition, the effectiveness of time constraints to allow graceful degradation of the quality of a program's answer was experimentally evaluated for a temperature sensor network with two different network topologies. Spatial Views is one of the first spatial programming models with a best-effort semantics. The model allows optimization such as parallelization (multiple threads), and quality of result vs. resources usage trade-offs.

References

1. Mike Addlesee, Rupert Curwen, Steve Hodges, Joe Newman, Pete Steggles, Andy Ward, and Andy Hopper. Implementing a sentient computing system. *IEEE Computer*, August 2001.
2. W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *SOSP*, 1999.
3. Cristian Borcea, Chalermek Intanagonwiwat, Akhilesh Saxena, and Liviu Iftode. Self-routing in pervasive computing environments using smart messages.
4. Cristian Borcea, Deepa Iyer, Porlin Kang, Akhilesh Saxena, and Liviu Iftode. Cooperative computing for distributed embedded systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, July 2002.
5. David Gay, Phil Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. In *PLDI*, 2003.
6. Ivan A. Getting. The global positioning system. *IEEE Spectrum*, December 1993.
7. Robert S. Gray. *Agent Tcl: A flexible and secure mobile-agent system.* PhD thesis, Dartmouth College, June 1997.
8. Robert S. Gray, George Cybenko, David Kotz, Ronald A. Peterson, and Daniela Rus. D'agents: Applications and performance of a mobile-agent system. *Software: Practice and Experience*, May 2002.
9. Andy Harter and Andy Hopper. A distributed location system for the active office. *IEEE Network*, 8(1), 1994.
10. Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, and Paul Webster. The anatomy of a context-aware application. In *MobiCom*, 1999.
11. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors. In *ASPLOS*, 2000.

12. 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.
13. Sun Microsystems Inc. *Java 2 Platform, Micro Edition (J2ME).*
14. Sun Microsystems Inc. *KVM White Paper.*
15. Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *MobiCom*, 2000.
16. Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. In *ASPLOS*, 2002.
17. U. Kremer, J. Hicks, and J. Rehg. A compilation frame work for power and energy management on mobile computers. In *International Workshop on Languages and Compilers for Parallel Computing (LCPC'01)*, August 2001.
18. Joanna Kulik, Wendi Rabiner, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCom*, 1999.
19. Philip Levis and David Culler. Mate: A tiny virtual machine for sensor networks. In *ASPLOS*, 2002.
20. Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom*, 2000.
21. Nissanka B. Priyantha, Allen K. L. Miu, Hari Balakrishnan, and Seth J. Teller. The cricket compass for context-aware mobile applications. In *MobiCom*, 2001.
22. H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:23–38, 1998.
23. M. Satayanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, August 2001.
24. Jim Waldo. The Jini architecture for network-centric computing. *ACM Communications*, July 1999.
25. Mark Weiser. The computer for the 21st century. *Scientific American*, September 1991.
26. Gang Xu, Cristian Borcea, and Liviu Iftode. Toward a security architecture for smart messages: Challenges, solutions, and open issues. In *Proceedings of the First International Workshop on Mobile Distributed Computing*, May 2003.