Smart Messages: A System Architecture for Large Networks of Embedded Systems

Phillip Stanley-Marbell,* Cristian Borcea, Kiran Nagaraja, Liviu Iftode,
Department of Computer Science
Rutgers University
Piscataway, NJ 08854

{narteh@ece, borcea@cs, knagaraj@cs, iftode@cs}.rutgers.edu

ABSTRACT

Embedded systems greatly outnumber desktop computing systems and are witnessing rapid growth as increasing numbers of industrial, commercial and consumer devices incorporate embedded intelligence. In addition, there are emerging application classes which were hitherto not feasible, but will be enabled by the programmability and connectivity of these pervasive computing elements. The scale and fragility of these networks make traditional distributed computing models inadequate.

Presented is a system architecture, Smart Messages, for computation and communication in large networks of networked embedded systems. In this model, communication is realized by sending Smart Messages in the network. These messages are comprised of code, which is executed at each hop in the path of the message, and a payload which the message carries in the network. The execution of the message at each hop determines the next hop in the path of the message, thus smart messages are responsible for their own routing.

This paper details the motivation for Smart Messages, describes the smart message model and proposes a system architecture to support this model. We conjecture that smart messages provide a suitable distributed computing model for networks of embedded devices and discuss some design issues.

1. INTRODUCTION

Embedded systems have outnumbered traditional desktop computing systems for several years and will continue to do so in the future. A new trend in embedded systems is to provide networking, either wired or wireless. As device costs plummet, it will soon be feasible to deploy networks of embedded computing devices of the order of tens of thousands to millions of nodes. These networks will be inherently heterogeneous, both in the interconnection technologies, and in the individual functions of the component nodes. Each node will be targeted towards performing a specific task, such as sensing motion, and the hardware will be accordingly specialized. These embedded computing devices will typically be mobile and have constrained energy resources, and are very likely to be battery powered. Even in cases where devices are not mobile and could be connected to a permanent source of power, the cost and difficulty of electrical wiring of power to individual devices might preclude the use of a permanent power source.

Networked embedded systems poses a unique set of challenges. Unlike the Internet, these networks will typically be deployed in situations void of human attention, situations in which it is unacceptable to require a human to hit a "reset" button to recover from a failure. The availability of nodes may vary greatly with time, with nodes becoming unreachable due to mobility, depletion of energy resources or catastrophic failure. Traditional naming architectures such as TCP/IP, that provide unique names on a per device basis, and depend on binding of properties to names at the time of message dispatch are inappropriate for networks of such scale and volatility. Instead, alternative addressing schemes where the binding between a desired destination and a unique name may vary over the lifetime of a message traversing the network are required [1].

Despite the architectural and interconnection diversity of these networks, it should be possible to utilize them to perform global tasks, ranging from those as simple as collection, aggregation and delivery of data from sensors and routing of data traffic in the network, up to more complex tasks like collectively tracking motion across a geographical area. Individual nodes will have unique properties such as geographic location, motion, energy resources or hardware. For applications to be able to fully harness these unique resources, we believe that these computing elements must be programmable and support user-defined applications. Merely providing system support for remote programmability doesn't solve the problem.

To benefit from programmability, and the aggregated computing resources deployed in these networks, new distributed computing models must be employed, which will necessarily be different from the traditional distributed computing models for several reasons. First, at such large network scale, a property-based naming rather than a unique identifier for the participant nodes in the computation must be supported. Second, given the fluidity and volatility of these networks in terms of node configuration and network topology, it may be impossible to synchronize computation, and round-trip communication may never complete. Third,

*Dept. of Electrical and Computer Engineering

[1] Intentional naming [2] has recently addressed this problem for IP networks.

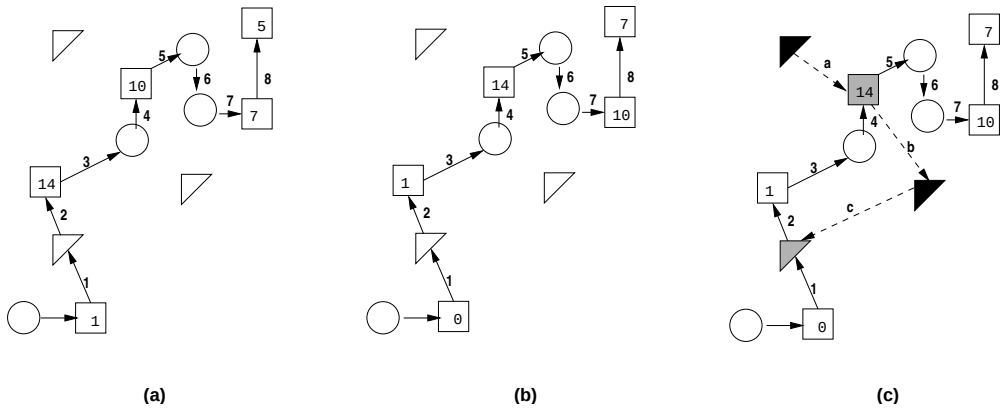

Figure 1: Smart Message Model Example

applications should accept partial execution as long as it is relevant, i.e. meets a certain **quality of result** (QoR).

We propose a new system architecture for large networks of embedded systems, based on the idea of intelligent carriers of data in a network. Communication in the network consists of collections of code and mobile data referred to as Smart Messages. Smart Messages migrate through the network, a single network hop at a time, executing at each step. They are responsible for determining their own paths through the network, utilizing a minimal set of facilities provided by nodes in the network. Nodes in the network that support Smart Messages are referred to as *cooperative* nodes, and they provide architecturally independent environments for the receipt and execution of Smart Messages.

We conjecture that the Smart message architecture is flexible enough to support a wide variety of applications, ranging from data collection and dissemination, content based routing and object tracking to more traditional distributed computing applications in which execution of a task is spread across a collection of devices.

The rest of this paper is organized as follows. The following section describes the model of use of Smart Messages. Section 3 describes the Smart Message architecture, followed by section 4, which discusses issues that must be investigated in the Smart Message architecture. Section 5 discusses related work and Section 6 describes the status of the research.

2. MODEL

We propose a distributed computing model based on **Smart Messages** (SM) and cooperative devices in a network. A Smart Message (SM) is an intelligent carrier of data in the network. [2] It migrates through the network one network hop at a time, executing at each step. The execution performed at each step may differ based on peculiar properties of that node. For example, on nodes in the network which have sensors of interest to the SM, it may read, and even process sensor data. Other nodes in the network might be used as ad-hoc "stepping stones" between nodes of interest.

Figure 1a illustrates a network consisting of three types of nodes, represented with squares, circles and triangles. The nodes represented by squares are nodes of interest to an SM which is launched from the circular node in the lower left of Figure 1a. The goal of the application implemented by this SM is to visit the five square nodes and to propagate a local data item of each node to the next one visited in order. The numbered arrows show the path and numerical order of nodes visited by the SM. The network maintains no routing infrastructure, and the SM is responsible for determining a path to its destination, the square node marked with the number '5' in the figure. The SM may use other nodes in the network, the circular and triangular nodes, as intermediates hops as it navigates through the network.

In moving from one node to the next, the SM must therefore carry with it the last value it read, maintaining state as it moves from node to node through the network. Figure 1b shows the state of the network after the SM is done navigating the network, and thus the application it implements is complete. This SM may represent a simple object tracking application, in which the motion of the SM through the network is determined by values it reads at sensors on nodes, the SM's path being recorded in the state of the network.

Devices in a network of embedded systems will typically have a wide variety of processor and system architectures. To support SMs across heterogenous architectures, it will be necessary to provide a hardware abstraction layer such as a virtual machine, that shields SMs from the peculiarities of the individual node architectures. Cooperative nodes provide a limited data store, termed the *Tag Space*, which is persistent across the executions of SMs. An SM executing at a node may read certain Tags and may create new Tags in the node's Tag Space. In our example, the data the SM transports from one node to another gets written into the destination's Tag space. Tags can also be used for naming and routing as well as for data exchange, data sharing and synchronization between SMs.

During its execution on a node in the network, an SM may spawn new SMs which may be sent out to other cooperative nodes in the network. A "reliable" SM may, for instance, spawn SM(s) to collect routing information or discover the path to the next node of interest before leaving. A collection of such SMs executing in a network constitute an application. A network may simultaneously contain several executing applications, as illustrated in Figure 1c. In the figure, nodes of interest to the first application are the square

[2]Smart packets [16], mobile agents [20, 10, 14] and active messages [19] can also be seen as "intelligent carriers" of data. Similarities and differences are discussed in the related work section.

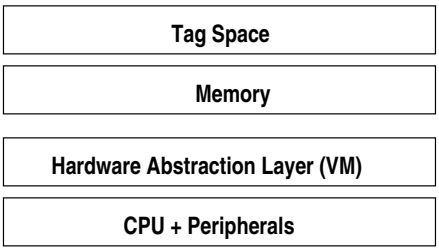

Figure 2: Node Architecture

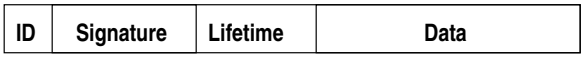

Figure 3: Tag Structure

nodes, and nodes exclusive to the path of this application are colored white. Nodes of interest to the second application are depicted with triangles, and nodes exclusive to its path are colored black. Nodes in the network which may at some point be either part of the first or second application are colored grey. For example, the grey triangle is just a "stepping stone" on the path of the the first application, but is a node of interest to the second application. Likewise the grey square is a node of interest to the first application but is just an intermediate host for the second application.

3. SYSTEM ARCHITECTURE

The goal of the Smart Message architecture is to keep the support required from nodes in the network to the bare minimum, placing intelligence in the Smart Messages rather than in individual nodes. Placing intelligence in SMs, and providing a common minimal support from cooperative nodes, provides flexibility and obviates the need for the potentially impossible task of updating all nodes in a network for the implementation of a new application or protocol. [3]

Figure 2 illustrates the general architecture of a device, termed a *cooperative node* that supports the execution of Smart Messages. The primary logical components of such a device are the processor and peripherals, a virtual machine to provide a hardware abstraction layer for executing SMs, memory for loading smart messages, and global Tags, which are persistent across Smart Message executions. The virtual machine provides a uniform view to executing SMs and incorporates basic system software support. It implements the security policies of the cooperative device, preventing potentially malicious SMs from accessing restricted resources. The Scylla virtual machine [17] was designed specifically to support architectures such as Smart Messages, and it is the intended virtual machine for an initial implementation. The architecture is however not tied to Scylla, which may be replaced with another virtual machine such as Sun Microsystems KVM [18].

3.1 Tag Memory Structure

[3]Some of these arguments were presented in the active networking community with modest success. Unlike in active networks, scalability is of primary importance, and partial results may be acceptable in networks of embedded systems.

Each node that supports Smart Messages manages a structured memory region, the *Tag Space*, consisting of a limited number of *Tags* that are persistent across the execution of SMs. Although the Tag space provides persistent storage across SM execution, it is not specifically intended to be implemented with any particular type of storage medium, and is equally likely to be implemented with volatile RAM or nonvolatile FLASH memory.

Each tag consists of an identifier, a digital signature, lifetime information, and data. The identifier field is the name of the Tag, and is similar to a file name in a filesystem. An SM may only write to an existing Tag or delete a tag if its signature matches that in the signature field of the Tag. Smart Message signatures are discussed further in the following subsection. The tag lifetime specifies the time at which the tag will be reclaimed by the virtual machine from the tag space. Figure 3 illustrates the structure of Tags on a device.

Tags may be used for a myriad of applications. They may be used to store state in a network. They may indicate the state of a node, for example a node might have a Tag which represents a local sensor. Reading this Tag returns a reading from the sensor, and SMs cannot delete such a Tag. SMs may use Tags to name nodes of interest or to store routing information. For example SMs which are part of an application that carries data through a network may create tags at visited nodes in the network, caching discovered route information. This information is stored in the data portion of a tag, thus an application may implement traditional routing algorithms, using Tags to store routing tables. Tags may also be used for synchronization as discussed further in Section 3.3.

3.2 Smart Message Format

Smart Messages are comprised of a digital signature, code and data sections, and a resource table. The digital signature identifies an SM, and is used by cooperative nodes to enforce access of an SM to Tags. The code and data sections are comprised of components referred to as *bricks*. Each code brick is an independent program that may be used together with the other code and data bricks to generate a new, possibly smaller SM. The data bricks contain mobile data of the SM that can be accessed by the SM during execution. The resource table consists of Tag identifiers, execution starting points and resource estimates (execution time, new tags to be created, communication traffic, etc). Before the SM is scheduled for the execution, the Tags in the resource table are checked against local Tags, to set the resource limits usage and the starting point in the execution. The resource estimates denote a bound on the expected need of the SM at a node, and may be used for scheduling or to reject SMs that the node cannot satisfy. Figure 4 depicts the structure of a Smart Message.

3.3 Smart Message Execution

Executing Smart Messages are embodied in *Tasks* and executed over the virtual machine. During this execution, a task may modify the data section of the message as well as the local Tags to which it has access, or may send new smart messages comprised of one or more of the bricks contained in the original Smart Message, to another node.

An executing task is never preempted by the virtual machine, but a task may yield the virtual machine by explicitly

Figure 4: Smart Message Architecture

requesting to do so, by blocking on a Tag to be written on the local node by another SM. However, if the execution time estimate based on which the task was scheduled expires, the task can be forcefully terminated. New Smart Messages can always be received at a node provided there are sufficient memory resources, but the corresponding tasks will be scheduled only after the current task completes its execution on yields the VM. When an executing Task terminates or blocks, the virtual machine may select a blocked Task for execution, if the tag on which it is blocked has been written in the meantime (by other tasks) or its lifetime has expired.

This mechanism can be used by an SM which spawns SMs for route discovery then blocks on a routing tag. When a spawned SM returns, it updates the this tag and exits, causing the original SM to become unblocked. Since tags are persistent, routing information once acquired can be used by subsequent tasks with similar interests, thus amortizing the routing discovery effort.

4. DESIGN ISSUES

In the following, we briefly outline some design issues that confront the Smart Message architecture.

The networks in which Smart Messages are deployed will be inherently volatile, due to node mobility, energy resource constraints, and catastrophic failure. It is essential to be able to provide some notion of functionality even under these conditions. It might be necessary to tradeoff energy efficiency for reliability - engaging in extra communications by generating extra Smart Messages to provide redundancy.

It is desirable to be able to amortize the cost of operations performed by a Smart Message over time and across the executions of other Smart Messages. This is possible through the use of Tags to cache data at nodes that may be subsequently read by migratory Smart Messages. The requirements on sizes of Tag Spaces to make this effective remain to be investigated, as do the Tradeoffs between Tag Space size and performance, network reliability, energy efficiency, and device cost.

It will often be desirable to obtain partial results from a computation. The concepts of correctness of a computation will need to be reconsidered, and a metric for the quality of a result developed.

The requirement to carry code in Smart Messages raises the issue of overhead in terms of code transmission in the network and there is a tradeoff between the flexibility provided and this overhead. Several schemes may be used to reduce this overhead, for example cooperative nodes may cache code bricks of Smart Messages.

Given the wide variety of processor and system architectures that will exist in the network, it is necessary to provide a hardware abstraction layer to shield applications from the details of the individual node architectures. The amount of system software support required, and whether this should be included in the virtual machine implementation or placed at a layer below it is an open question. If this virtual machine enforces security policies, should these policies be set per device or across an entire network, and what kind of security infrastructure is appropriate.

It will be necessary to define a programming model for constructing applications made up of Smart Messages. The facilities provided by the programming interface and language must match those of the underlying architecture. Whether current solutions are sufficient, or new solutions are needed remains to be seen.

5. RELATED WORK

Smart messages bear some similarity to active messages [19], smart packets [16] and mobile agents [20, 10, 14].

Like active messages, the arrival of a smart message at a node leads to the execution of a task on the node. Unlike Active messages that point to a handler at the destination, smart messages carry code with them. Beyond the superficial similarity between the Smart Messages and Active Messages, the two models address two completely different problems. Active messages target fast communication in system-area networks and therefore, the handler execution is short and triggered as soon as the active message arrives. On the other hand, smart messages target remote programmability of massive networks of embedded devices. Energy savings and quality of result are typically more important in these scenarios than performance.

Smart messages are similar to mobile agents [20, 10, 14] which also involve migration of code in the network. A mobile agent may be viewed as a task that explicitly migrates from node to node assuming the underlying networking assures its transport between them. Smart messages unlike mobile agents, are defined to be responsible for their own routing in a network. The smart message architecture further defines infrastructure that nodes in a network supporting smart messages must implement, which makes this self routing in smart messages possible.

The smart packet architecture [16] provides a flexible means of network management through the use of mobile code. Smart packets are implemented over the Internet Protocol architecture, using the IP options header. They are routed just like other data traffic in the network, and only execute on arrival at a specific location. Unlike smart packets, Smart Messages are executed at each hop in the network and his execution determines the net hop in the route. Smart Messages, unlike smart packets encapsulate the state of the application as it is executed at each hop in the network.

Mobile ad hoc networking research such as[11, 3, 13, 12] and its applications [15], has resulted in numerous routing protocols for peer-to-peer multi-hop networking in infrastructures without base-stations. These protocols have generally been designed for networks based on the Internet Protocol, and have been targeted primarily towards traditional mobile computing applications such as mobile personal computers and PDAs. These protocols can be leveraged and implemented over the smart message architecture.

Recent work specific to large networks of embedded systems has focused networking protocols for wired and wireless sensor networks [4, 9, 7, 6] and system architectures for fixed function sensor networks [8]. This research is comple-

mentary to the Smart Message architecture. It is our hope to provide enough flexibility in our architecture to enable the implementation of these models over the Smart Message architecture.

6. STATUS

We plan to validate the model through simulations and a hardware prototype implementation. We are in the process of finalizing a simulation platform for this architecture, with which we plan to investigate tradeoffs in the Smart Message architecture such as reliability through the use of multiple SMs versus energy costs, scheduling and routing policies as well as scalability of the model across a large network of devices.

We plan to investigate the implementation of previously proposed protocols for data dissemination [7], data collection [9] and energy efficient routing [6], over the Smart Message architecture.

We have commenced a hardware prototype implementation, with a implementation of the Scylla virtual machine[17] for the Hitachi SH3 architecture. We plan to extend this implementation to other microprocessor and microcontroller platforms, and also to implement a test-bed using wireless networking technologies such as Bluetooth [5, 1].

7. REFERENCES

[1] Bluetooth Special Interest Group. `http://www.bluetooth.com`.

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

[3] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, 1998.

[4] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of the Fifth annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 263–270, 1999.

[5] J. Haartsen, M. Naghshineh, J. Inouye, O. Joeressen, and W. Allen. Bluetooth: Visions, goals, and architecture. 2(4):38–45, October 1998.

[6] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Hawaii International Conference on System Sciences*, January 2000.

[7] 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*, pages 174–185, 1999.

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Networked Sensors. pages 93–104. Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, November 2000.

[9] 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.

[10] D. Johansen, R. van Renesse, and F. Schneider. Operating system support for mobile agents. In *5th IEEE Workshop on Hot Topics in Operating Systems*, 1995.

[11] D. B. Johnson and D. A. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*. T. Imielinski and H. Korth, (Eds.). Kluwer Academic Publishers, 1996.

[12] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. 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.

[13] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 255–265, 2000.

[14] D. Milojicic, W. LaForge, and D. Chauhan. Mobile objects and agents. In *USENIX Conference on Object-oriented Technologies and Systems*, pages 1–14, 1998.

[15] R. Morris, J. Jannotti, F. Kaashoek, J. Li, and D. S. J. D. Couto. CarNet: A scalable ad hoc wireless network system. In *Proceedings of the 9th ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System*, Kolding, Denmark, September 2000.

[16] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge. Smart Packets for Active Networks. *ACM Transactions on Computer Systems*, pages 397–413, February 2000.

[17] P. Stanley-Marbell and L. Iftode. Scylla : A Smart Virtual Machine for Mobile Embedded Systems. In *3rd IEEE Workshop on Mobile Computing Systems and Applications*, pages 41–50, December 2000.

[18] Sun Microsystems. *Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices*. May 2000.

[19] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th annual international symposium on Computer architecture*, pages 256–266, May 1992.

[20] J. White. *Mobile Agents*. J. M. Bradshaw (Ed.), MIT Press, 1997.