A Mobile Code Platform for Distributed Task Control in Wireless Sensor Networks

[1]Sergio González-Valenzuela, [2]Son Vuong, [1]Victor C.M. Leung
[1]Department of Electrical and Computer Engineering
[2]Department of Computer Science
The University of British Columbia
Vancouver, BC, Canada V6T1Z4
{sergiog, vleung}@ece.ubc.ca; vuong@cs.ubc.ca

ABSTRACT
In this paper, we propose an adaptation of the *Wave* system for code mobility as an alternative to existing agent platforms, and describe current work-in-progress of its design and implementation. Our proposed scheme adopts a coarse-grain approach to code mobility, whose functionalities are targeted at the distributed control of wireless sensor networking tasks, while promoting the local execution of repetitive data processing tasks as native code programs. Consequently, our approach introduces conceivable better energy-reduction features by simplifying code processing requirements at the local interpreter, which are augmented by its ultra-compact language constructs aimed at reducing the transmission time of the mobile scripts.

Categories and Subject Descriptors
C.2.4. [**Distributed Systems**]: Distributed Applications, D.3.3 [**Languages Constructs and Features**]: Frameworks.

General Terms: Design, Languages.

Keywords
Wireless Sensor Networks, Mobile Agents, Mobile Code.

1. INTRODUCTION
Recent advancements in both hardware miniaturization and radio technology have enabled the creation of Wireless Sensor Networks (WSNs), whose applicability ranges from sensing of natural and industrial environment settings, first-responder support, health care, and defence [1]. A number of issues pertaining to the design and implementation of WSNs have already been identified in the literature, among which energy-efficient design remains by far the hardest to solve.

To this effect, the use of code mobility has gained significant attention as a plausible alternative to address energy-conservation issues in WSNs. The selling point for the use of code mobility here is twofold. First, mobile code allows the network to be conveniently re-tasked according to the user needs. Second, a programmable approach enables the data computation element of an application to be re-located to the site where relatively large amounts of data were collected, enabling potential energy-savings.

MobiDe'06, June 25, 2006, Chicago, Illinois, USA.
ACM 1-59593-436-7/06/0006.

In this paper, we identify current trends in the development of Mobile Agent Systems (MAS) targeted at WSNs, and introduce the architecture and design foundations of our own Wave-based scheme for the distributed control of spatial tasks. The rest of the paper is organized as follows: Section 2 elaborates on the existing state of MAS for WSNs and how we employ the lessons learned as a benchmark in the design of our own system. Section 3 explains the Wave framework, how it differs from existing approaches, and implementation related aspects. Finally, we present concluding remarks in Section 4.

2. CURRENT STATE OF MOBILE AGENT SYSTEMS IN SENSOR NETWORKS
Interest on the applicability of MAS[1] for WSNs stems from their ability to both facilitate application re-tasking of the network, and introduce energy conservation features [2]. Whether these objectives can be efficiently accomplished depends on a number of factors associated with their individual architectures. As a consequence, the effectiveness achieved by existing mobile code systems in WSNs may be the subject of unfair categorization at present. An example of this can be readily observed between the Agilla [3] and Impala [4] systems. While both systems were designed to support target-tracking applications, the underlying environment in one of them is assumed mobile, while the other is considered static. Thus, although Agilla [5] has been regarded superior than Impala, their actual efficiency will likely be constrained by their deployment environment.

Since WSNs' design and deployment is considered application-specific, it makes sense for MAS to be customized so that agents can accomplish the tasks they were programmed for with as much ease as possible over their targeted environment. We argue that the design of MAS for WSNs must take into consideration factors such as: intended application, update management, hardware platform, and underlying environment, instead of following a general-purpose design. The incorporation of these factors also has a direct effect on the characteristics of the interpreted code that the MAS run. To this effect, existing MAS for WSNs rely on code mobility as a method to enable network re-tasking through the use of disposable code scripts, whereby only the interpreted code that defines the current application is modified. We refer to this as *soft re-tasking*.

[1] In this paper, MAS is used interchangeably to depict a Mobile Agent System or Systems.

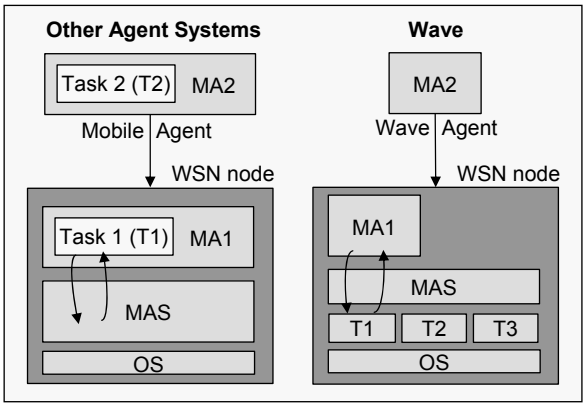

Figure 1. Incorporation of Wave into WSN nodes

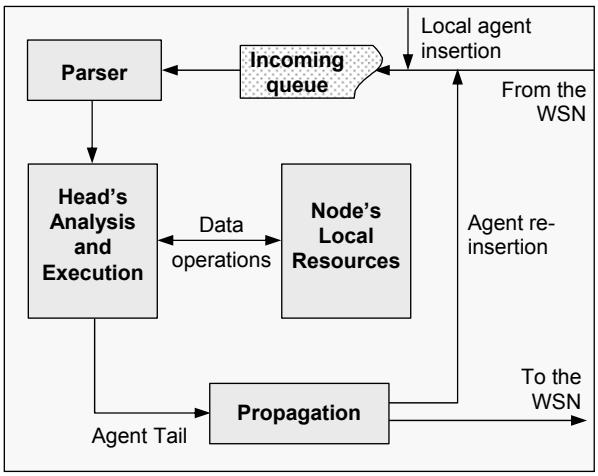

Figure 2. Architecture of the simplified Wave interpreter

Soft re-tasking systems like Agilla, SensorWare [6] and SmartMessages [7] have the ability to redefine the WSN's application by injecting disposable code scripts. This approach enables potential energy savings through the re-location of the data processing element of the WSN application, instead of transmissions of the data to be remotely processed. However, this convenience comes at a price, as code interpreters need to be efficiently designed in order to occupy the least possible amount of memory space, while maintaining robustness. In this regard, MAS for WSNs should attempt to achieve a balance between the degree of functionality incorporated into the actual code interpreter, and the one provided to the agents in accordance to the intended WSN application. In particular, we argue that it makes little sense to provide agents in WSNs with excessive control of the system's functionalities if the tasks to be performed in the WSN are consistently repetitive. Instead, we advocate for the use of a programmable system based on mobile code that enables the efficient control of distributed (and potentially repetitive) tasks in a WSN.

3. DISTRIBUTED CONTROL FOR SPATIAL TASKS IN WSNs

We rely on our previous experience with mobile code in data networks, and propose a simplified version of the original Wave system [8] for incorporation into WSNs. Wave's high-level language constructs allow creating highly compact programs that encompass a suitable degree of distributed task coordination. This approach is highly appealing to WSNs, since it promotes the use of existing functionalities in the node, instead of creating agents that repeatedly perform the same task on every node they visit, as depicted in Figure 1. As a result, the process coordination part of the distributed application is decoupled from the data processing, and is now left for the agents to perform. These agents may then call upon potentially complex data-processing algorithms that are made locally available beforehand at every WSN node. In addition, language constructs compact enough to describe the desired coordination methodology of the mobile process can be defined, which translates into simplified interpreter's architecture and reduced memory footprint. The proposed architecture of our system is shown in Figure 2. The interpreter is comprised of an incoming agent queue, a code parser, an instruction execution block, and an agent propagation block implemented on top of the existing operating system in the WSN host, such as TinyOS [9]

Agents are individually removed from the incoming queue, and parsed into its three fields: code, data and control. The code is then separated into its *head* and *tail* segments. The head is recursively unwrapped from within any precedence delimiters or top-syntactic operations, stopping once a single indivisible operation is found. This indivisible operation is subsequently sent to the execution block for further processing as occurring in the code. The operation's outcome is defined either as success or failure, and is always signalled back to the parser. The agent execution process is halted when an operation cannot be successfully completed or explicit process termination is indicated, in which case the agent' tail is simply discarded. Otherwise, the parser sends the next indivisible operation to the execution block and the process continues. In other words, the agent's tail segments are incrementally executed until depletion. Alternatively, the agent's segments may be contained within a repeat cycle to ensure the continuous execution of the agent until some condition causes the agent to stop. Note that the agent may be forwarded to another node for further processing, and so the tail of the agent is sent to the propagation block for dispatching. When doing so, the agent may be duplicated with as many copies as outgoing links are from the current host as indicated by the agent's construct. It is assumed that the agent's transport and routing to the intended destination is separately performed, and is considered out of the scope of this paper.

Our simplified Wave's constructs are defined by: operators, variables, rules and delimiters. Agents may employ *nodal* variables at the local node that can be of *numeric* or *character* type. These variables are semantically similar to public variables in object-oriented programming, and are accessible to all agents arriving at the interpreter. Agents may additionally carry one or more *mobile* variables with them. These variables are accessible only to the agent that carries them, resembling the role of private variables in object-oriented programming. The programmer, according to the WSN application, can determine the actual number of nodal and mobile variables available to the agents in advance, thus eliminating the need for complex memory management schemes. In addition, agents have access to other e*nvironmental* variables employed to obtain local host information.

Algorithm 1. An avalanche risk assessment process in Wave

```
 1. Assign a risk threshold level to frontal variable M1
 2. Repeat
 3.     Hop to all neighbouring nodes through all virtual links
 4.     if M1 is smaller than the latest rough assessment N1 then
 5.         Assign a "risk" label to the traversed link L
 6.     else halt the current process
 7.     end if
 8. end Repeat
 9. Repeat
10.     Hop to neighbouring hosts through "risk" virtual links
11.     if N2 indicates no assessment performed yet then
12.         run "eval" to assess the localized avalanche risk level
13.         if risk level returned in N3 is greater than M1 then
14.             run "notify" to send an alert signal to base station
15.             Set N2 flag for current node
16.         else halt the current process
17.         end if
18.     end if
19. end Repeat
```

Program 1. An avalanche risk assessment process in Wave

```
M1=6;R(#;O((M1<N1;L="risk");!));
R(risk#;N2==0;N3="eval";O((M1<N3;"notify";N2=1);!))
```

Wave defines simple operators that help fulfill the agents' needs. For example, value comparisons are carried out through conventional *equal-to* and *less-than* operators, whereas arithmetic operations enable the agent with the means to perform simple hop-counting operations, and the like. In addition, memory access operators allow the agent to access information by either type or value, whereas process control operators indicate whether an agent is to hop to another host through a specific link, or to explicitly halt the current task. Finally, other rule constructs such as *And* and *Or* are defined as a way to manipulate execution of the agent's thread by testing whether the code embraced yields a *true* or *false* value for every code segment it includes.

Consider the case of a WSN with various nodes deployed in a sub-alpine terrain setting employed to constantly assess the risk of an avalanche. Lines 1-8 in Algorithm 1 explain the first line of code in Program 1, which takes care of an initial assessment process; possibly triggered by a node once its fictitious reading has reached certain level ("*M1=6*"). Semicolons are employed as segment delimiters. Agents are then cloned and dispatched for a preliminary situation assessment by employing the hop operator "*#*". This operator is employed to indicate that the agent needs to be forwarded to the node specified on the right-hand side of the "*#*" character through the link specified on its left-hand side. Absence of such parameters indicates a hop to all neighbouring nodes through all neighbouring links, which results in the Wave agent's cloning with as many copies as outgoing links are. Once at a remote node, the agent tests whether the local avalanche assessment reading is greater than the specified threshold of 6.

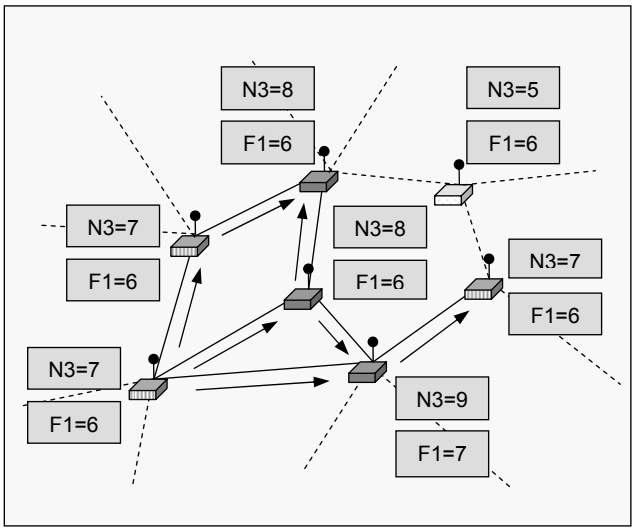

Figure 3. Wave agent propagation example

If the local value in the numeric variable *N1* is greater than the specified one ("*M1<N1*"), the agent labels the previously traversed link with the word "*risk*" by setting the environmental variable "*L*" accordingly (" *L="risk"* "). This action incorporates the current node to a virtual network, whose nodes are linked by a certain semantic descriptor. Note, however, that both of these segments are contained within an "*Or*" rule, indicating that they will be sequentially executed, stopping if one of them results in a *true* value (*O(...;...;...)*). This hop-assess-label process will be recursively executed, since its corresponding code segments are also contained within a repeat rule "*R*" indicated in the first line of the Wave agent in Program 1. Failure of the initial situation assessment segment instructs the current agent to halt ("*!*"). Otherwise, a more rigorous situation assessment is made immediately after the agents finish creating the virtual network as shown in the second line of Program 1, as described in lines 11-19 of Algorithm 1.

For the second part of the distributed risk assessment process, the Wave agents traverse the virtual network through the "risk"-labelled links built by the previous agents in order to reach any nodes that reported a potential avalanche risk reading ("*risk#*"). Variable *N2* is used here as a flag that indicates the agents whether avalanche data readings at the local node have been already processed ("*N2==0*"). Avalanche risk is determined through data analysis by invoking the local program "*eval*". The return value stored in *N3* (" *N3="eval"* ") is used to decide whether an alert should be sent to the base station. As explained before, the agent carries no data processing algorithm. Instead, the "*eval*" program is made available on every node, given the likelihood of repeatedly being invoked in the future. The agents' task is limited to controlling the way in which the risk assessment is made, and can always be changed. The assessment result is then compared to the predefined risk threshold level ("*M1<N3*"), whose successful outcome leads to the invocation of the local program "*notify*". Here, "*notify*" also represents a local binary program in charge of sending certain type of notification to the WSN's base station. Finally *N2* is set to 1, so that other agents

arriving to the local host at a later time omit the local avalanche risk assessment process. Figure 3 depicts the previous example, where the dark-shaded nodes are assumed to finally issue an alert notification to the base station.

Based in the preceding example, we now elaborate on the features that readily make Wave an appealing MAS for deployment over WSNs.

- **Compactness:** The coarse-grained characteristic of the mobile processing concept embraced by Wave enables the definition of a highly compact language construct as seen previously. Wave's compactness would clearly have been compromised if its architecture had been designed with fine-grained control in mind. To this regard, code compactness becomes one of the most significant factors in MAS for WSNs given the scarcity of resources needed to interpret and forward code.
- **Efficiency:** Code compactness enables Wave to incur less bandwidth overhead and transmission power when forwarding agents between hosts. In addition, code compactness might play a crucial role if the application requires rapid data processing (e.g., tracking a fast moving target, while predicting its trajectory). In other words, the efficiency of the WSN might become compromised if the MAS' language constructs are complex enough so that the virtual machine has to spend precious time processing interpreted code. Wave strives to find a suitable balance between code compactness and efficiency that takes into consideration the actual application of the WSN and the underlying environment.
- **Modularity:** Wave's architecture is structured in a modular fashion, making it easier to replace modules if needed, leading to energy conservation and reduced risk of errors during its transmission. This in turn yields better efficiency too. In addition, data forwarding mechanisms tend to be more complex if the amount of data being transferred is larger than a node's RAM can handle [10]. On the other hand, inter-module synchronization might add extra complexity to the MAS depending on its architecture. In the end, the actual WSN application should dictate whether modularity is warranted as a beneficial feature of the MAS.
- **Simplicity:** In addition to the benefits provided by decoupling the control of distributed tasks, we note that Wave's architecture rids the system from the need of forwarding a program counter or process state information. Instead, strong mobility is inherently obtained from the recursive execution of code segments as explained at the beginning of this section. This is a highly beneficial characteristic of Wave, as the typical complexity associated with strong mobility in MAS is avoided.

4. CONCLUSIONS

WSNs in many respects represent the ultimate challenge in data communications systems given their evident scarcity of hardware and energy resources. MAS design for WSNs should take these constraints into careful consideration, along with the intended application in an attempt to achieve maximum energy efficiency, as opposed to generalizing their applicability. We expect an increased interest in mobile processing systems targeted at the efficient control of distributed tasks in WSNs, such as the one proposed in this paper. Wave for WSNs is currently at the implementation and testing stage in a discrete event simulator [11] with the purpose of identifying potential improvements before the system is put in real hardware devices.

5. ACKNOWLEDGMENTS

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through grant STPGP 322208-05. We thank the anonymous reviewers for their comments to improving this paper.

6. REFERENCES

[1] Chong, C.-Y. and Kumar, S.P. "Sensor networks: Evolution, opportunities, and challenges," *Proceedings of the IEEE*, Vol. 91, No. 8, August 2003.

[2] Qi, H., Iyengar, S. S. and Chakrabarty, K. "Multiresolution Data Integration Using Mobile Agents in Distributed Sensor Networks." *IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews*, Vol. 31, No. 3, August 2001.

[3] Fok, C.-L., Roman, G.-C. and Lu, C. "Rapid development and flexible deployment of adaptive wireless sensor network applications," *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05)*, Columbus, USA 2005.

[4] Liu, T. and Martonosi, M. "Impala: A middleware system for managing autonomic, parallel sensor systems," *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, USA 2003.

[5] Fok, C.-L., Roman, G.-C., Lu, C., "Mobile Agent Middleware for Sensor Networks: An Application Case Study," *Proceedings of the 4th International Conference on Information Processing in Sensor Networks* (IPSN'05), Los Angeles, USA, April 2005.

[6] Boulis, A., Han, C.-C. and Srivastava, M. "Design and implementation of a framework for efficient and programmable sensor networks," *Proceedings of the First International ACM Conference on Mobile Systems, Applications and Services,* San Francisco, USA 2003.

[7] Kang, P. et. al. "Smart messages: A distributed computing platform for networks of embedded systems," *The Computer Journal*, *Special Issue on Mobile and Pervasive Computing,* Oxford Journals, 47(4):475-494, 2004.

[8] Sapaty, P. "Mobile Processing in Distributed and Open Environments". *John Willey & Sons*, 2000.

[9] TinyOS: An Operating System for Wireless Embedded Sensor Networks, http://www.tinyos.net.

[10] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems,* Baltimore, USA 2004.

[11] The OMNeT++ Discrete Event Simulator. http://www.omnetpp.org.