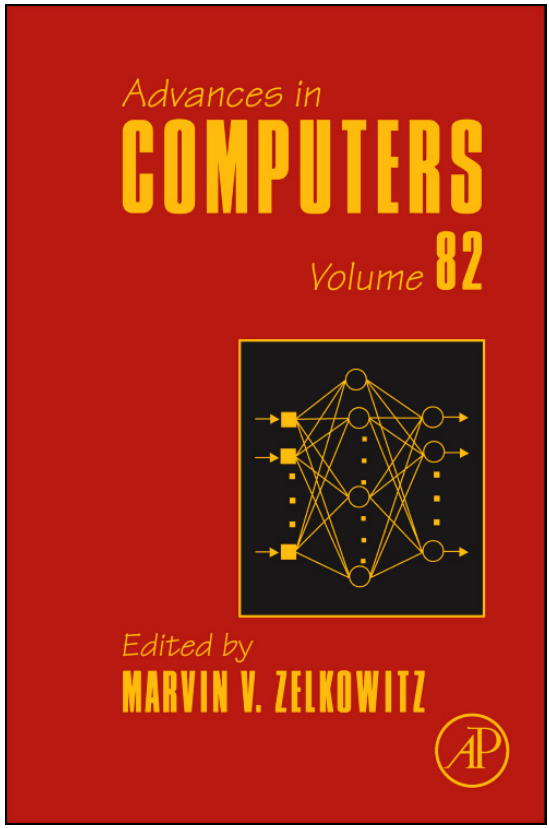

From: Sergio González-Valenzuela, Min Chen, Victor C.M. Leung, Applications of Mobile Agents in Wireless Networks and Mobile Computing. In Marvin V. Zelkowitz, editor: *Advances in Computers,* Vol. 82,
Burlington: Academic Press, 2011, pp. 113-163.
ISBN: 978-0-12-385512-1

Applications of Mobile Agents in Wireless Networks and Mobile Computing

SERGIO GONZÁLEZ-VALENZUELA

Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada

MIN CHEN

School of Computer Science and Engineering, Seoul National University, Korea

VICTOR C.M. LEUNG

Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada

Abstract

We examine the applicability of mobile software codes to perform networking tasks in wireless and mobile computing environments. We contend that the advent of wireless technologies during the past decade has turned computer networks increasingly complex to manage. In particular, factors such as context awareness and user mobility are now crucial in the design of communications protocols used by portable devices with moderate to severe bandwidth and battery power limitations. Unlike hard-coded communication protocols that fulfill a specific need, mobile software agents can be deployed to deal with a range of tasks and can be designed to efficiently adapt to diverse circumstances. We present our latest advancements in the areas of mobile *ad hoc* networking and wireless sensor networks using mobile agents (MAs). We also elaborate on the importance of engineering an efficient migration strategy as the single most distinctive

ADVANCES IN COMPUTERS, VOL. 82
ISSN: 0065-2458/DOI: 10.1016/B978-0-12-385512-1.00004-9

proceeding that an MA performs to operate efficiently. In addition, we describe the *Wiseman* system for scripting MAs that can perform networking tasks in both homogeneous and heterogeneous wireless network environments. Monitoring, tracking, and E-healthcare applications are discussed and evaluated at length.

1. Introduction

Many definitions have been offered throughout the literature to describe mobile agents (MAs). Essentially, a software agent differentiates itself from a regular computer program by having the ability to observe and estimate the current state of the environment where it executes, deciding how to act based on this information and then performing a corresponding action [1]. In addition to this, an MA has the distinctive feature of being able to arrange for its forwarding from one device to another. Conversely, regular programs do not incorporate any feedback mechanism when making a decision and do not possess the mobility feature. There are many types of MAs, but they are generally categorized as being at the opposite side of the communications mechanisms plane, in which shared memory is the simplest scheme, as illustrated in Fig. 1. Message passing is the oldest and most employed data communications mechanism, whereby a network device encapsulates information in a tight fashion before transmitting it. However, devices can also forward either interpretable programs or executable binary codes that encapsulate instructions for local or remote execution. We are particularly interested in the applicability of interpretable programs that are parsed and executed by virtual machines implemented by networked devices.

1.1 Foundations of Mobile Agent Systems

Early discussions in this area argued that there are no MA applications, but rather applications that benefit from the use of MAs [2], which has proven a sensible statement after more than a decade of research into the subject. Similarly, MAs can be regarded as the enabling technology for a networking application. In general, security issues have proven to be the most discouraging factor toward a wider

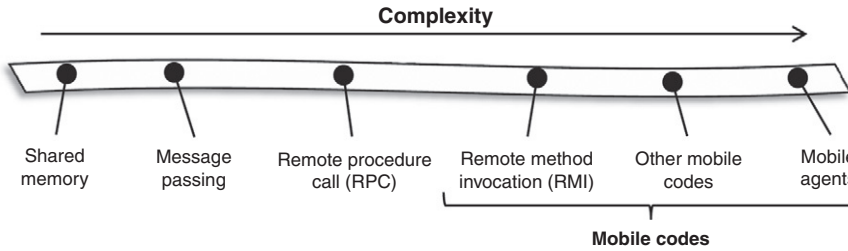

FIG. 1. Data communications mechanisms.

adoption of MA technology [3–5]. In other words, opening the doors to seemingly benign, interpretable programs make networked devices susceptible to a number of malicious attacks. As a result, we observe that certain types of networks have benefited more than others from implementing MA-based applications in real systems. In general, closed networks provide a secure environment to deploy MA technology and benefit from its unique features. Thus, not only the application type but also the network environment plays an important role for deciding how an MA-based solution should realize one or more tasks. Figure 2 illustrates the implementation types that software agents can take.

The most important characteristic of the MA approach and software agents in general is that it allows *programmability*, thus enabling an MA-based system to change its operation on demand to adapt the circumstances determined by the underlying environment. For example, a network with sufficient bandwidth can support the deployment of many MAs collaborating to achieve a number of tasks. Nevertheless, a sudden bandwidth shortage as sensed by these agents could activate a low-usage mode of operation, whereby only a few agents perform only the most crucial tasks. It follows that the MAs' policies can be either static or variable. By the same token, the most important benefit introduced by the mobility feature of a software agent is that it enables better bandwidth usage by moving the processing element to the location where the data to be analyzed resides. For example, moving a 500 KB program to process 100 MB of data at a remote location is more bandwidth

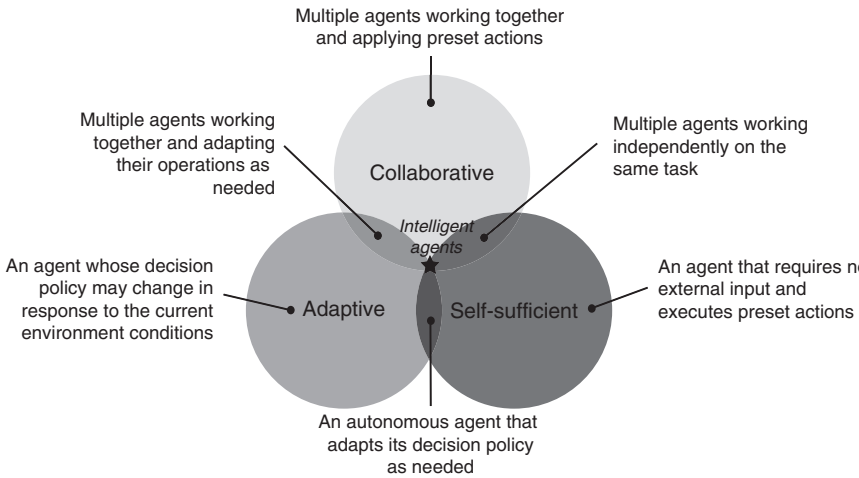

FIG. 2. Implementation types of software agents.

efficient than carrying out the reverse procedure. However, it is also clear that implementing an efficient migration policy is crucial in achieving this bandwidth-saving goal.

Thus far, we have pinpointed some key aspects that are relevant to deploying MAs: flexibility to implement diverse applications, adaptability to deal with unforeseen situations, efficient migration mechanisms to improve performance, an application-dependent strategy, and preference for closed-network deployment. It is straightforward to see that MAs are best suited for highly specialized applications in access-restricted networks that are subject to unexpected, variable conditions, and resource constraints. As a result, we turn our attention to exploring the applicability of MAs to support diverse tasks in wireless and mobile networks. These types of networks possess some or all the peculiarities just mentioned. In particular, we direct the focus of our investigations to wireless sensor networks (WSN) and *ad hoc* networks created by portable devices. However, to better understand the relevance of investigating MA applicability in these networks, we present a concise discussion on the advantages and disadvantages of MA technology, followed by a brief historical perspective with concerning previous research efforts involving MA technology, its shortcomings, the current state of affairs, and what we can expect to see in the near future.

1.2 Advantages and Disadvantages of Using MAs

The benefits and drawbacks of the MA approach were extensively discussed in the initial years of its research. In general, there are some advantages that are attributable to all agent types, whereas others are more specific. For instance, compactness is oftentimes referred to as an inherent MA characteristic, though this is not always the case. For instance, an MA coded to perform a complex brokering task requires that a significant amount of functionality be implemented into it to deal with a wide variety of possible situations for the transactions it supports. However, MAs used for active networking tasks (e.g., routing) can be significantly more compact because they are targeted at specific tasks with well-known outcomes. Another advantage regularly associated to using agents is bandwidth savings, which can be achieved if an efficient agent migration policy is employed. However, it is possible that the bandwidth overhead incurred by moving a relatively large agent could offset the one incurred by using a simple message-passing scheme, depending on the application. Still, the bandwidth-savings potential remains by far one of the most compelling reasons for using agents [6].

Figure 3 exemplifies the bandwidth savings-feature by showing both a traditional and an MA-based approach for collecting data in a WSN. In the first case, the occurrence of an event as sensed by individual WSN nodes initiates the

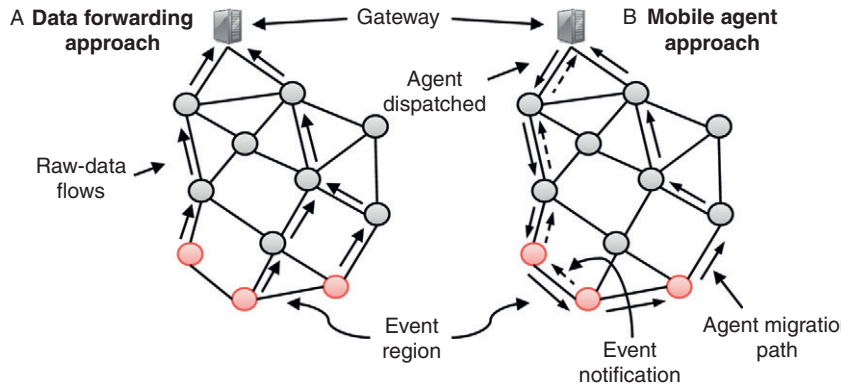

FIG. 3. Raw data forwarding versus mobile agent processing.

corresponding client–server interactions to send raw data to the WSN gateway or sink for subsequent analysis. In this approach, each client–server session incurs one data flow from source to destination, leading to higher bandwidth utilization. Moreover, this method places a higher burden in the nodes closer to the WSN gateway because their links observe heavy data traffic as compared to the wireless links located farther away from the gateway. Conversely, the mobile agent system (MAS) approach dispatches an agent to the WSN's region of interest (ROI) where the event was observed. Once there, the MA processes data and sends back to the WSN gateway either a concise assessment of the situation or a digest of the analyzed data. This has the benefit of incurring a single traffic flow, in contrast to the client–server approach that observes multiple flows.

In addition to enabling bandwidth savings, MAs can also help reduce processing delay. Revisiting the example shown in Fig. 3, the actual data pooling process triggered by an event in a WSN region might require multiple interactions between the WSN gateway and the nodes involved. This obeys to the possibility of having a relatively large amount of data that have to be progressively transferred if, say, the first data block yields no conclusive results and one block or more need to be pooled from the corresponding devices until a result is found. Conversely, in the MA approach, the codes that actually process data migrate to the devices that triggered the event, and the analysis is realized *in situ*, so that no messages or data need to be sent back and forth from the devices to the WSN gateway. In addition, if a result is not found, the MA can migrate to another node to immediately begin analyzing more data, whereas using the message-passing approach entails initiating a new session between the WSN gateway and another device, followed by the respective data

transfer process. It is clear that the MA approach incurs less delay during its migration process than consecutively forwarding raw data segments. Resilience is yet another advantage that an MA-based solution can incorporate in environments whose behavior is unstable or highly uncertain. For instance, whereas a message-passing scheme incurs significant signaling to recover from failures during an ongoing data transfer session under adverse circumstances (e.g., in the presence of a noisy-channel, or frequent disconnections), an MA-based solution could have the agent monitor the channel conditions until the circumstances are favorable to migrate back to the WSN sink with the desired information.

The previous discussion provides some compelling reasons in favor of using MAs to solve distinct networking tasks. In fact, it is easy to see that these advantages are highly appealing for the case of wireless and mobile networks. However, there are important counterarguments against MA technology. For instance, bandwidth savings can only be achieved if the size of one or more MAs performing a task is sufficiently compact to offset the bandwidth otherwise incurred by employing the message-passing mechanism. This might be hard to achieve if the MA is coded to provide added resilience, thereby sacrificing compactness as per the extra codes that implement this added feature. Another aspect that adds complexity to an MA-based solution is the migration strategy employed to visit multiple nodes, either throughout the network or in a portion thereof. An inefficient migration strategy incurs added bandwidth because of the total number of times that one or more MAs hop to accomplish a certain task. However, a carefully engineered migration strategy would ostensibly be capable of achieving better results. It thus follows that large-sized MAs implementing an inefficient migration strategy would result detrimental to the overall system's performance. In addition, the effectiveness with which an MA solution provides resilience depends directly on the programmer's ability to anticipate and deal with situations that the MA could encounter. In the message-passing mechanism, the respective communications protocol daemon running into an unexpected situation could simply reschedule the data transfer process at a later time if the current circumstances are unfavorable. Conversely, an MA could remain stranded at a remote node, perhaps unable to return to the network's gateway upon encountering a situation that steered it into a deadlock state. An additional issue that can be used to argue against MA technology is that of security. To this regard, it is easy to see that an attacker could inject a malicious agent into a network to disrupt its normal operation (e.g., as a typical computer network virus). Conversely, one or more malicious device(s) could be used to disrupt an agent's normal operation or to embed a malicious code segment into it. As a result, using MAs can be deemed a safer option in closed networks where access is controlled. Additionally, well-known cryptographic methods, such as digital signatures, can be readily employed to reduce to some extent the inherent security risks, though some performance

degradations would be inevitable. However, this would be highly detrimental for network tasks requiring near real-time response, or in networks formed by devices with limited hardware resources, given that processing digital signatures entails additional memory availability and processing capabilities that increases power consumption.

1.3 A Historical Perspective of MASs

A myriad of MA-based solutions ranging from the network to the application layer were proposed throughout the literature, mostly from the mid-1990s to the early 2000s when the first investigations into MASs and their potential applicability took place [7]. A great deal of research focused on the high-level application aspects of this technology (e.g., [8]), whereas other efforts were aimed at a subject widely known as active networking [9]. Several MASs were proposed, as per the wide range of possible applications that MAs could support. The majority of these systems were built to run on the Java virtual machine (JVM). In fact, some contemporary Java-based platforms such as JXTA natively support code mobility in the form of MAs [10]. In this section, we summarize the most important aspects of their deployment, instead of engaging in a detailed review of individual MASs.

JVM-based MASs were highly popular for applications that required portability and support to accomplish complex tasks. This provided a programming and execution platform with unmatched flexibility that enabled MAS deployment to support a wide variety of applications [11,12]. Nonetheless, performance issues and security vulnerabilities were important concerns in applications that employed MAS for cellular phones and personal digital assistants sporting lightweight versions of the JVM, as mentioned before [13]. In addition, their overall effectiveness was sometimes compromised by distinct versions and flavors of the JVM installed in personal devices. This was not the problem of custom-built MAS, which provided a more homogeneous platform for implementing agent-based applications. However, these systems were less portable and were built for specific types of applications. It is also worth noting that some MA investigations did not involve any particular MAS and instead focused on other important aspects, such as migration strategy [14,15]. In fact, some early work advanced alternative MA schemes for wireless and mobile networks management without actually promoting a particular system [16]. In addition, some agent research targeted web-based applications for enhanced service discovery and composition (i.e., the combination of two or more services to form one single application) [17], though subsequent years saw a significant decrease in MA and MAS research activity. Nevertheless, enough interest remained on the subject to motivate sporadic research efforts on distinct networking technologies. In our case, the benefits of employing MA technology in wireless networks

applications are apparent because it has the potential to significantly reduce (1) bandwidth consumption incurred by network management overhead and (2) overall power consumption to help extend battery lifetime, as explained in Section 1.2.

One particular technology that exemplifies the benefits introduced by using MAS is WSNs. In brief, using MAs in WSNs may facilitate application programmability (also known as retasking) and collaborative signal and data processing. The MAS approach has a good potential to decrease bandwidth use (and its associated battery consumption), contrary to conventional WSN operations that rely heavily on the client–server communications model. WSNs have been the focus of much attention in the research community for nearly a decade [18–20], which is driven in part by a large number of theoretical and practical challenges. WSNs are intended to support specialized applications. However, it is tempting to try and employ a single WSN deployment to implement multiple applications due to the high cost of acquiring hundreds or even thousands of sensor nodes, if so necessitated by the application, or to cover a wide geographical area as proposed in Refs. [21,22]. The problem with this approach is that storing a multifunctional program to support diverse applications incurs significant memory utilization, which could be alleviated by employing the MAS approach. The advantage of using MAS here is that it enables the deployment of different types of agents to accomplish various tasks without the need to reprogram the WSN's nodes. It could be argued that the MAS approach is not much different from a multifunctional program for WSN applications. However, the multifunctional program approach will always be limited to the specific applications it has been built to support. However, the MAS approach provides the same functional value of a virtual machine, thus allowing MA deployment for applications that might not have been considered. In other words, it enables adaptation to unforeseen circumstances, which can be considered an inherent trait of environments monitored by WSN hardware.

Even though MASs were extensively studied by prominent researchers, to a certain extent it failed to fulfill the high expectations that many had placed on it [23]. Nowadays, MA research is still well positioned to enable contributions with a significant value, particularly in an area commonly known as *smart spaces* that has drawn considerable interest as of late [24–26]. This term is used by *pervasive computing* researchers when referring to intelligent environments enabled by consumer electronics and appliances with embedded devices whose behavior can vary as a result of their context awareness capabilities. Smart space applications are in turn enabled by *ambient intelligence*—a group of technologies assembled to provide an automated, personalized service or experience to one or more persons [27–29]. To this end, advances in electronics miniaturization technology enable the deployment of sensors and data processing devices with limited capabilities to create abstract representations of the surrounding environment, along with other devices

that can be used to provide a personalized, context-aware service. In other words, a smart space can be envisaged as a complex, interactive system that enacts one or more actions by using one or more of the surrounding devices as outputs, in accordance to a series of inputs provided by intelligently networked sensors. This is a relatively new technology with both growing interest in the research community and a vast potential for commercialization. However, a great deal of research needs to be conducted before experimental devices implementing this technology leave the industry and university labs to become consumer devices available to the general public. It is not hard to see that a smart space is actually a hybrid system formed by both hardware and software deployed in and around people in a distributed fashion. Therefore, we envision using MA technology as a prime candidate supporting complex, mobile Ambient Intelligence systems.

1.4 Applications of MAs in Wireless and Mobile Networks

As mentioned before, this chapter describes our latest MA technology advances in WSN and in mobile computing environments that employ radio frequency identification (RFID) technology. In Section 2, we look into the design issues encountered when engineering both MASs and applications in WSNs. We survey the particular example of video sensor networks (VSNs) as a WSN application with unique traits that can be favored by employing MAs. Interest in VSNs stems from the commercial application of video surveillance in deployment settings where intrusions are extremely rare, there is no supporting infrastructure (i.e., electricity), and there is no personnel to monitor the operation of the system in a permanent basis [30,31]. As a result, MAs can be employed to provide an autonomous, low-power mode of operation. We dissect the MAS design functionality into the following components: architecture, MA itinerary planning, middleware system design, and agent cooperation methodology. This classification spans low- and high-priority design issues and assists in the creation of an MAS that can be useful in an ample range of applications. We argue that flexible trade-offs between energy and delay can be reached, depending on the specific requirements set by the application.

In Section 3, we present the results of our investigation after putting theory into practice through *Wiseman*: a middleware system developed for the deployment and execution of compact MA scripts in WSNs. The architecture of Wiseman was designed as a simplified version of a much older agent system originally devised for the effective coordination of active networking processes (e.g., routing performed by mobile programs, instead of using hard-coded protocols and algorithms). To this end, we developed a simpler but effective interpreter of codes for embedding

in WSN devices characterized by having severe hardware constraints. Here, we detail the groundwork of our approach and its unique language constructs that minimize its operating cost. Wiseman was coded in the NesC language to produce a TinyOS ver. 1.x [32] binary image that spans 19KB of code and 3KB of SRAM. In addition, we elaborate on the distinct agent migration methodologies that the interpreter supports and present some performance evaluations regarding consumed bandwidth and internode hopping delay.

In Section 4, we advance a novel idea that relies on RFID technology as an enabler of diverse ambient intelligence applications. Given their small size and low cost characteristics, RFID tags can be easily embedded into consumer electronics devices in support of smart spaces. To this effect, RFID tags would readily enable the immediate identification of persons and objects to rapidly retrieve prestored, smart space configurations from a database and enact system personalization actions. However, there are problems that need to be sorted out with regard to the personalization of services and system configuration when person moves from one ambient intelligence environment into another. One of the major impediments for achieving this functionality is the way in which existing RFID systems function, which complicates their straightforward adaptation into real-world dynamics so as to fulfill application-specific requirements. We refer to this as identification-centric RFID system (IRS), which stores and forwards simple ID values that are referenced during a simple database lookup process to retrieve relevant information about the object that carries the corresponding ID tag. To address this problem, we promote advancing IRS into CRS—a code-centric RFID system, whereby actions are dynamically encoded and stored in RFID tags that possess improved memory capabilities. We argue that this innovative approach facilitates the operation of the implementing system to enact actions on demand in environments comprised by distinct objects, and under varying circumstances to achieve improved scalability. We present an E-healthcare management application that exemplifies the potential of our proposed CRS approach, and its importance in a smart space scenario. In Section 5, we conclude this chapter by summarizing our experiences using MA technology to solve distinct tasks in wireless networks and discussing possible improvements and areas of future research.

2. Architecting MA Applications and Systems for WSN

Recent innovations in the field of very large system integration (VLSI) facilitate the mass production of sensor devices that can be networked to enable the implementation of many distributed applications, which we introduced as a WSN. To this

effect, energy efficiency becomes one of the core design principles for all research done in this area, given that these WSN devices are generally powered by batteries. Moreover, many commercial WSN platforms are comprised by devices with severe memory and processing power limitations. These and other circumstances motivate research of flexible and improved schemes that allow WSNs to be reprogrammed when a new data collection/dissemination methodology is needed. At the same time, it is important to determine whether these new schemes are practicable from an engineering point of view to ensure that neither performance nor data integrity is compromised. Given that the primary role of WSNs is data collection, it follows that environmental monitoring and people/object surveillance comprise a good portion of their intended applications. In this section, we explore the intricacies that WSN researchers and engineers encounter when architecting an MA-based solution aimed at monitoring and surveillance applications. In particular, we study VSNs and target tracking (TT) applications, both of which provide a prime example of how MA technology can be employed.

2.1 Using WSNs for Image Retrieval

One of the most challenging tasks that can be observed in WSNs is image retrieval. To this end, both heavy image preprocessing load and high bandwidth usage have adverse consequences in the battery lifetime of sensor devices. In addition to this, such amount of data has the potential to clog the wireless link (s) when being forwarded to the VSN gateway for subsequent processing, as discussed in Section 1. As a result, a number of approaches have been promoted in the literature to retrieve images or video from WSNs [33, 34]. Some of the latest advancements in this area include Cyclops [35] and SensEye [36], among others, which are image processing testbeds especially developed for WSN use [37]. A quick survey of existing VSN schemes reveals that they rely on source coding and multipath forwarding schemes to achieve their task. Because of the bandwidth limitations encountered in commercial WSN platforms employing IEEE 802.15.4 radio technology that supports up to 250Kbits/s data rates, there will always be a hard threshold for the amount of data that these and other schemes will be able to transport from a sensor device to the gateway.

From the above discussion, we can see that MAs are well positioned as a plausible solution to leverage the performance of VSNs [38]. To this end, we recently proposed a solution whereby an agent is dispatched to the node that initially captures an image that needs to be analyzed and executes a preprocessing algorithm to obtain the picture's ROI, as depicted in Fig. 4. There, we can observe that the MA carries with it image segmentation and preanalysis codes to isolate a portion of the original image and perform a preliminary assessment procedure before forwarding the image

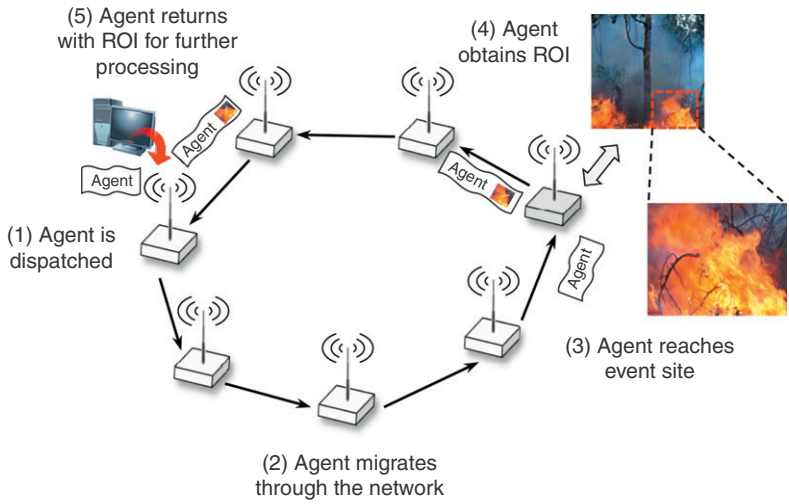

FIG. 4. Mobile agent-based image preprocessing in wireless sensor networks.

segment to the VSN's gateway. As a result, an otherwise large volume of image data originating at any region of the VSN can be significantly reduced to a much smaller, manageable one. The main feature introduced by the MA approach here is that if the conditions surrounding the monitored environment vary, then a new MA with an alternative image segmentation algorithm can be dispatched to the corresponding VSN devices to maintain the overall system's efficiency.

2.2 Target Tracking in WSNs

TT normally refers to the process whereby two or more devices work in conjunction to estimate the location of an object. Although there are instances in which a single device can be used to track an object, we are interested in the case where multiple devices are employed to reduce uncertainty about the object's position. A traditional TT application focuses on the design and implementation of the corresponding algorithms as a signal processing problem. To this regard, the (distributed) system's operation is expected to remain unchanged. However, if the circumstances surrounding the object being tracked change, it is possible that the performance of the current (static) approach could be compromised to the point of becoming ineffective. However, an MA approach would enable system operators to deploy distinct TT algorithms on demand to adapt to the prevailing

circumstances. It could be argued that a simple remote method invocation (RMI) mechanism meets the necessary requirements to implement this application, in which sensor nodes in the tracking region would maintain communications with a control unit outside of the WSN employed to orchestrate the task. However, it is clear that this approach would incur additional delay during the communications between the WSN control unit and the respective nodes.

Several solutions to the TT problem that employ MAs have been proposed in the literature. In Ref. [39], moving targets are tracked by MAs by employing a simple "trilateration" algorithm, and the result is periodically sent to a server that stores the targets' location. To achieve this, a node employs its own location measurement information and combines it with the readings obtained by two of its direct neighbors to produce a target location estimate. Figure 5A illustrates this approach, where the three-circled areas specify the possible positions of the target object based on the measurements taken by an equal number of MAs. One of these agents is referred to as the "mother agent," whereas the other two are referred to as "child agents" that are controlled by the mother agent to work cooperatively to obtain a better estimate of the target object's location. Figure 5A also shows that the mother agent temporarily stationed at node *A* dispatches the child agents to nodes *B* and *C* to help locate the target object. The child agent at *B* ends operations when the received signal strength at this node decays beyond a certain threshold, whereas node *D* receives a new child agent, as depicted in Fig. 5B. Later, the mother agent itself decides to migrate to node *C* to avoid losing track of the moving target. At this point, all child agents terminate, and new ones are dispatched to nodes *D* and *E*, as shown in Fig. 5C. From this example, it follows that multiple child agents can be deployed to track a moving object, and that their number can vary depending on the number of WSN nodes present in the monitored region. An alternative approach proposed in Ref. [40] also promotes dispatching an MA to track a moving object, as shown in Fig. 6. Upon migrating to a sensor node, the agent collects the necessary information to gradually

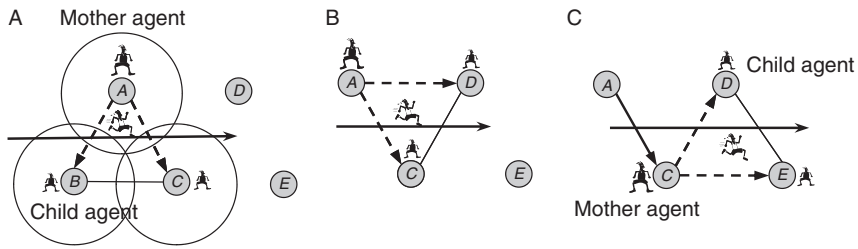

FIG. 5. Geographical target tracking using trilateration.

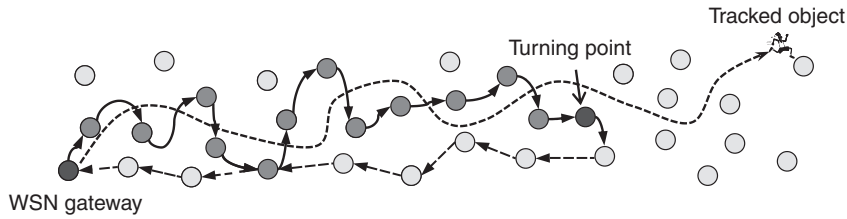

FIG. 6. Target tracking based on object recognition.

increase the accuracy with which an object is followed. When a certain threshold is met and the tracked object has been successfully recognized, the MA halts the tracking process and returns to the WSN gateway with the collected results.

2.3 Architecting Data Dissemination Schemes in WSN Using MAs

We classify the MA-based architectures as being either hierarchical or flat in accordance to the WSN's configuration. In a hierarchical architecture, the nodes' roles are different, whereas in a flat sensor network, the nodes' roles are either equal or very similar [41]. In this section, we will elaborate on each of these types of architectures and describe their pros and cons depending on the actual WSN application and configuration.

2.3.1 Hierarchical WSN Architecture

In general, the operation of an MA-based solution is simplified in hierarchical WSN deployments, such as in the clustered topology proposed in Ref. [42] that promotes intra- and intercluster hybrid methods. In the first case, every cluster head dispatches an MA that migrates to all cluster members collecting and combining data. Upon returning to its corresponding cluster-head, the MA sends the accrued data results back to the WSN's gateway for further processing. Conversely, the intercluster approach does not perform any MA operations inside the cluster. Instead, an MA follows an itinerary that takes it to all the cluster-heads, until the WSN gateway is reached. We can see that the intracluster method is intuitively favorable in cases where clusters are formed by many nodes, and the number of cluster-heads is reasonably small. On the contrary, the intercluster method is more efficient in clusters comprised by a smaller number of nodes. On the downside, network maintenance in hierarchical WSNs may involve significant signaling

overhead. Evidently, this problem can be solved by resorting to a flat WSN configuration that may be appropriate for a broad variety of sensor applications.

2.3.2 Flat WSN Architecture

In Ref. [43], researchers introduced the MA-based distributed sensor network (MADSN) approach for using in both hierarchical and flat WSNs. Here, the WSN gateway dispatches a certain number of MAs that gather data in a target region. For this, it is assumed that source nodes are close to the WSN gateway; however, an MA can traverse a number of hops before reaching the first source node, if needed. One caveat is that dispatching MAs from the gateways to the source nodes can incur significant bandwidth, thus defeating the benefits obtained by using MAs. To address this side effect, we proposed a novel approach that we coin MA-based WSN or MAWSN [44]. In our approach, a single "mother" MA is dispatched by the WSN gateway to the target area and remains temporarily stationed at a designated node therein waiting for a trigger event or command. This MA carries the codes that perform the desired action within that region. When triggered, the mother MA dispatches one or more "child" agents that carry out one or more tasks. Child agents separately visit a set of data source nodes to collect and aggregate data. Then, the accrued data is either sent back to the mother MA or directly to the WSN gateway. Fresh batches of child MAs can be dispatched periodically by the mother MA in accordance to the strategy being implemented. In addition, several child MAs can also be concurrently launched by the mother MA to perform one or more tasks in parallel to reduce latency. From our description of MAWSN's operation, it follows that using MAs enables the reduction of data traffic at three levels. In the first level, unprocessed data can be reduced at the source node level as stipulated by child MAs, so that only digested information is sent to the WSN gateway. In the second level, data redundancy can be eliminated by having child MAs visit the source nodes that produced the highly correlated data. Finally, a mother MA can further aggregate data pooled by child MAs upon returning to the mother MA's location.

2.4 Agent Migration Itinerary Planning

The migration itinerary of an MA is defined as the path followed when hoping through the underlying network as it performs a given task. In our case, planning the itinerary of an MA involves two main steps that can be realized either by the WSN gateway or autonomously by the MA: (1) defining the subset of nodes to visit and (2) defining the actual path to follow to preserve bandwidth. This last aspect can have an important effect on the overall energy consumption of the WSN nodes involved. This is a well- and long-known problem in computer networks known as

the traveling salesman problem [45]. It has also been shown that finding an optimal sequence for visiting the respective nodes of such a path is a nondeterministic task that can be solved in polynomial time of the factorial of the number of nodes to visit; that is, it is an NP-complete problem. If the visiting path is fixed, then the MA migration itinerary is determined by the WSN gateway before dispatching the MA. Conversely, if the visiting path is dynamic (variable), then the MA autonomously decides what WSN nodes to visit, depending on the current network conditions or on its task execution progress. However, it is also possible to implement a hybrid approach, whereby a preliminary WSN node-visiting path is decided by the gateway, whereas last minute changes can also be performed appropriate by the MA as deemed suitable. We now elaborate on the particularities of these approaches.

2.4.1 Planning a Static MA Itinerary

This approach relies on current global network information to derive a possible migration path before an MA is dispatched. Two methods that address this problem were presented in Ref. [46]: local closest first (LCF) and global closest first (GCF), both of which assume that out of the nodes to be visited, the executing one is the closest to the gateway. For this reason, LCF first searches for the node that is closest to the current node, whereas GCF does so for the node closest to the gateway. Alternative solutions also exist. For instance, a genetic algorithm is presented in Ref. [47] to devise MA itineraries for WSNs, which assumes that each sensor node can be visited only once to reduce the search space. This solution achieves global optimization, though it is a computationally heavy one whose actual suitability in resource-constrained nodes is debatable.

2.4.2 Planning a Dynamic MA Itinerary

Our previous descriptions of static MA itinerary planning solutions reveal that they may be unsuitable for WSNs that experience varying conditions if the global information stored at the gateway becomes outdated in the presence of continuous changes in the underlying environment. On the contrary, dynamic itinerary planning enables MAs to determine which node to visit as it hops through its migration path. To achieve this, trade-offs between the migration plan change costs and possible efficiency degradations should be taken into account. For instance, researchers in Ref. [40] promote a dynamic planning method that achieves progressive fusion accuracy without incurring excessive costs. To this end, the dynamic itinerary planning approach ensures that the visited sensor nodes (1) have enough battery power energy, (2) require minimum energy consumption for the MAs migration, and (3) yield significant information gain. As discussed before, one of the objectives of

the MA should be visiting sensor nodes that reduce uncertainty to shorten the migration path, reduce bandwidth usage, and decrease task completion delay.

2.4.3 Planning a Hybrid MA Itinerary

This approach selects a static set of sensor nodes that should be visited, leaving the migration sequence open to changes. A hybrid itinerary planning scheme coined as mobile agent-based directed diffusion (MADD) is proposed in Ref. [38]. In it, exploratory packets are sent to the WSN's gateway as soon as sources in a certain region detect an event of interest. Then, the gateway defines a cost-effective migration path for the MA as it visits a WSN node subset.

2.5 Middleware Layer Design

MASs are frequently implemented as middleware that bonds the underlying operating system with high-level software components, which in our case has to take into account significant limitations and technical challenges inherent to WSNs. MAS middleware (MASM) provides a platform that enables MAs to realize application-specific tasks and rapid application development in WSNs. However, a trade-off arises by the need to produce a drastically simple system that consumes the least possible amount of resources, without sacrificing functionalities that are key to ensuring a successful WSN deployment. One way to achieve this is by creating a custom programming language with the necessary high-level constructs that embody repetitive sequences of low-level tasks that are routinely encountered. This approach reduces agent codes' size, simplifies the structure of the MA interpreter, and promotes using local function libraries at the sensor nodes. As a result, programmers relieve MAs from having to carry the codes to perform low-level tasks and reduce programs' sizes. The degree to which this approach can actually be implemented depends on several factors. For instance, if WSN nodes are resource constrained, then the number of library functions locally available might be limited too.

The previous discussion evidences the need to create custom MASMs whose architecture depends on the applications they are intended to support, as well as on the underlying hardware and the environment being monitored. An MASM could be designed to support either coarse- (high-level) or fine-grained (low-level) MAs employed for distributed task coordination or local data processing. For instance, SensorWare [48] employs high-level language constructs, whereby a single expression realizes multiple low-level tasks in WSNs. Conversely, Agilla [49] resorts to low-level codes that resemble assembly programming mnemonics. In general, fine-grained constructs yield lengthier programs that control task execution in greater detail, thereby requiring a more intricate MASM that consumes more memory at the

local sensor nodes. However, an MASM architected under the coarse-grain approach yields shorter MAs that resemble Macroprograms—condensed instructions mapped to a lengthier sequence of detailed commands—that promote using the nodes' local function library. Consequently, the complexity of the MASM is reduced along with the MA's size. On the downside, this approach sacrifices detailed task control.

2.6 Multiple Agent Cooperation

MAs can be deployed to operate independently or collaboratively to realize one or more tasks. In fact, previous research has shown that active agent collaboration can decrease energy consumption in the WSN as a whole [39,48,49]. The reason for this is that parallelism is exploited more efficiently by sharing information that other agents can employ to reduce task execution delay and bandwidth due to repeated hoping through the network during the data collection process. Thus, it follows that fewer MA hops in its migration path leads to a reduced battery consumption rate in a shorter time frame, as agents finish their task quicker. The type of information that agents exchange can be simple or abstract. In addition, these data can be continuously modified by other agents in a time-decoupled manner until a certain outcome is found, which can be subsequently used in other decision processes, including, but not restricted to, the course of the current WSN task. To this effect, the tuple-space mechanism remains widely popular, whereby MAs communicate indirectly by saving typified data at the respective WSN nodes, contrary to directing messages to each other, as shown in the example of Fig. 7. Here, we see that *Agent 1* visits *Node n* to change the contents of a tuple-space entry, which is later updated by another MA. For this communications mechanism to work, it is necessary that all agents know how to access and store data in the corresponding tuple-space, so that no information is corrupted. Therefore, the semantic meaning of the tuple-space needs to be predefined. It is straightforward to see that this approach favors applications with loose timing requirements in which the significance of the information stored at the WSN nodes deteriorates slowly in comparison to the occurrence rate of the events being

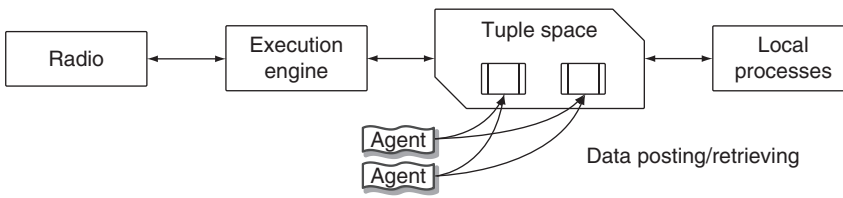

FIG. 7. Multiple agent cooperation using tuple-spaces.

monitored. Alternatively, direct agent communications might be the best solution under a tight timing requirement, as in the case of applications that require real-time information processing. An instance of this could be a WSN used for intrusion detection based on partial image evaluation, as explained before. Here, the WSN gateway could deploy agents that seek evidence of a certain event as captured by a sensor camera. Upon finding the piece of evidence sought for, agents could directly send signals indicating the current outcome, which would serve as an indication to the other agents to either continue or halt the process. However, it is always possible to employ more than one communications mechanism to accomplish a certain task.

2.7 Summary

An effective MAS design is crucial for overcoming the limitations found in alternative methods to solve diverse WSNs problems, where bandwidth and batter power conservation are fundamental. This section introduced important aspects that are relevant to MAS deployment in WSNs. Applications, system architecture, migration itinerary planning, middleware design, and MA cooperation issues were discussed. We stressed the notion that important bandwidth savings can be achieved by locally processing sensor data as specified by the codes carried by an MA. However, performing efficient MASM design is imperative to support the intended WSN application to solve a particular kind of problem. Thus, it is important that the MASM includes custom features to provide enhanced efficiency. We also discussed aspects directly related to MA cooperation and the corresponding communications mechanisms, which are ultimately a direct influence of the intended objective to achieve maximum performance. Consequently, the benefits that can be obtained by applying MA technology in a WSN, or even in other types of networks, also depend to a good extent on the skill of the programmer that creates the respective codes. Finally, there is an important qualitative aspect of using MAS: they provide an effective means to deploying alternative solutions for dealing with unexpected problems or unforeseen situations, which are much harder to tackle when message-passing schemes are used. In Section 3, we will present our original work in the area of MASMs, which takes into account all the provisions and discussions that we have presented.

3. Programmable Tasking of WSN

3.1 Agents in WSN

In the previous section, we described the importance of providing WSNs with the ability to be seamlessly retasked as required by the underlying conditions of the environment being monitored. We mentioned the shortcomings encountered by

employing classical message-passing schemes and vied for using the MASM approach to solve this issue. In recent years, researchers designed and implemented programmable-tasking schemes in WSNs by embedding virtual machines that systematically process and forward highly compact codes between nodes. This reduces packet transmissions and enables WSN nodes to effectively switch from one mode of operation into another. Virtual machines entered the WSN realm recently after WSNs became a popular research topic. The first systems proposed were Maté [50], Impala [51], and Deluge [52] and were promptly followed by SensorWare [48], SmartMessages [53], and Agilla [49]. Initial approaches were specifically designed to work on devices with severe resource constraints, as explained before, whereas the latter approaches introduced architectures for interpretable codes with more functionalities than their predecessors.

Prior to these advances, extensive investigations had already taken place in the late 1990s and early 2000s to gauge the advantages of deploying mobile codes in WSNs, contrary to implementing traditional message-passing protocols [6]. An important finding was that MASs indeed are able to produce better performance results than their conventional message-passing counterpart in WSNs, subject to the migration strategy employed to reducing the distributed data processing delay. In addition, execution and forwarding overhead, which depend to a good extent on the MAS' architecture, also play an important role. Consequently, we argue that MASM design is crucial for efficiently supporting the agents' operation. In other words, whereas MASs for WSN need to incorporate the necessary mechanisms that minimize bandwidth and power consumption, they warrant a design for dealing with the peculiarities of each individual WSN application, contrary to creating general-purpose solutions.

3.2 The *Wiseman* Approach

Unlike existing schemes that promote general-purpose approaches, we advance an MASM that implements specific network-level features for simplifying distributed task coordination in WSNs. Our proposed MASM, wireless sensors employing mobile agents (*Wiseman*), is characterized by the following features: (1) its architecture supports distinct migration strategies and provides a flexible code execution flow; (2) it follows a text-based, code scripting scheme, whereby compact MAs can be efficiently deployed to implement a number of actions, or to perform WSN maintenance tasks as needed; and (3) it implements an execution model whereby self-depleting text strings can be progressively eliminated to expedite their processing time and reduce the incurred transmission bandwidth and forwarding delay.

The ingenuity of embedded software engineers has been put to the test to create MASM that is amenable to hardware-constrained WSN devices to achieve the desired balance between system functionality and performance. To this end, there are two paradigms that can be followed. One promotes a fine-grained approach, whereby simple, mnemonic-like language constructs resembling assembly programming codes are used to specify a detailed flow of operations that an MA will perform (e.g., *<CLR REG0>*, *<JMPTO 0x04>*, etc.). The other paradigm follows a macroprogramming scheme, whereby course-grained constructs describing compound operations are employed to describe the order in which task will be executed (e.g., *<Run Task X>*, *<Run Task Y>*, *<IfSuccess END*, *Else Run Task Z>*). Evidently, there are both advantages and disadvantages to using one or the other. For the first, the MAS interpreter would deal with more compact agents invoking locally available libraries at the nodes (e.g., *Run Task X/Y/Z*) but would also encounter limited execution thread flexibility. In the other approach, a detailed operation flow sequence is possible at the expense of larger programs, even if a byte-code scheme is implemented to reduce their size. Ultimately, MASM engineers need to define the right degree of granularity required for implementing MAs, as dictated by the intended WSN application(s). However, it is apparent that the prevailing conditions of WSN hardware favor the implementation of coarse-grained approaches, combined with limited fine-grained functionalities that allow MAs to test for certain conditions to veer the flow of the current process as needed. In addition to this, a WSN is expected to operate in the same manner for a certain period of time after it has been (re)tasked, meaning that a number of subtasks execute repetitively until the WSN is reprogrammed into a different operation mode. In fact, contemporary WSN devices provide some limited memory space intended for logging sensor data, which can be conceivably partitioned to store additional data-processing algorithms, or "dormant" MAs that can be activated on demand. Regardless of this, the incentive behind employing MAs weakens if the tasks they are meant to perform become deterministic in relation to the WSN's operations. However, WSNs deployed in environments where a higher degree of uncertainty prevails benefit more from using MASM that enables flexible programmability, which enables them to adapt to unforeseen circumstances.

A caveat of the previous rationale is that, because WSN technology is still deemed as being relatively recent, it becomes tempting to assume that contemporary MASMs need to be designed from the ground up, and that older systems designs are altogether inappropriate (e.g., those based on the JVM). However, we make note of the existence of one particular system that dates back to the mid-1980s, whose design was based on the premise of deploying mobile codes in a time where message passing was a vastly predominant communications mechanism. The *wave* system [54,55] promoted one of the earliest code mobility paradigms in wired computer networks specifically conceived for the efficient coordination of distributed tasks [56].

It could be argued that looking into the past for earlier technologies that could be adapted to contemporary needs is counterintuitive. However, we note that limitations found in earlier computer network systems (e.g., processing power and bandwidth) still exist in some contemporary systems. This clearly is the case of WSNs formed by devices possessing 8-bit microprocessors with 128 KB of RAM that communicate at 250 Kbits/s. Therefore, we considered worthwhile revisiting previously explored concepts and technologies to understand how these issues were originally addressed, and whether they could be adapted to solve current WSN issues (i.e., contrary to attempting to reinvent a solution for a reasonably comparable problem). After a careful analysis, we observed that the wave's language constructs originally devised for bandwidth-constrained network systems could be adapted for WSN use. Consequently, Wiseman inherits many of the traits already seen in wave:

1. The MAs' primary task is geared toward distributed process coordination of a WSN by decoupling the data processing element, and instead promoting the execution of local algorithms, as explained before.
2. The size of MAs' scripts can be shortened by defining the language constructs that adequately describe the process coordination methodology that a WSN operator/engineer wishes to deploy.
3. As a result of the two previous aspects, Wiseman's interpreter architecture can be further simplified, yielding a smaller memory footprint that is amenable to WSN hardware.
4. MA forwarding delays are reduced by transmitting compact MAs' codes, which also promotes bandwidth conservation.

3.3 Architecture and Code Processing Flow of Wiseman MAs

Wiseman was designed to minimize program memory footprint. Therefore, it implements only four components: an incoming MA queue, a parser, a processor, and an MA dispatcher (see Fig. 8). In addition, the Engine and Session Warden blocks are incorporated to assist the other components. The incoming queue temporarily stores agents after being received from the wireless interface and are subsequently parsed for immediate processing once the currently executing MA finishes. It is also possible to inject agents that are stored in the node's local library, as explained before. In this case, an MA arriving from another node instructs the interpreter to fetch a particular MA for immediate queuing and processing. (Because of the processing limitations prevalent in WSN hardware, Wiseman is not designed to execute in a multithreaded manner.) Once an MA is removed from the incoming queue, the parser tokenizes individual MA codes by dividing them into two parts

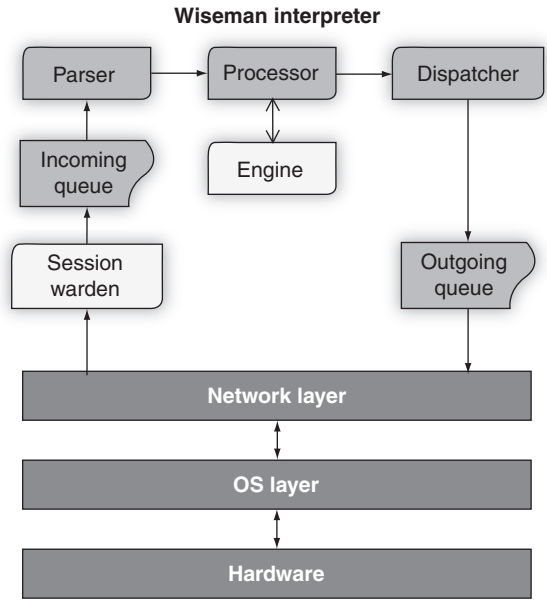

FIG. 8. Architecture of the Wiseman interpreter.

that we refer to as *head* and *tail*. The head is simply an indivisible code segment that cannot be further split, which is sent to the processor for immediate execution. The tail, however, is made of the rest of the MA's codes, which are also tokenized for subsequent processing, as needed. The processor almost always relies on the Engine as a helper block that performs certain simple tasks, including the transformation and/or adaptation of data.

The parser regains control of the MA's processing task as soon as the head finishes executing, at which point the tail is subsequently tokenized to obtain the next indivisible segment that becomes the new head, whereas the remaining codes become the new (shrunk) tail. At this point, the new head is passed onto the processor for immediate execution as before. Figure 9 illustrates the MA execution sequence of the Wiseman interpreter. We note that the processor may stop executing an MA if (1) the current code segment's outcome is unsuccessful, (2) the MA requires being forwarded to another node as indicated by the current instruction, or (3) an explicit MA termination is specified. The first case implies that the remainder of the MA being executed must be discarded if the current condition being evaluated results in a FALSE outcome. The second case implies that a hop operation was found, and so the tail of the MA is immediately passed onto the

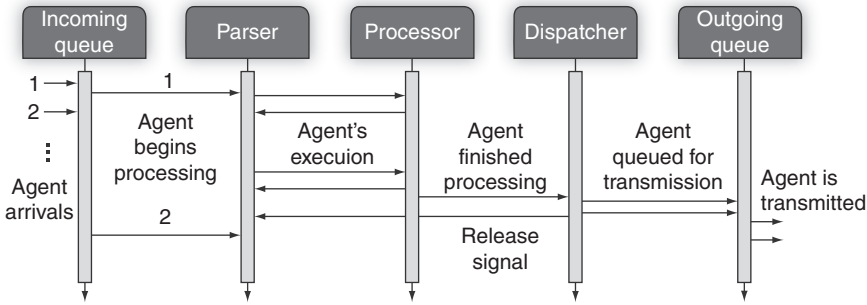

FIG. 9. Execution sequence of Wiseman agents.

dispatcher for its transmission to another WSN node. Finally, the third case is usually incorporated by the MA programmer as a fail-safe provision to avoid further propagation. Aspects pertaining to the Session Warden of the interpreter will be detailed later in this section. Figure 8 illustrates Wiseman's interpreter architecture.

3.4 Wiseman's Instruction Set and Language Constructs

Most of Wiseman's language constructs are derived directly from wave, although they have been further condensed to reduce program size and forwarding delay. Unlike other MASM approaches for WSNs that resort to bytecode-type agents, Wiseman agents are written as text scripts, making it flexible to dynamically incorporate modifications to their program structure. Therefore, Wiseman MAs can morph dynamically after they have been dispatched from the WSN's gateway. Wiseman's language constructs are made of variables, rules, operators, and delimiters, as explained next.

3.4.1 Variables

Wiseman requires that enough processor memory be reserved to store floating point type variables, known as *numeric*. The memory content of this type of variable is accessible through the letter *N* (e.g., *N0*, *N4*, etc.). Similarly, *character* variables require that memory be reserved to store single characters that can be referenced by the letter *C* (e.g., *C3*, *C5*, etc.). Access to these variable types is public, implying that all MAs that reach the local node have read/write privileges on them. Unlike Agilla, Wiseman does not implement the functionalities to perform operations on remote

nodes' variables. Therefore, local operations on numeric and character variables have no effect on other nodes' variables. However, it is straightforward in Wiseman to spawn a simple MA that can hop to a neighboring node to update a local variable, if needed. In addition to numeric variables, MAs may carry with them *mobile* variables, whose purpose is analogous to that of private variables in object-oriented programming, and so they are inaccessible to other MAs. The contents for each of these variables is accessible through the letter *M* (e.g., *M7*, *M8*, etc.), which are semantically similar to private variables in object-oriented programming. All nodes reserve enough memory to store the contents of mobile variables when an MA is fetched from the incoming queue for immediate execution. When an MA finishes executing and is either dispatched or terminated, the reserved memory space is overwritten to store mobile variables carried by another MA being brought into execution. The WSN engineer must then ensure that the semantic meaning of all three variable types is kept throughout the network. For instance, the contents of variable *N1* at node *m* should have the same meaning as that of variable *N1* at node *n*. The *clipboard* variable is also provided by the interpreter to temporarily store data for diverse operations, and its contents can be accessed through the letter *B*. Finally, the interpreter defines three additional *environmental* variables that store data regarding the executing environment. The first of these is a read-only variable that holds the local node's identification number (ID) (i.e., 1, 2, etc.) and is defined as the *identity* variable *I*. The next is the *predecessor* variable *P*, which is populated with the identity number of the node that an MA arrived from as it is loaded into the parser for processing. The third and last of the environmental variables are designated as the *link* variable *L*, which is similar to *P* but is used to store a virtual label identifier of the link that the MA employed for hopping into the local node, as discussed later in the text. It follows that the contents of both the predecessor and the link variables are overwritten with every MA that enters the execution space, and that each MA carries this information that is provided by the node that last dispatched it.

3.4.2 Operators

Wiseman implements a number of operators that are available to the programmer. The first type is the arithmetic operator type (i.e., +, −, *, /, and =) that is used to update the contents of numeric and mobile variables. It follows that arithmetic operations are not applicable to character variables. Other general-purpose operators are also available to evaluate different conditions (i.e., <, <=, ==, =>, >, and !=). However, Wiseman defines other system-specific operators that are unique to our approach. The *hop* operator # is employed to indicate that an MA's tail is to be forwarded to a node with the ID specified on the right-hand side of the character

(e.g., *#5* indicates a hop to node 5). Alternatively, a virtual link identifier (i.e., a label) can be employed to refer to a subset of neighboring nodes to which an MA can be multicast. For example, a virtual link labeled as *d* can be assigned to odd-numbered nodes in the node set [1 . . . 5] (i.e., 1, 3, 5), and a label *e* can be assigned to even-numbered nodes (i.e., 2, 4). In such case, multicast hopping is instructed by placing the corresponding label on the left-hand side of the operator (e.g., *d#*). Here, the Wiseman interpreter automatically clones the agent to forward a copy to each node in the corresponding subset. Then, agents resume executing at the code that follows the hop operation immediately after reaching the destination node. This simplistic form of strong mobility eliminates the need for forwarding a program counter and execution state variables with the MA. Alternatively, the *broadcast* operator *@* can be used to forward a short agent to all neighboring nodes. Wiseman also provides the *query* operator *?* that allows an MA to check whether a given label has already been defined at the local node by an agent that visited the node earlier in the process (e.g., *e?*). As mentioned before, MAs can inject locally stored codes retrieved from the local library by employing the *code injection* operator ∧. To this effect, the MA programmer must know in advance the identifier needed to reference the respective agent's codes. For example, the code ∧*7* fetches a local agent labeled with the number 7 for immediate insertion into the incoming queue block and subsequent execution. Similar to this is the *execution* operator *$*, through which an MA invokes a local function or algorithm that is not in the form of an MA. One operand is used to indicate which function is being called, and the other operand represents a single parameter being passed to the functions, which appear in the left- and right-hand side of the operator, respectively. To date, functionalities to switch on/off LEDs are available, to retrieve temperature/light sensors readings, and to retune the frequency channels of the local radio circuit. Finally, the *halt* operator *!* is employed to indicate explicit termination of the MA currently executing.

3.4.3 Rules

Wiseman implements three methods to control the execution flow of MAs' codes. The first and most often used is the *repeat* rule *R*. This rule is provided as a simple loop, and so the interpreter executes the codes embraced by curly brackets cyclically. To this end, the codes inside the repeat rule are copied and reinserted before the whole construct (i.e., *R{. . .}* yields *. . .;R{. . .}*). Consequently, the codes within the *repeat* rule execute until a certain condition evaluation fails. In addition to this, the *And*/*Or* rules control an MA's execution flow by verifying the outcome of the delimited codes within square brackets. For example, an Or rule construct *O [<code1>;<code2>;<code3>]* executes until the outcome of one of these segments yields a TRUE outcome. However, if the last code to be evaluated (e.g.,

<*code3*>) yields a FALSE outcome, then the MA terminates. It follows that the "And" rule operates under a similar premise, though all the delimited codes must return TRUE if the agent is to continue executing.

3.5 Wiseman's Agent Migration Methodologies

3.5.1 Explicit Path Hopping

One intrinsic feature of existing MASM approaches is that they all provide the means for explicitly defining the migration path of MAs through the WSN. There are two implications here. The first is that either the WSN gateway or a server/coordinator beyond the gateway must have knowledge of the network's topology for defining the corresponding hopping sequence, and the second is that the node-visiting sequence for the MA cannot/should not be revised once defined. Though this migration method is the easiest to implement, it hinders the ability of an MA to dynamically react to changes that occur rapidly or unpredictably in the WSN. As a result, an MA might visit a node that is no longer a relevant part of a subset of nodes that observed a certain event or, worse yet, an MA might be unable to follow the initial migration plan. Therefore, this approach is most useful if it is anticipated that the conditions of the underlying environment being monitored by the WSN will remain stable [38]. The hop operator available in Wiseman enables explicit path migration by specifically indicating the destination node on the right-hand side of the operator. Therefore, a sequence of hop operators with their respective destination creates the explicit sequence of nodes to visit (e.g., *#3;...;#7;...;#4;...;#8;...*). To this effect, it becomes the responsibility of the WSN engineer to ensure that the MA will perform the required task at each respective node.

3.5.2 Variable-Target Hopping

Wiseman provides the means to implement a migration scheme whereby the MA's path can be modified as needed after it has been dispatched from the gateway. In this case, either mobile or numeric variables holding the identity of target node can be specified on the right-hand side of the hop operator. Therefore, changing the contents of the corresponding variable has a direct effect on the outcome of the hop operation. For example, the operation *#N4* specifies that an MA will hop to the node the ID of which has been previously stored in *N4*. It follows that there must be a separate process, either carried out by another MA or executed by a local function that is in charge of updating the contents of *N4*. However, if the hop operation employs a mobile variable (e.g., *#M7*), then it becomes the task of the corresponding MA that carries this variable to update its contents accordingly as it migrates

between WSN nodes. To this effect, either direct assignment or regular arithmetic operations can be employed. For instance, if the variable *N4* holds the number 2, then the operation *N4+1* reassigns the number 3 to *N4*, and so *#N4* would forward the MA to node 3. An identical process applies to mobile variables used as operands.

3.5.3 Labeled Path Hopping

A disadvantage of the previous agent migration methods is that they are unable to forward multiple copies of an MA to different nodes as specified in a single hop operation so as to multicast the MA. Wiseman addresses this issue by assigning the ID numbers of one or more nodes to a letter that becomes a label, as explained previously. When the interpreter finds a character on the left-hand side of the hop operation, it forwards a copy of the MA to each node that belongs to the subset assigned to the corresponding label (e.g., *a#*). From this, we infer that (1) a separate process is needed to label such virtual links and (2) the node subset assigned to a specific label may differ from node to node. For instance, node *9* may have nodes *5* through *8* as neighbors, from which two labeled subsets need to be formed: one for the even-numbered nodes and one for the odd-numbered nodes. To achieve this, the codes *L=s;#6;#P;#8* and *L=g;#5;#P;#7* lead to the creation of the corresponding labeled paths assigned to letters *s* and *g*, respectively, by employing the environmental variables *L* and *P*. Then, any MA that subsequently traverses either of these paths can employ the codes *s#* or *g#* to reach at once nodes *6* and *8*, or *5* and *7*, respectively. From here, it can be inferred that MAs traversing these paths do not need to know in advance the IDs of the nodes they will visit—the sole use of a label suffices. Similarly, this hopping method can be implemented by explicitly defining labels or by assigning labels to character variables (e.g., *C1#*).

3.6 Middleware Implementation of Wiseman

We now describe the most important aspects of the implementation of the Wiseman interpreter in actual WSN devices. After its initial coding and verification in the OMNeT++ discrete event simulator [57], Wiseman was ported to the NesC language to create a binary image of the interpreter that runs on top of TinyOS ver. 1.x that we installed and tested on the Micaz sensor platform [58]. The Micaz sensor nodes remain widely popular in academia to evaluate a number of WSN schemes ranging from the MAC layer to the application layer, as they provide a perfect example of a resource-limited wireless sensor node with merely 128 KB of program memory and 4 KB of volatile SRAM memory. Wiseman's implementation follows the architecture shown in Fig. 8, whose program requires almost 2400 NesC lines that yield a 19-KB binary image, and consumes around 3 KB of SRAM space for holding MAs that span at most

170 bytes. Given the lack of a transport layer in TinyOS ver. 1.x, Wiseman implements a simple scheme for MA segmentation and reassembly implemented by the Session Warden module. To this end, MAs spanning more than the default 29-byte user space available in TinyOS packets are segmented into smaller pieces that are individually forwarded onto the next node as specified by a hop command. At the destination, the segmented MA is sequentially reassembled before being injected into the incoming agent queue. The header of a Wiseman agent has the following fields, some of which can be used to recover from any possible error that may occur in the wireless channel during an MA forwarding session:

1. *Segment number*: employed for identifying any given MA segment i out of a total n, whose value is gradually incremented for each segment being sent.
2. *Last segment*: used to indicate that the current segment is the last of a total of n.
3. *Source ID*: used by the destination node to populate the environmental variable P.
4. *Session number*: a pseudo-random number assigned to the current MA forwarding process.

An MA forwarding session initiates when the sender issues a *request-to-send* (RTS) packet to the destination node. This is done to ensure that this node is not involved in active communications with another node and that it is ready for being engaged with. If this is the case, the destination node sends a *clear-to-send* (CTS) packet to acknowledge the RTS signal back to the source node. At this point, segment numbers are employed in subsequent packets being transmitted so that any discrepancy of the expected segment number in the current session is dealt with accordingly. This is a well-known mechanism that is not natively implemented in TinyOS ver. 1.x when forwarding packets, and so the programmer needs to implement it to ensure that packets are correctly forwarded. When the last segment is successfully received, both the sender and the receiver nodes reset their current session values in preparation for a future forwarding session. Deadlocks are avoided by employing timeouts that trigger the corresponding event after 300 ms at the sender when no CTS acknowledgement is received from the destination. Similarly, a maximum of three retransmissions are attempted for any given MA segment. The maximum number of MAs that the incoming/outgoing queues support in the current Wiseman implementation is 3 and 5, respectively, which is a limitation of the Micaz SRAM memory space. However, a TelosB mote that possesses 10 KB of SRAM memory can support more and larger agents that can be held at the queues. It follows that a sensor mote will not issue a CTS signal back to the sender if the incoming queue is full. We also note that no routing service is implemented in the WSN. However, labeled paths can be readily created by MAs to accomplish the same result.

3.7 Case Study: Early Forest Fire Detection

3.7.1 Experiment's Rationale

We conducted experiments that exemplify the usefulness of employing MAs in an environment where the underlying conditions can change unexpectedly. In this case, we emulate Wiseman's support in a forest fire detection application, which has the following characteristics:

a. The underlying conditions of the deployment setting may change rapidly and unpredictably.
b. A WSN can be employed to assess the likelihood of one or more sectors of the monitored area experiencing an uncontrolled fire.
c. Given the vast area of the deployment setting, the WSN's will likely be comprised of several hundreds of sensor nodes, depending on the desired monitoring granularity.
d. Sensor node battery replacement and/or reprogramming would be highly impractical. Therefore, a reprogrammable, energy-efficient design becomes a top priority.
e. Only two sensor types are needed: one for humidity and one for temperature.

We focus on early fire prevention as the main WSN task, instead of fire tracking [59], even though this and other applications relevant to the deployment setting type can be readily implemented in Wiseman. Figure 10 illustrates an example of how a WSN can be deployed in a large forest area partitioned into three sectors (i.e., *A*–*C*). The early forest fire detection application is implemented in two steps. In the first step, MAs are dispatched from the WSN's gateway to collect temperature readings at the sectors being monitored. To simplify the process, cluster-head nodes can be

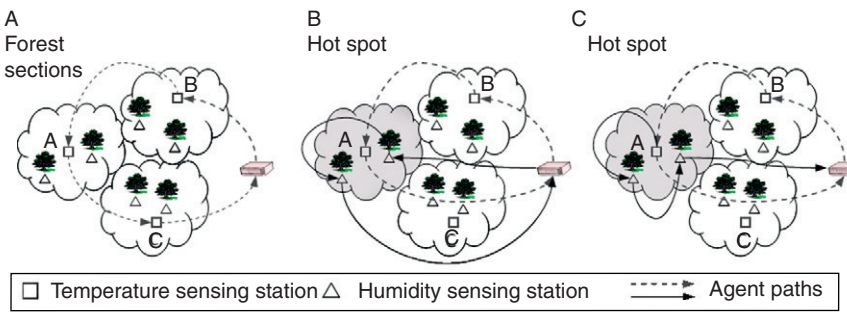

FIG. 10. A sample deployment setting for forest fire monitoring using Wiseman agents.

predesignated to pool temperature readings from individual sensor nodes dispersed in each sector. If an MA finds that temperature readings of one or more sensor nodes exceed a preset threshold, then the second step of the monitoring process collects humidity readings at the corresponding sector. If the MA deems that the humidity readings are below a preset threshold, then a warning signal is sent back to the monitoring center.

The previous rationale for this application indicates that MAs only have to migrate through a predefined path under normal circumstances, as illustrated in Fig. 10A. However, under adverse environmental circumstances, the secondary assessment process might be necessary, which can follow one of three possible approaches as shown in Fig. 10B and C:

1. *Approach A:* The MA returns to the WSN gateway immediately after the anomaly is found, which in turn dispatches another MA employed for obtaining humidity readings.
2. *Approach B:* A single MA is used for monitoring both temperature and humidity readings, implying that the MA implements both codes that realize two migration strategies.
3. *Approach C:* The MA operates as in the first approach; however, the secondary MA is dispatched from the respective cluster-head that coordinates the monitoring processes of the sector that observed the corresponding temperature anomaly, instead of from the gateway.

3.7.2 Case Study Setup

Our experiments implement four WSN topologies that emulate the deployment scenario explained in the previous section. Due to the limited number of sensor devices at hand, we programmed a few nodes to assume the role of cluster-heads, while a few others were programmed as regular sensor nodes assigned to a fictitious Sector 2, where the readings are made. In this sample setting, node 2 is the cluster-head of the forest area monitored by humidity sensor nodes 6–9, as seen in Fig. 11. The MA dispatched in experiments 1 and 2 migrates only through three cluster-heads, whereas in experiments 3 and 4, five cluster-heads are used. However, experiments 1 and 3 have four sensor nodes in the sensor node migration path, and experiments 2 and 4 employ six sensor nodes as part of Sector 2.

Table I illustrates the MA codes that implement approaches A, B, and C explained before for the forest monitoring application. The Micaz green LEDs are employed as visual aids to verify when the corresponding MA travels through the labeled paths (by means of the code "*l*$*n*" that toggles the corresponding LED—*l* for LED, and *n* for green), and the clipboard variable *B* holds the value 45 as the fictitious

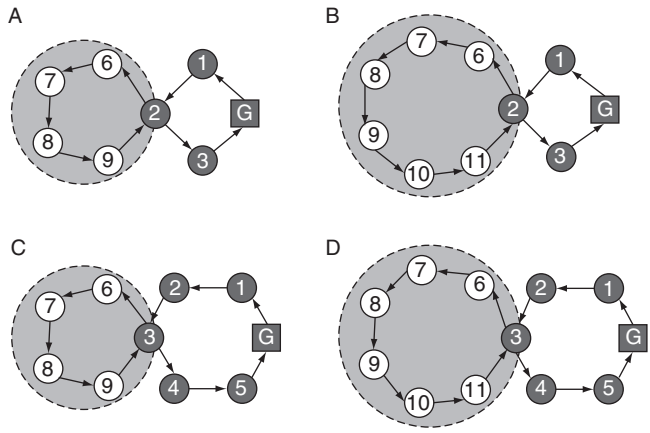

FIG. 11. WSN topology setup for the evaluation of Wiseman agents in a forest fire application.

TABLE I
WISEMAN AGENTS THAT IMPLEMENT APPROACHES A–C

Scheme	Agent	Script
A	1	`l$n;M0=1;R{#M0;I!=0;M0+1;l$n;r$t;O[B>40;M1=I];O[M0<6;M0=0]}`
	2	`l$d;#1;#2;#3;M0=6;R{#M0;M0+1;l$d;I!=0;O[(I==9;M0=3);(I==5; M0=0);!1];r$h;O[B<20;M1=I]}`
B	1	`a#;R{l$w;I!=0;O[(I<4;r$t;B>40;M2<1;M2=1;M0=I;b#);(I>3;r$h; B<20;M1=I;!0);a#;b#]}`
C	1	`R{a#;l$w;I!=0;O[(B>40;M0=I;2∧0);!1]}`
	2	`b#;R{l$d;I!=0;O[(I>5;r$h;B<20;M0=I;!0);a#;b#]}`

threshold that is referenced to trigger the second stage of the process. A 59-byte-long MA that implements approach A visits the cluster-head route formed by nodes 1–3, as mentioned previously. It can be seen in that the variable-target migration method is used here, which does not rely on virtual links. Thus, mobile variable *M0* is incremented by one (*M0+1*) to reach the next destination node in the path made by nodes with sequential ID numbers (*#M0*). After hopping to the next node and toggling the LED, the MA obtains the latest temperature reading (*r$t*) and compares it against the maximum allowed temperature threshold (*B>40*). If the temperature reading exceeds the threshold, then the ID node is stored in mobile variable *M1* (*M1=I*), and it is delivered to the WSN gateway when the MA returns. To ensure

that the MA reaches its final destination, variable *M0* is set to *0* just before the end of the migration itinerary. The second stage of the process initiates next by dispatching a second 85-byte-long MA, whose itinerary is preset to traverse the sensor node path of Sector 2. The second MA enters the corresponding humidity-sensing route, and verifies at each sensor node that the minimum humidity exceeds the predefined threshold. Mobile variable *M1* stores the ID of any node that returns a humidity reading below normal, as per the codes *r$h;O[B<20;M1=I]*. As mentioned before, approach *B* promotes a virtual path scheme, whereby MAs implementing the labeled-path hopping method visit the corresponding sensor nodes in the temperature-reading route assigned to the virtual path *a*, whereas letter *b* is assigned to the humidity-sensing route. The corresponding MA that carries out approach *B* spans 79 bytes of Wiseman codes. Finally, a 36-byte-long MA implements approach *C* by utilizing the local code injection operator (∧) to dispatch a 46-byte-long MA stored at the WSN nodes, which performs the second stage of the monitoring process after a 2-s delay (*2∧0*).

3.7.3 Case Study Results

We performed experiments to gauge Wiseman's generic performance, as well as its efficiency for the implementation of approaches A–C of the early forest fire detection application. The performance metrics that we consider are task completion delay and packet overhead. Table II shows the time it takes to run each Wiseman operation as averaged over 1000 runs in the Micaz sensor nodes, where it is evident that the hop operation incurs the shortest execution time, and the arithmetic operations incur the longest.

On the whole, average operation execution time lingers around 800 μs, which is higher than Agilla because Wiseman parses and tokenizes codes in the form of text strings. Nonetheless, the overall operation execution delay remains within acceptable boundaries. Table III shows the delay incurred by Wiseman MAs averaged over the course of 100 runs.

For these results, the MA's size in bytes is linearly increased to force a growing number of MAC layer segments for each consecutive experiment. When referenced in combination with the results presented in Table II, they yield a good approximation of the overall delay that the WSN operator can expect to see when deploying applications supported by Wiseman agents. Table IV illustrates the numerically calculated number of MAC packets containing the segmented MA for each of the three migration types through 40 hops. The outcome of these experiments further verifies that the labeled-path migration technique performs better than the other two. However, the bandwidth cost of implementing explicit hop migration can rival the one produced by labeled-path hopping if the number of hops that the MA performs is low.

TABLE II
OPERATION COMPLETION DELAY FOR WISEMAN OPERATIONS AND OPERATORS

Operation	Completion delay (μs)
Local broadcast (@)	543
Hop through label (a#)	603
Find label (a?)	613
No operation	732
Halt (!)	743
Hop to node (#1)	753
Insert local code ($\wedge$)	823
Execute ($)	843
Numeric comparisons	873
Add, subtract (+, −)	963
Assignment (=)	973
Multiply, divide (*, /)	1160

TABLE III
WISEMAN AGENT MIGRATION DELAY

MAC layer packets	Migration delay (s/hop)
1	0.235
2	0.250
3	0.264
4	0.278
5	0.294

TABLE IV
BANDWIDTH CONSUMPTION (NUMBER OF MAC LAYER PACKETS) OBSERVED FOR EACH AGENT MIGRATION TYPE

Hops	Fixed path	Variable target	Labeled path
2	2	4	2
10	12	20	10
20	45	40	20
30	103	60	30
40	166	80	40

TABLE V
NUMBER OF MAC LAYER PACKETS INCURRED BY THE WISEMAN AGENTS IN EACH APPROACH

Approach	Topology 1	Topology 2	Topology 3	Topology 4
A	161 [44(A1)+117(A2)]	187 [44(A1)+143(A2)]	209 [66(A1)+143(A2)]	235 [66(A1)+169(A2)]
B	135	165	165	195
C	99 [36(A1)+63(A2)]	117 [36(A1)+81(A2)]	126 [54(A1)+72(A2)]	144 [54(A1)+90(A2)]

Finally, Table V illustrates the results of our experiments for approaches A–C. We can see that Topology 1 yields the shortest task duration. In contrast, Topology 4 yields the longest one, being that this parameter is proportional to the itinerary's path length. In addition to this, approach A always yields the longest task duration because two agents are employed to complete the task. However, approach C yields the shortest task duration, which is directly attributed to employing a single MA that injects another one locally (instead of dispatching it from the WSN gateway).

3.8 Summary

In this section, we have presented the design, implementation, and evaluation of the Wiseman system for supporting WSN applications. The design of our proposed system is based on a much earlier system targeted at supporting active networking in early local area networks. Several modifications had to be incorporated to make Wiseman amenable for WSN node deployment that is characterized by possessing severe hardware and communications bandwidth limitations. Our experiments corroborate that Wiseman can be effectively employed to support WSN applications, and that the efficiency of the MAs employed depends to a good extent on the type of migration technique employed. Specifically, we see that the explicit path migration technique reduces MA design complexity at the expense of higher bandwidth usage for the case of large WSNs, although good results can be obtained for the case of an MA hopping through a relatively small number of nodes before returning to the gateway. One important consideration when coding MAs that implement the labeled-path migration approach is that if the underlying conditions of the monitored environment change, then the semantic relationship expressed by the labeled paths becomes stale. In that case, a separate label maintenance process would need to be executed, thus incurring additional bandwidth. In this case, the gains introduced by

the labeled-path hopping approach could be lost, whereas the variable-target hopping scheme might become the most efficient.

4. Embedding Agents in RFID Tags

4.1 Introduction to RFID Technology

RFID has been a crucial enabling technology for a myriad of commercial and industrial applications, including (but not restricted to): object tracking, inventory control, asset administration, supply chain management, and even an emerging field known as E-healthcare. RFID allows quick access to information assigned to an ID stored in an RFID device, also known as a tag. A RFID tag possesses a radio-frequency transmitter and an antenna to communicate with a special reader that retrieves the tag's ID number and uses it to retrieve the corresponding information from a database. Then, a handling procedure for the bearer object can be determined. This process presupposes a centralized scheme that controls the flow of events, and the actions that need to be taken for every object that has its RFID tag read. Consequently, multiple issues that are detrimental to the operation of the overall system and to its overall efficiency also arise. We refer to this approach as the Identification-centric RFID System, or IRS, whereby RFID tags store a simple alphanumeric string that identifies the bearer object. Though this approach has sufficed for a wide range of applications, one major problem is that updating the database from which any associated data is retrieved is not a straightforward process. This issue also raises a synchronization problem, in addition to possessing two significant disadvantages. First, the database entry for the corresponding object must be created and maintained prior to accessing it to retrieve any associated action. Second, this approach is not feasible for situations that require rapid processing of an object or an event, such as in emergency situations. It is also the case that the information stored in the corresponding database entry might become stale or even unreachable in case of network/system failures, which results in unwanted operation delays or interruptions.

In this section, we advance a CRS as an alternative solution to the problems found in traditional IRS, as previously reported in Chen et al. [60]. CRS employs MAs that are individually stored on demand in RFID tags in place of a regular alphanumeric code. These agents can later enact specific service directives as they are interpreted by the corresponding middleware system when the tag's contents are read. Because of recent memory enhancements in existing RFID tags (e.g., such as in the MB89R118 model by Fujitsu that provides 2 KB of memory [61]), we argue that the compactness of Wiseman agents makes them a good candidate for implementing

CRS applications. It is straightforward to see that CRS enables implementing a much more flexible scheme that promotes local decision making and process handling in response to specific situations. Additionally, full CRS-based solutions can be plausibly integrated with other technologies, such as WSNs, the IP backbone, and cellular networks that form scalable systems for implementing a number of user-driven applications. We believe that CRS has the potential to introduce the following benefits:

a. It enables a scalable and robust system that eliminates the need to access RFID-related information from a remote database.
b. It decreases system response time by implementing local processes.
c. It alleviates system complexity at the database side.
d. It introduces enhanced resilience to database and network failures.

In this section, we introduce the principles of our proposed MA-based system to implement an effective CRS. We also provide an overview of traditional RFID systems, and how they can evolve into CRS solutions. In addition, we provide a detailed rationale behind CRS based on our proposed middleware design. Finally, we describe and discuss preliminary results of an experimental testbed for CRS.

4.2 Review of Identification-Centric RFID Systems

As mentioned before, IRS applications are mostly designed to track or locate objects as they are moved from one place to another. For instance, RFID tags can be attached to chemical containers being transferred from one warehouse to another to provide close monitoring of potentially harmful material. However, RFID technology has become pervasive in supply chain management applications in which the IRS approach suffices by enabling real-time tracking. The same is true in other industrial sectors, such as manufacturing, warehouse storage, shipping and receiving, and purchase transactions. Figure 12 illustrates the product-tracking procedure

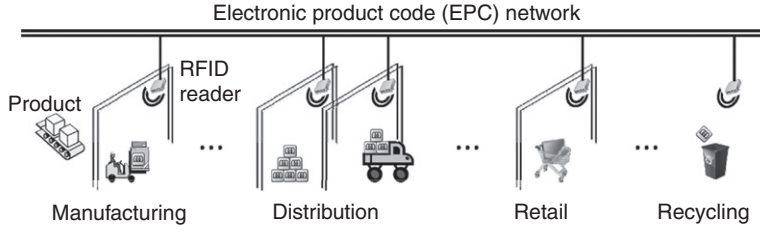

FIG. 12. Supply chain process using a classical IRS system.

through a whole supply chain process. We can see here that an RFID reader retrieves the data stored in a product carrying an RFID upon leaving a manufacturing plant. Then, the object's electronic product code (EPC) information is stored in a database server as soon as it is delivered to a distribution center or to a retailer. After the product is purchased and/or consumed, the RFID tag's information may be stored, for example, in a recycling center's database. In this scheme, the most important components of an IRS are

1. *Rule database:* This module maintains a list of rules associated to RFID code entries to which the corresponding actions can be readily mapped.
2. *Processing module:* This component is in charge of enacting actions/tasks immediately after retrieving the corresponding information from the rule database. Immediately following this, the processing module verifies that any required condition is satisfied.
3. *EPC network:* This subsystem comprises an object naming service (ONS), an EPC information service (EPCIS), and an EPC discovery service (EPCDS), all of which interact to ensure the seamless flow of RFID-related information.

A closer inspection into the operation of the IRS scheme reveals important shortcomings that need to be addressed:

a. The ID number that RFID tags hold does not provide any additional information on what type of service or handling the bearer object requires.
b. The type of service or handling instructions that the bearer object requires needs to be retrieved from a database, whose contents require manual updating.
c. Network and database scalability problems appear when the processing system is unable to handle a growing number of items carrying an RFID tag.
d. Database and/or network malfunctions negatively impact the system's performance.

In Section 4.3, we describe in detail the benefits introduced by the system that we advance to overcome the shortcomings just described for IRS.

4.3 Code-Centric RFID System

4.3.1 System Rationale

The primary goal of an IRS deployment is to help locate goods or objects, as seen in a typical supply chain scenario. Nonetheless, this approach has severe limitations when the service requirements of goods/objects change with time and/or location, in which case a dynamic approach that allows flexible object handling/servicing is

needed. To address this issue, we advance the CRS, in which an MAS becomes the centerpiece of the improved RFID system architecture. In it, the object's handling/servicing directives are specified by MAs embedded into RFID tags in place of regular ID numbers. Our proposed CRS approach serves as a catalyst to harmoniously combine the prevalent environment circumstances and user's requirements. When the conditions surrounding the bearer object change, so can the MA codes embedded into the RFID tag as stored by the corresponding device that provides this context awareness service.

4.3.2 System Architecture

Our proposed CRS is divided in two main parts: the RFID tag, and the code-processing equipment that reads the tags contents (the MA) and executes the necessary actions. To achieve this objective, we introduce an extended message format used for storing information in RFID tags, as illustrated in Fig. 13. In essence, this change stipulates that enough room needs to exist in the RFID tags' memory to store an MA encoded in plain text, in addition to other information embedded into the RFID's memory space. Our proposed CRS approach comprises five principal components: a passive information manager, a middleware subsystem, a codes' information manager, a context awareness subsystem, and a service response system. The passive information manager is employed for retrieving the information stored in the RFID tag, which can be forwarded to the EPC network to generate a backup record. The retrieved MA is then sent to the code information manager, and then onto the middleware subsystem that interprets the MA codes. The context awareness subsystem provides the necessary environmental parameters and information that the middleware subsystem needs to make decisions and execute the appropriate actions. Finally, the commands that enact these actions are forwarded to the service response system, which realizes the commanded tasks that the object requires. The architecture of our proposed CRS is illustrated in Fig. 14.

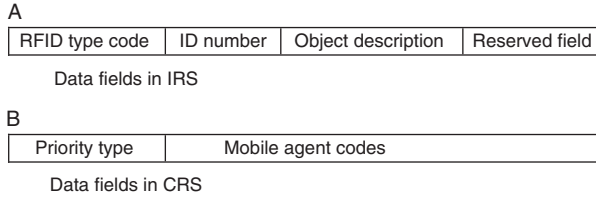

FIG. 13. Extended message format for CRS.

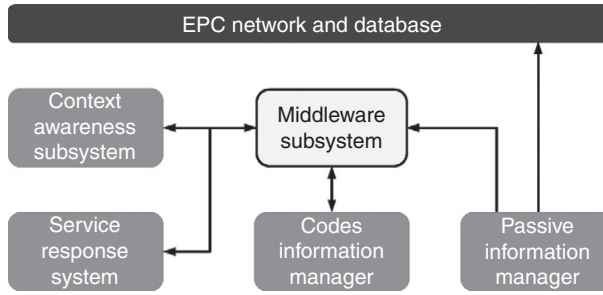

FIG. 14. Principal components of a CRS.

While all the modules mentioned above are indispensible for its operation, we note that the middleware subsystem is what gives CRS its distinctive feature. The middleware subsystem is in charge of interpreting the agent's commands that are eventually enacted by the rest of the infrastructure. MAs may receive limited input to make a decision, which is provided by the context awareness subsystem (e.g., in the form of geographical position, or as environmental information—humidity, temperature, etc.). However, other information relating to the overall system's internals can also be employed. Similarly, different action types can be enacted by the corresponding infrastructure devices after processing the MA, such as interacting with a video surveillance system, updating certain information in the tags of other objects, issuing an alarm signal, etc. Consequently, our proposed system becomes "code-centric."

4.3.3 Updating Mobile Codes

We note the importance of providing the CRS with the means to update MAs on demand, which ultimately makes it possible to provide enhanced quality of service (QoS). To this end, we propose the implementation of three code updating methods: passive, active, and hybrid:

a. *Passive mode:* In this method, the RFID tag has to be updated by a reader device located somewhere in the deployment setting (e.g., along a conveyor in an automatic assembly line). In such case, the handling operations on the product have to be performed one at a time. When the operation associated to the current code being executed finishes, the tags' codes are updated *in situ*. As more MA operations complete, it is expected that the size of the MA codes will shrink as the codes that indicate object handling/servicing actions

gradually deplete. For the case when the object needs multiple processing/handling steps, the RFID tag would have to be replenished with new codes that will be later read, interpreted, and enacted by other devices in a different location (e.g., when the object passes through multiple assembly lines). It follows that the MA's size stored in the RFID tag varies during the bearer object's handling/servicing process.

b. *Active mode:* If the bearer is a person, then the MA's codes possess the necessary directives that indicate specialized treatment or service for him/her. In this case, the user employs a portable device equipped with an RFID reader that seamlessly updates the tag's codes as instructed by a software application. Therefore, the users' interactions with the portable device may lead to the RFID tag's update without his/her knowledge.
c. *Hybrid mode:* This mode of operation combines both passive and active updates. Therefore, a person might have his/her RFID tag updated by a portable device through a software application, or an RFID reader might update the codes after a service has been provided.

For objects and miscellaneous goods, it is expected that the embedded MA will define the actions that need to be carried out, whether to the bearer, or to the surrounding environment, as needed.

4.3.4 Design Issues for CRS

In this section, we introduce the necessary requirements for designing efficient language constructs that can be employed to code an MA for use in CRS. The first step is to determine all the actions that the CRS is intended to support, whether as a service or as a handling. A second, but also important consideration is to determine the memory space that the RFID tags will have. It is easy to see that an efficient language construct design will have a direct impact on the second consideration. An inadequate MA description language design will lead to large RFID tag memory requirements that directly translate into more expensive tags. Therefore, a deployment that spans thousands or even millions of tags would see its associated costs increased by a wide margin. However, RFID's memory capacity and tag type (i.e., active or passive) have a direct impact on its physical size. Therefore, a memory cap might have to be imposed for applications where size restrictions apply, which would evidently pose a constraint for the engineers designing MA language constructs. For the case where RFID tags have more memory, action script constructs with a fine granularity level can be implemented. Conversely, limited memory availability requires that the language constructs' granularity be coarser, as seen in Fig. 15, which shall be meticulously devised. Moreover, from the perspective of the

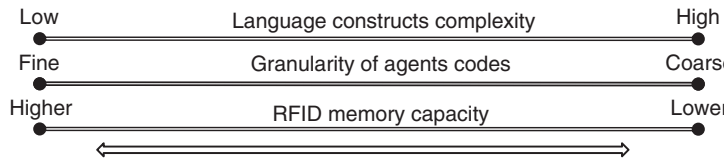

FIG. 15. Language design trade-off impact in RFID memory size.

middleware's design, more complex language constructs require greater processing capabilities and memory space for the interpreter, whereas simpler constructs are needed for hardware devices with limited capabilities.

4.4 An E-Healthcare Application Based on CRS

4.4.1 System Operation's Rationale

In this section, we advance an E-healthcare application based on our proposed CRS, as originally reported in Ref. [62]. In this approach, patients' health conditions are monitored at home, and information is collected and stored at a remote database after being forwarded using a cellular network, a WiFi connection, etc., depending on where the patient resides. To this end, any health-related anomalies not requiring immediate attention are immediately logged into a database, as determined by the patients' ambulatory monitoring devices. At the same time, these conditions are encoded and stored in an RFID tag that the patient carries. The advantage of this scheme is that any critical health information always travels with the patient, which can be retrieved at any location. For example, a patient with a chronic heart condition might decide to take a vacation at a remote place, where his/her condition suddenly worsens. When this happens, the patient's ambulatory monitoring system contacts his/her home healthcare provider, which in turn contacts the local healthcare provider where the patient is currently visiting. If necessary, the RFID's tag information can be encoded in a format that is employed by the remote healthcare provider, which may include instructions on the type of treatment that the patient needs. As a result, a remote healthcare practitioner would not have to establish neither a verbal communication with the patient's home doctor nor a database connection with his/her healthcare provider's computer to obtain this information. Moreover, in case of emergency, medics administering care can readily obtain critical information and the required actions from his RFID tag, which could also encode actions to have the remote equipment directly contact the patient's home doctor. After arriving at a hospital, the local doctor can obtain the patient's relevant

health records and prescription drug history. Similarly, the doctor at the remote location can have his own equipment to update the contents of the RFID tag. This type of enhanced services would be very hard to achieve employing a classical IRS. In addition to this, it is straightforward to see that this application enables a number of enhanced services:

a. A patient facing a health emergency while away from home or a hospital can still be able to receive adequate care by any medical practitioner that has an RFID reader to retrieve critical information. For instance, a person walking by a patient that has collapsed due to an emergency can use his/her cell phone as a gateway, whereby an embedded RFID reader obtains the required information and forwards it to the pertinent healthcare provider. It is here where we can see the benefits of employing an MA that encodes the necessary commands that instruct the RFID reader what to do with the information it just retrieved, such as giving indications to an ambulance's medics of how to deal with the patient.
b. Where feasible, the MA can instruct medical personnel to administer enhanced healthcare services as covered by the patient's medical insurance policy. For instance, the patient might be assigned a private hospital room.
c. The flow of actions that need to be taken according to the situation and/or place can be dictated by the MA, including how to contact the patient's immediate family in case of an emergency, or whether a life/death condition that requires immediate action can be performed as long as the family has encoded a preclearance code in the form of a digital signature that can be validated by the corresponding system.

4.4.2 Infrastructure Architecture

The architecture of the proposed CRS-based E-healthcare system is illustrated in Fig. 16, which employs existing telecommunications technologies and systems as detailed next:

a. *RFID Tag:* As mentioned before, RFID tags can indicate the service level and specific treatment priorities for the patient. In addition, the patient's RFID tag can be programmed to request access to certain areas of a hospital once inside. For example, after being admitted, RFID readers carefully placed in the hospital's gates and corridors can help determine whether a patient is in the correct place. This would help streamline the administration of the required care and prevent errors. Consequently, the contents of a patient's RFID tag can be dynamically updated depending on the healthcare stage where he/she currently is. Therefore, a direct database connection is not needed, in contrast to IRS, given that the RFID tag already encodes an MA that instructs

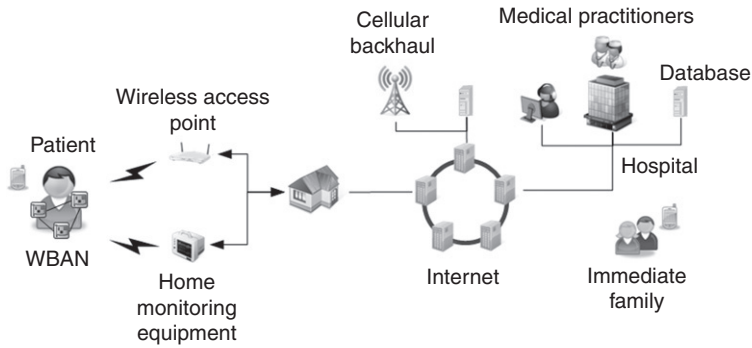

FIG. 16. An E-healthcare scheme based on a code-centric RFID system.

automated gates to grant access in accordance to a previously embedded access code or password.

b. *WBAN:* Physiological signals, such as temperature, heart rate, and muscle tension, provide vital information that can be sensed, collected, and forwarded by a WBAN to a remote healthcare facility for diagnosing the current health of a patient. These signals are collected by small sensors adhered to the patient's skin as strategically placed throughout his/her body. These sensors create wireless links to form a wireless body area network (WBAN) by means of ultra low-power radio technology, such as the one specified by the IEEE 802.15.4 standard [63].

c. *Gateway:* Another element that is crucial for the effectiveness of the proposed CRS-based, E-healthcare system is the communications gateway. This device enables the WBAN to forward collected data to an off-site healthcare provider for remote monitoring, condition assessment, and diagnosis by qualified medical personnel. Different types of communications interfaces can be used here: a cellular phone, a WiFi AP, or a WiMax modem. This implies that the communications gateway will probably be equipped with two radios: one for communications with the low-power WBAN and the other for external communications, possibly making it one of the most expensive devices that a user has to carry.

d. *Database:* In contrast to a classical IRS where a database is key to its operation in a CRS, this element takes on the role of a backup system. As explained previously, the database keeps medical records of patients and other pertinent information that needs to be maintained. In addition, the database needs to be always available for doctors to access directly, as needed. As mentioned before, the contents or corresponding patient's entry can be specifically encoded and stored in patients' RFID tags.

4.5 Summary

In this section, we have advanced an innovative concept that we call Code-Centric RFID System, or CRS. We discussed the potential benefits that our proposed CRS scheme can introduce in a number of applications and deployment settings. Among the most important contributions that CRS provides is: enhanced scalability, distributed information storage and availability, automated processing of goods, and enhanced access control schemes and service for people. We contend that these contributions can be readily achieved by means of RFID tags that possess larger memory to store a combination of carefully encoded information, along with compact MAs that can be interpreted and processed by the corresponding subsystems to enact specific actions. Whereas these type of RFID tags are already commercially available, their high cost precludes their widespread use for handling all kinds of goods. Therefore, the applications where CRS can be conceivably employed are those concerning high-value objects, or people. We believe that CRS would be a key enabler of enhanced IT systems currently being employed in the provision of E-healthcare. As per existing forecasts made for the current aging population, novel and improved systems need to be incorporated to help improve or replace altogether aging technologies that will be unable to cope with future requirements. Our proposed CRS solution employs MAs to provide unmatched flexibility for enhanced services, as clearly exemplified in our E-healthcare application.

5. Final Summary and Conclusion

We have provided an account of our latest advancements in MA technology applied to wireless networks and mobile computing systems. In Section 1, we introduced the reader to basic concepts and particularities of employing mobile codes, and we discussed their potential advantages and disadvantages. We explained that although this technology has the potential to introduce significant bandwidth and process completion delay savings, the degree to which these features become a reality depends on several factors, including the type of MAS employed, the application, and the way in which agents are coded to solve the problem. To this regard, the scheme applied can become inefficient if the number and/or size of MAs deployed to solve a particular problem exceeds the one produced by employing message passing or any other communications mechanism for that matter. Therefore, a trade-off analysis might be necessary to ensure that the MA approach has a good chance of yielding better performance than the current solution. We also

provided a brief historical review of this technology and stressed the importance of revisiting early MAS developed for computer networks that possessed severe bandwidth and power processing constraints, as seen in contemporary WSNs.

In Section 2, we discussed important aspects for architecting MAS from the perspective of WSNs. In particular, we discussed the advantages of employing MAs for image retrieval and TT. Both of these WSN applications are prime examples that show how MAs can be deployed in an attempt to achieve significant bandwidth savings (in the case of image retrieval), or flexibility in the WSN operations (in the case of TT applications). In the first case, one or more MAs can be dispatched from a WSN gateway to examine image data at the location where it was captured, instead of sending raw images for remote processing. For the second case, the MA approach enables WSN operators to deploy distinct types of tracking algorithms on demand to adapt to possible changes in the type or motion characteristics of the object being tracked, thereby providing a much more flexible solution than message passing. We also noted that the WSN might have to be virtually partitioned in a hierarchical fashion to be serviced by multiple MAs when a classical, flat approach results inconvenient. By the same token, we discussed the advantages and disadvantages of employing static, dynamic, and hybrid MA migration itinerary schemes. Though static MA itineraries are more straightforward to implement, they might be insufficient to cope with rapidly changing circumstances as observed by WSN devices, thus raising the need to support dynamic itinerary changes. Alternatively, a hybrid approach might be beneficial, at the expense of increased complexity in the MAS' architecture. Multiple MA cooperation was also discussed, which is an essential trait that MAS designers have to take into account to ensure that agents can operate effectively and efficiently.

In Section 3, we introduced our own MAS for enabling programmable tasking of WSNs. Wiseman's architecture was discussed, along with its instruction set and unique language constructs and code processing flow features. The agent migration methodologies that Wiseman supports were explained in detail, along with examples and design premises for employing one scheme or another. Wiseman's ability to support distinct MA migration techniques is one of its most important features, as it enables static, dynamic, and hybrid schemes. We implemented Wiseman as a proof-of-concept prototype for commercially available WSN devices that possess severe hardware limitations. To this end, we presented both generic performance evaluations and case study results for an early forest fire detection application. In it, we provided examples to show how this WSN application can be effectively implemented through MA technology.

Finally, in Section 4, we advanced a scheme whereby MAs are embedded in RFID Tags to streamline a number of processes and operations. First, we provided a review of IRS that handle simple alphanumeric numbers employed for referencing

database entries that store specific object handling instructions, if any. Then, we introduced what we call the Code-centric RFID System, or CRS. Here, agents are embedded in RFID tags possessing enhanced memory capabilities. This approach enables systems to apply object handling instructions or service directives for people carrying the tag. These actions are obtained directly from the agent, instead of a database. In addition, agents can be updated on demand, and *in situ* without the need to explicitly interact with any off-site system. Agent mobility occurs in two forms. First, actual physical mobility is inherited from the tag's movement with the carrier object. Second, an agent retrieved from a RFID tag can be injected into a network and can be forwarded to distinct system blocks to enact certain actions as specified by its code. As an example of this, we elaborated on an E-Healthcare application that employs our proposed CRS for providing enhanced services to ambulatory patients as they travel. In the coming years, we foresee a renewed interest in MA technology to deal with intricacies of resource-constrained systems where ambient intelligence capabilities are a key requirement. To this end, personalization and customization needs provide the clearest motivation for employing MAs that operate based on the prevailing circumstances, and/or the parameters of the deployment setting.

ACKNOWLEDGMENTS

This work was supported by the National Sciences and Engineering Research Council of the Canadian Government under grants STPGP 322208-05 and 365208-08. In addition, this work was supported in part by NAP of Korea Research Council of Fundamental Science & Technology.

REFERENCES

[1] S. Franklin, A. Graesser, Is it an agent, or just a program? A taxonomy for autonomous agents, in: J.G. Carbonell, J. Siekmann (Eds.), Proceedings Third International Workshop on Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science (LNCS), vol. 1193, Springer-Verlag, New York, 1996.

[2] D.B. Lange, M. Oshima, Seven good reasons for mobile agents, Commun. ACM 42 (3) (1999) 88–89.

[3] P. Braun, W. Rossak, Mobile Agents - Basic Concepts, Mobility Models, and the Tracy Toolkit, Elsevier, The Netherlands, 2005.

[4] N. Borselius, Mobile agent security, Electron. Commun. Eng. J. 5 (14) (2002) 211–218.

[5] W. Jansen, T. Karygiannis, "Mobile Agent Security", NIST special publication 800-19 (Technical Report), National Institute of Standards and Technology (NIST), Computer Security Division, Gaithersburg, Maryland, United States, 2000.

[6] H. Qi, S.S. Iyengar, K. Chakrabarty, Multiresolution data integration using mobile agents in distributed sensor networks, IEEE Trans. Syst. Man Cybern. C Appl. Rev. 31 (3) (2001) 383–391.

[7] D. Milojicic, Mobile agent applications, IEEE Concurrency 7 (3) (1999) 80–90.

[8] M. Ma, Agents in E-commerce, Commun. ACM 42 (3) (1999) 78–80.

[9] W.S.E. Chen, C.L. Hu, A mobile agent-based active network architecture for intelligent network control, Inf. Sci. 141 (1/2) (2002) 3–35, Elsevier.
[10] H. Wang, Z. Deng, Research on JXTA architecture based on mobile agent, in: Proceedings of the Third International Conference on Genetic and Evolutionary Computing, 2009, pp. 758–761, Guilin, China, October, 14–17.
[11] A.R. Tripathi, N.M. Karnik, T. Ahmed, R.D. Singh, A. Prakash, V. Kakani, et al., Design of the Ajanta system for mobile agent programming, J. Syst. Softw. 62 (2) (2002) 123–140.
[12] D. Horvat, D. Cvetkovic, V. Milutinovic, P. Mocovoc, Mobile agents and Java mobile agents toolkits, Telecommun. Syst. 18 (1–3) (2001) 271–287.
[13] G.M.W. Al-Saadoon, A flexible and reliable architecture for mobile agent security, J. Comput. Sci. 5 (4) (2009) 270–274, Science Publications.
[14] S. González-Valenzuela, S.T. Vuong, et al., Evaluation of migration strategies for mobile agents in network routing, in: Proceedings of Mobile Agents for Telecommunication Applications, LNCS, vol. 2521, Springer, Berlin, 2002, pp. 141–150.
[15] Y. Xu, H. Qi, Dynamic mobile agent migration in wireless sensor networks, Int. J. Ad Hoc Ubiquitous Comput. 2 (1/2) (2007) 73–82.
[16] N. Minar, K.H. Kramer, P. Maes, Cooperating mobile agents for mapping networks, in: Proceedings of the First Hungarian National Conference on Agent Based Computation, 1998, Hungary.
[17] J. Raoa, P. Küngasa, M. Matskin, Composition of Semantic Web services using Linear Logic theorem proving, Inf. Syst. 31 (4–5) (2006) 340–360.
[18] S. Arnon, Collaborative network of wireless microsensors, IEEE Electron. Lett. 36 (2) (2000) 186–187, IET Journals.
[19] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirc, Wireless sensor networks: A survey, Comput. Netw. 38 (4) (2002) 393–422.
[20] S. Misra, M. Reisslein, X. Guoliang, A survey of multimedia streaming in wireless sensor networks, IEEE Commun. Surv. Tutorials 10 (4) (2008) 18–39, IEEE Journals.
[21] M. Cortez, J. Sánchez, Wireless communication system for a wide area sensor network, Wirel Sensor Actor Netw, IFIP 248 (2007) 59–69.
[22] A. Sharma, K. Yoon, D. Vanhorn, M. Dube, V. McKenna, M.S. Bauer, ScoutNode: A multimodal sensor node for wide area sensor networks, in: Proceedings of 18th International Conference on Computer Communications and Networks, 2009, pp. 1–6, San Francisco, USA, August.
[23] G. Vigna, Mobile agents: Ten reasons for failure, in: Proceedings of the IEEE International Conference on Mobile Data Management (MDM), 2004, Berkeley, California, USA, , Juanuary, 19–22.
[24] N. Dimakis, J.K. Soldatos, L. Polymenakos, P. Fleury, J. Curín, J. Kleindienst, Integrated development of context-aware applications in smart spaces, IEEE Pervasive Comput. 7 (4) (2008) 71–79.
[25] R. Bose, Sensor networks motes, smart spaces, and beyond, IEEE Pervasive Comput. 8 (3) (2009) 84–90.
[26] I. Marsa-Maestre, M.A. Lopez-Carmona, J.R. Velasco, A. Navarro, Mobile agents for service personalization in smart environments, J. Netw. 3 (5) (2008) 30–41.
[27] G. Bosch, C. Barrue, Managing ambient intelligence sensor network systems, an agent based approach, in: Bio-Inspired Systems: Computational and Ambient Intelligence, LNCS, vol. 5517, Springer, Berlin, 2009, pp. 1121–1128, June.
[28] W. Weber, J. Rabaey, E.H.L. Aarts (Eds.), Ambient Intelligence, Springer, Berlin, 2005.
[29] J. Delsing, P. Lindgren, Sensor communication technology towards ambient intelligence, Meas. Sci. Technol. 16 (4) (2005) 37–46.

[30] T. He, S. Krishnamurthy, J.A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, et al., Energy-efficient surveillance system using wireless sensor networks, in: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services, 2004, pp. 270–283, Boston, USA, June 6–9.
[31] A.B. Mahmood Ali, M. Jonsson, Wireless sensor networks for surveillance applications? A comparative survey of MAC protocols, in: Proceedings of the Fourth International Conference on Wireless and Mobile Communications, 2008, pp. 399–403, July 27–August 1.
[32] TinyOS for deeply embedded Wireless Sensor Networks, http://www.tinyos.net.
[33] A. Fallahi, E. Hossain, QoS provisioning in wireless video sensor networks: A dynamic power management framework, IEEE Wirel. Commun. 14 (6) (2007) 40–49.
[34] Y. Gu, Y. Tian, E. Ekici, Real-time multimedia processing in video sensor networks, Image Commun. 22 (3) (2007) 237–251.
[35] M. Rahimi, R. Baer, O.I. Iroezi, J.C. Garcia, J. Warrior, D. Estrin, et al., Cyclops: In situ image sensing and interpretation in wireless sensor networks, in: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, 2005, San Diego, USA, November 02–04.
[36] P. Kulkarni, D. Ganesan, P. Shenoy, Q. Lu, SensEye: A multitier camera sensor network, in: Proceedings of the 13th Annual ACM International Conference on Multimedia, 2005, Singapore, 6–11 November.
[37] I.F. Akyildiz, T. Melodia, K.R. Chowdhury, Wireless multimedia sensor networks: Applications and testbeds, Proc. IEEE 96 (10) (2008) 1588–1605.
[38] M. Chen, T. Kwon, Y. Yuan, Y. Choi, V.C.M. Leung, Mobile agent-based directed diffusion in wireless sensor networks, EURASIP J. Appl. Signal Process. 2007 (1) (2007) 219.
[39] Y.-C. Tseng, S.-P. Kuo, H.-W. Lee, C.-F. Huang, Location tracking in a wireless sensor network by mobile agents and its data fusion strategies, Comput. J. 47 (4) (2004) 448–460.
[40] Y. Xu, H. Qi, Mobile agent migration modeling and design for target tracking in wireless sensor networks, Ad Hoc Netw. 6 (1) (2007) 1–16.
[41] R. MacRuairi, M.T. Keane, G. Coleman, A wireless sensor network application requirements taxonomy, in: Proceedings of the Second International Conference on Sensor Technologies and Applications, 2008, SENSORCOMM, Cap Esterel, France, 25–31 August.
[42] Y. Xu, H. Qi, Distributed computing paradigms for collaborative signal and information processing in sensor networks, Int. J. Parallel Distrib. Comput. 64 (8) (2004) 945–959.
[43] H. Qi, Y. Xu, X. Wang, Mobile-agent-based collaborative signal and information processing in sensor networks, Proc. IEEE 91 (8) (2003) 1172–1183.
[44] M. Chen, T. Kwon, Y. Yuan, V.C.M. Leung, Mobile agent based wireless sensor networks, J. Comput. 1 (1) (2006) 14–21.
[45] D.L. Applegate, R.E. Bixby, V. Chvatal, W.J. Cook, The Traveling Salesman Problem: A Computational Study, Princeton University Press, New Jersey, USA, 2006.
[46] H. Qi, F. Wang, Optimal itinerary analysis for mobile agents in ad-hoc wireless sensor networks, in: Proceedings of the 13th International Conference on Wireless Communications, vol. 1, 2001, pp. 147–153, Calgary, Canada, July.
[47] Q. Wu, N.S.V. Rao, J. Barhen, S.S. Iyengar, V.K. Vaishnavi, H. Qi, et al., On computing mobile agent routes for data fusion in distributed sensor networks, IEEE Trans. Knowl. Data Eng. 16 (6) (2004) 740–753.
[48] A. Boulis, C.-C. Han, M. Srivastava, Design and implementation of a framework for efficient and programmable sensor networks, in: Proceedings of the First International ACM Conference on Mobile Systems, Applications and Services, 2003, San Francisco, USA, May.

[49] C.-L. Fok, G.-C. Roman, C. Lu, Rapid development and flexible deployment of adaptive wireless sensor network applications, in: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS), 2005, Columbus, USA, June.
[50] P. Levis, D. Culler, Maté: A tiny virtual machine for sensor networks, in: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, 2002, San Jose, USA, October.
[51] T. Liu, M. Martonosi, Impala: A middleware system for managing autonomic, parallel sensor systems, in: Proceedings of ACM SIGPLAN: Symposium on Principles and Practice of Parallel Programming, 2003, San Diego, USA, June.
[52] J. Hui, D. Culler, The dynamic behavior of a data dissemination protocol for network programming at scale, in: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, 2004, Baltimore, USA, November.
[53] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer, L. Iftode, Smart messages: A distributed computing platform for networks of embedded systems, Comput. J. 47 (4) (2004) 475–494, Special Issue on Mobile and Pervasive Computing, Oxford Journals.
[54] P. Sapaty, Mobile Processing in Distributed and Open Environments, Willey, New York, 2000.
[55] P. Sapaty, Ruling Distributed Dynamic Worlds, Wiley-Interscience, New York, 2005.
[56] P. Sapaty, A wave language for parallel processing of semantic networks, Comput. Artif. Intell. 5 (4) (1986) 289–314.
[57] The OMNeT++ Discrete Event Simulator. http://www.omnetpp.org.
[58] Micaz Sensor Nodes by Crossbow Technology. http://www.xbow.com.
[59] C.-L. Fok, G.-C. Roman, C. Lu, Mobile agent middleware for sensor networks: An application case study, in: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, 2005, pp. 382–387, Los Angeles, USA, April, 25–27.
[60] M. Chen, S. Gonzalez, Q. Zhang, V. Leung, Software agent-based intelligence for code-centric RFID systems, IEEE Intell. Syst. 25 (2) (2010) 12–19, 12–19, March/April.
[61] FRAM Embedded, High-speed RFID tag MB89R118 by Fujitsu. http://www.fujitsu.com.
[62] M. Chen, S. Gonzalez, Q. Zhang, M. Li, V. Leung, A 2G-RFID based E-healthcare system, IEEE Wireless Commun. Mag. 17 (1) (2010) 37–43.
[63] H. Cao, V.C.M. Leung, C. Chow, H.C.B. Chan, Enabling technologies for wireless body area networks: A survey and outlook, IEEE Commun. Mag. 47 (12) (2009) 84–93.