Designing Decentralized Software for a Wireless Network Environment: Evaluating Patterns of Mobility for a Mobile Agent Swarm

Vincent A Cicirello
The Richard Stockton College
Computer Science and Information Systems
Pomona, NJ 08240

Andrew Mroczkowski and William Regli
Department of Computer Science
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104

Abstract

Designing decentralized software applications for a wireless network environment offers harsh challenges to the software engineer. All of the usual difficulties associated with a distributed system are present, but are amplified by the inherent dynamics and uncertainty of the wireless network. This paper takes an agent-oriented software engineering perspective in considering how to design decentralized software systems for a mobile ad hoc network (MANET) of resource-constrained devices. Specifically, the authors codify within the context of a software design pattern the concept of an agent swarm. Swarms of mobile agents have been used in the development of applications to support coordination and collaboration in a live MANET testbed. Work is underway to transition some of this technology into use by public protectors as part of the Philadelphia Area Urban Wireless Network Testbed. The objectives of this paper include motivating the need for a swarm-based approach to distributed software for wireless environments and discussing the critical issues involved with mobile agents swarming on a MANET. For example, one such design issue is the selection of migration patterns for use by the swarming agents. Several different types of itinerary patterns are evaluated within the context of a mobile agent swarm.

1. Introduction

First response scenarios in urban environments offer difficult challenges to the development of computing systems that support communications, coordination, and management of public protectors (police, firefighters, paramedics, search and rescue etc.). The likely hardware platform for any such system consists of lightweight computing devices that can be easily carried by personnel who are already weighed down by other equipment of critical importance. Some likely devices include PDAs, Tablet PCs, and possibly laptops. These devices offer the challenge of resource-constrained computing, particularly for a PDA platform. The biggest challenges, however, come from the communications network. The mobile computing platform necessitates a wireless network environment. Wireless networks offer harsh challenges for distributed software systems that are not faced by software operating on wired networks. For example, available bandwidth can be highly limited. Communications intensive software applications are more likely to push the network to the edge of usability in wireless environments than they are in wired networks. In a first response scenario, the reliability of fixed wireless networking infrastructure may be further diminished (e.g., perhaps one or more access points were destroyed by fire). This can result in hosts periodically leaving the network as their users move through wireless blindspots. Many of the challenges of the wireless network environment can be further amplified if the environment is the special infrastructureless case of the mobile ad hoc network (MANET) in which any host may act as a router for wireless network traffic. Although such MANETs offer a “bring-your-own-network” solution to communications and collaboration within a first response scenario, they further increase the difficulty of designing software in an already daunting wireless environment.

It becomes necessary to design software capable of surviving the severe network conditions found in wireless environments. An agent-oriented approach offers the software engineer a structured, reusable approach to designing decentralized software applications for wireless networked environments that are robust in the face of the inherent high level of disruption. Drexel University has been considering many of the issues associated with designing survivable software for MANET environments in their devel-

0-7803-9447-X/05/$20.00 © 2005 IEEE.

opment of the Philadelphia Area Urban Wireless Network Testbed (PA-UWNT) [6, 18]. The PA-UWNT is an initiative linking Drexel with local law enforcement and transportation authorities in the development of a wireless system for the support of communications, coordination, and collaboration of first responders. It consists of approximately two dozen mobile nodes (e.g., Tablet PCs and PDAs). Agent-based systems form the basis for the design of decentralized software applications on this network.

An analysis of the authors' approach to designing software applications for this environment indicates some general patterns of design that are often employed to solve commonly encountered problems. The first is that the agent-based software developed by the authors in this environment is often built upon the concept of a mobile agent swarm. Swarm-inspired agent systems are not a new concept (see for example [4, 17]). But to the best of the authors' knowledge, the concept has not yet been codified in terms of a software design pattern, documenting the concept of a mobile agent swarm as a reusable element of design. Secondly, as one begins to design swarm-based agent software, it becomes clear that the migration logic employed by the individual swarm members is a critical design decision. This is especially true in MANET environments with inherently dynamic network topologies. Aside from the obvious considerations related to task efficiency for the individual agents, in an agent swarm, the individual agents can affect the performance of other swarming agents and the collective performance (e.g., possible network congestion from a poor choice of migration logic).

The remainder of the paper is organized as follows. Section 2 discusses background on agent-oriented software design and also discusses the major challenges to designing distributed software for MANET environments. Section 3 presents our approach to designing distributed software applications for MANET environments. In particular, this section presents the *mobile agent swarm* design pattern, gives example applications for a mobile agent swarm, and discusses the selection of a migration pattern for use by mobile agents swarming on a MANET. Section 4 presents experimental results in a simulated MANET environment evaluating different migration patterns for use by mobile agent swarms. The paper concludes in Section 5 with a discussion and plans for future work.

2. Background

2.1. Agent-Oriented Design Patterns Background

The concept of reusable design is one of great prevalence in the field of software engineering [8], among other fields. The key building block to reusable design is the concept of a *design pattern* which describes a problem ubiquitous to a given environment as well as a solution to that problem in a way that allows for its reuse under varying circumstances for specific instances of the general problem. In terms of software engineering, design patterns have been particularly important for object-oriented (OO) design, but are becoming more commonplace in other areas as well. For example, design patterns are also of growing interest to researchers in agent-oriented software engineering [29, 28, 7, 20, 14]. In particular, several have considered design patterns for mobile agents [15, 1, 11, 27, 5, 24, 25, 13, 16].

2.2. Challenges of a Wireless Network for an Agent-based System

Developing survivable distributed software for a wireless network presents several challenges:

Software operating on a wireless network is faced with a dynamic and resource-constrained network environment. If the wireless network is a MANET, then additionally the environment can be characterized by a dynamic topology and multi-hop routes. A software component may need to communicate with another component located on a host to which a route ceases to exist—or perhaps the host itself has left the network. Due to infrastructure failure or damage (or due to a MANET that has become severed), access to key services can be lost. Software must deal with uncertain communications quality—packets may be lost in transit, signal strength can change, the length in the number of hops or even the existence of routes between hosts can change dynamically.

Software needs to be able to detect and react to network intruders and malicious insiders. This can include requiring agents to reason not only about the posture of the wireless network in terms of information integrity and assurance, but also involves reasoning about the interoperation of network dynamics and information assurance. For example, this can include requiring an agent to reason about route redundancies to avoid transmitting data through a suspected compromised host. If the wireless network is a MANET, then the solution to the compromised host problem is not always as simple as removing the compromised host, as this can potentially sever necessary communications channels, thus adversely affecting information availability.

How best to measure and evaluate the performance of a software system operating on a wireless network? There are the usual issues for the evaluation of a distributed system, including the large amount of data generated by the system; the coordination of data gathered and composed from the elements of a distributed system; measuring performance without having a significant negative impact on it [10]. These issues are greatly amplified in wireless networks, where network dynamics add to an already massive volume of data. Resource constrained computing devices are particularly challenging for the storage of performance data. Limited computing cycles must be allocated to the system software and to the computation of performance metrics. Limited bandwidth must be allocated to mobile agent migration and communications, as well as to transmission of performance data.

2.3. The Philadelphia Area Urban Wireless Network Testbed

The *Philadelphia Area Urban Wireless Network Testbed (PA-UWNT)* [6, 18] is a unique facility developed at Drexel University to study integration, networking and information assurance for next-generation wireless mobile agent systems. It is a system that fully integrates, 1) mobile code, 2) wireless networking, and 3) security infrastructure. The PA-UWNT currently supports industrial-strength, fielded, mobile agent architectures that include, but are not limited to, the Extendable Mobile Agent Architecture (EMAA) [12] from LM ATL and Cougaar [3].

The PA-UWNT enables mobile agents to react to network dynamics. It supports MANET environments, in which hosts have the ability to identify routes dynamically, forwarding packets between hosts out of direct wireless range of each other and which may require multi-hop routes. However, it is not limited to MANET environments. In the PA-UWNT framework, mobile agents can modify the network state, make decisions about itineraries based on network state, and adapt communication modalities to avoid network congestion. In ad hoc mode, the PA-UWNT is not limited to any particular routing protocol, currently supporting OLSR.

3. The Mobile Agent Swarm Design Pattern

Here we present a formal description of the *Mobile Agent Swarm* design pattern in the notation of Gamma et al [8]. Although the concept of an agent swarm is not new, to the authors' best knowledge this is the first attempt to codify it as an element of reusable software design.

Intent: How can the components of a distributed software system operating on a wireless network maintain an up-to-date, synchronized view of some feature of the network state? How can the components of a distributed software system operating on a wireless network maintain a view of the services available on the network? The intent of this design pattern is to enable a distributed software system to monitor the system state despite a potentially highly dynamic, uncertain network characterized by limited system resources (e.g., bandwidth, CPU).

Motivation: The solution is not as simple as having each host broadcast its services (or whatever state feature is to be synchronized), since new hosts may come online after the broadcast. For network state monitoring, there would be an enormous quantity of messages being sent over a bandwidth limited wireless network. Not all hosts or software components may be interested in all of the characteristics of the network's state. Continual broadcasts would flood a bandwidth limited wireless network. Furthermore, if the network is a MANET, then the location, as well as accessibility, of a host may change over time as the network topology changes and as network partitions occur. This makes polling the hosts of interest difficult, if at all possible, since their locations and state of existence can change frequently.

Participants: This design pattern employs a population of relatively simple mobile agents. These are the participants of the pattern.

Collaborations: Each mobile agent knows only about the network service or network state feature that it is to monitor and synchronize. Its knowledge of this service or state feature is rudimentary (i.e., it knows only to gather and timestamp data about the service and to deliver it to hosts interested in it). It migrates from host to host on the network gathering and delivering data.

Consequences: There are several consequences to choosing this design pattern for your particular network monitoring problem. For example, how many mobile agents are needed in your swarm for your particular wireless network? This number is not necessarily a constant. For example, hosts may leave the network or new hosts may join the network. The topology may change (e.g., in a MANET): a completely connected topology may require less mobile agents to keep all hosts up-to-date since any host can be reached by a single hop from any other host; while for other topologies, more mobile agents may be needed in order to update less accessible hosts or to update other hosts about less accessible hosts. Furthermore, if the wireless network partitions into separate networks (e.g., in a MANET), there is the possibility that one partition may end up with an overload (or underload) of mobile monitoring agents. The system may need to adapt an appropriately sized swarm of monitoring agents. The population size may need to be learned and adjusted according to dynamic characteristics of the wireless network in order to fine-tune the performance of the system. Another important consequence to using this design pattern is the need to choose a policy for migration. We discuss some potential migration patterns in Section 3.2. In other work, we have begun considering some of these issues such as network-aware mobile agents that reason about their itineraries [6, 18, 2] as well as ecologically-inspired approaches to swarm population management [6, 19].

Known Uses: In the Secure Wireless Agent Testbed (SWAT) [22, 2] there are several example uses of the pattern *Mobile Agent Swarm*. For example, it is used for message delivery in a distributed whiteboard application. It is also used for collecting and delivering GPS coordinates of the hosts of the SWAT network to other system hosts in the form of map annotations shared by the system users.

Related Patterns: There are several other patterns that can be used in conjunction with this one, including: mobility patterns [11, 15, 13] such as the itinerary pattern [5] or the forwarding pattern [1], and task patterns such as the plan pattern [1]. Some pattern of mobility is necessary for using a mobile agent swarm. The use of a Mobile Agent Swarm also likely involves the Emergent Society Pattern described by Kendall et al [11].

3.1. Examples

Here we illustrate example applications for the *Mobile Agent Swarm* design pattern.

Multi-Robot Mapping. There is much work in the literature concerning mapping of unknown environments by a team of multiple robots, particularly for exploration applications (e.g., [26, 9]). Sharing map data among the robots can be useful for refining map uncertainty and for providing individual robots with a basis for further exploration in regions it has not yet visited. The multi-robot system's communications network can be implemented as a MANET. Depending on the operational environment, this resulting MANET can be highly dynamic. For example, an urban environment would task this robot system with severe obstacles to line-of-sight communications (e.g., buildings, etc). One possible design approach to synchronizing the distributed mapping data across the team of robots is to use a mobile agent swarm. A swarm of mobile agents can "live" on the multi-robot's communications network, migrating from host to host (i.e., robot to robot) collecting map data and delivering it to the robots of the system.

Shared Whiteboard for a MANET. One of the existing applications in the PA-UWNT is a distributed, shared whiteboard. In a highly dynamic MANET environment, a whiteboard application is not a trivial design task. If a host becomes disconnected from the rest of the MANET, it is important to update its whiteboard as soon as possible when it returns to the network to minimize how out of date its information has become. Or for example, when a user annotates the whiteboard, it is important to make efficient use of the limited bandwidth of the MANET to update the other users' copies of the whiteboard. With the dynamic nature of the MANET, this is not an easy task. Broadcasts would create network congestion. Also some nodes may be out of range of the broadcast requiring recipients to repeat the broadcast. This creates additional congestion also requiring whiteboard instances to filter annotations it has already received. The PA-UWNT's whiteboard application instead uses a swarm of mobile agents that wander the MANET collecting annotations and delivering them to other instances of the MANET with the objective of maintaining a synchronized shared whiteboard environment.

MANET Host Localization. Several hosts in the PA-UWNT are equipped with GPS receivers. Given a map of the environment and a host's GPS coordinates, a user can visualize their location. In some applications, it may be desirable to know the physical location of other hosts. For example, consider a host that does not have a GPS receiver. Shang et al. showed it possible to localize a network node by using network connectivity [21]. Therefore, if there is an efficient method for propagating current GPS coordinates of the network hosts with others, then localization of hosts without GPS receivers is possible. The PA-UWNT MANET testbed uses a swarm of simple mobile agents whose job is to collect and deliver GPS coordinates of GPS-equipped nodes. The rationale for using a swarm of mobile agents rather than message broadcasts is the same as that of the whiteboard annotations. It is too costly to feasibly implement a broadcast mechanism.

3.2. Migration Patterns for Mobile Agent Swarms

One of the consequences of choosing the *Mobile Agent Swarm* pattern is that a *migration pattern*, which dictates the way in which a mobile agent chooses the next host it visits, must be selected. No migration pattern is well-suited for all scenarios, so care must be used in the selection of the migration pattern for a particular application. Also, it is not necessary to select only one migration pattern for use within the swarm. If a swarm is composed of multiple types of agents for different applications, then each agent type may benefit from using a different migration pattern depending upon its application's requirements. For example, a fixed itinerary may be sufficient for scenarios in which nodes are relatively stationary. An efficient migration plan for the mobile agent can be generated a priori. In this relatively static network, a pre-planned itinerary will be successful since there is a high confidence that links between nodes will not change during the migration. In other more dynamic networks, a reorderable or even a random itinerary may be better suited. Four migration patterns are now considered: *Pre-defined Itinerary*, *Re-orderable Itinerary*, *Gradient-based Migration*, and finally *Random Migration*.

There are a number of mobility patterns described in the literature (e.g., [11, 15, 13]). For example, one can employ fixed itineraries (e.g., a fixed sequence of hosts that are to be visited in the specified order). This is known as a *Pre-defined Itinerary* or one can allow the mobile agent to modify its itinerary such as in the *Autonomous Itinerary* pattern described by Liotta [15]. There are other possibilities as well. For example, in the *Star-shaped Movement Pattern*, the mobile agent migrates back to its "base" after each host it visits [24]. In the *Branching Pattern*, the agent spawns one copy for each host it needs to visit and simultaneously sends each of the mobile agents to their destinations [24]. Choosing a mobility pattern is not necessarily a straightforward decision. In particular, for a wireless network, the software designer must take care to choose the mobility pattern that best utilizes the limited bandwidth.

One thing that needs to be considered with all migration patterns is how they deal with changing network conditions, such as those which are typical of MANETs. This paper uses the term *migration failure* to refer to the event when a migration to a host at a particular time is unsuccessful. A migration failure can be caused by any number of things including: hardware malfunction, network congestion, or in the case of a wireless network, out of communications range. Also this paper uses the term *neighbors* to refer to the remote hosts reachable by exactly one network hop from the agent's current host. The concept of hops is most apparent in MANETs, where the only way to reach a certain host may be by using another host as an intermediary router. We assume that mobile agents in a MANET have access to up-to-date information about the neighbors of their current host, although not all migration patterns may take this information into consideration.

Pre-defined Itinerary. This type of itinerary consists of an ordered list of hosts to visit that does not change once it has been established. The agent tries to migrate to each host in the itinerary in order. A pre-defined itinerary is useful for situations where the network is not likely to change once the itinerary is given to the agent. If the network is in fact static, then this kind of itinerary is very attractive. First, the ordering of hosts in the itinerary can be generated or even handcrafted for efficiency or to balance network traffic. Also, fixed itineraries put very little computational overhead on the agent, since it does not need to make decisions about where to migrate. Finally, it is easier to track the agent's location because its itinerary is known ahead of time. However, this type of migration pattern does not deal well with failed hosts or highly dynamic network conditions.

There are several ways with which an agent using a predefined itinerary can deal with migration failure. In its simplest form, an agent can simply record the failed migration and continue to the next host on its itinerary. Alternatively, the agent can be instructed to append failed hosts to the end of its itinerary, which causes it to re-visit these hosts at a later time.

Re-orderable Itinerary. With this type of itinerary, the agent is given a list of hosts it should try to visit, but no predefined order in which to visit them. Instead, the agent is constructed with some means to choose a "good" next destination. A re-orderable itinerary is much better suited for highly dynamic applications where network connectivity is not known ahead of time. Some extra computation responsibility is put on the agent, but that is acceptable considering that the agent will be much better equipped to deal with changing network conditions. There are many possible ways an agent can choose its next migration candidate. A simple example heuristic is to migrate to a current, unvisited neighbor of the current host. This would lessen unnecessary network hops. If no unvisited neighbors exist, the agent may then randomly select a remaining host on its itinerary. More robust agents may be able to sense connectivity beyond the next hop, and also take into consideration the trustworthiness of hosts in the network.

A re-orderable itinerary is much more robust in terms of migration failures than a pre-defined itinerary. If a migration failure occurs, the agent may go to the next best host, and place the failed host back in its itinerary. The agent should be given a maximum number of times to retry a particular host to avoid repeatedly retrying a host that may no longer exist. However, this type of agent will not do as well if the information needed to make its decisions is missing or if the information is (perhaps maliciously) innaccurate.

Gradient-based Migration. A third migration pattern considered in the experiments of this paper is called gradient-based migration. In this migration pattern, the agent does not directly employ a set of hosts for its itinerary. Instead, it explores the MANET, maintaining a list of the hosts it has encountered. The agent maintains a record of the number of times it has visited each of these hosts. When faced with a migration decision, it visits the neighbor of its current host that it has visited least often. If more than one neighbor has been visited equally least often, then it chooses randomly from these. This migration pattern encourages exploration of parts of the network for which it has the least experience. It is robust to network dynamics, not restricting its migration path to a pre-defined list of hosts. Although this pattern does not use an explicit itinerary, it is essentially a variation of the autonomous itinerary pattern.

Random Migration. In this type of migration pattern, agents choose the next host to which to migrate at random from a list of the current node's neighbors, basically perfoming a "random walk" of the network. This migration pattern is included in the experiments of this paper as a baseline for comparison. Besides being a good baseline, a random migration pattern may be a good choice for some applications. When network conditions are unpredictable and data about network routes is either unreliable or unavailable, random walking agents usually do no worse, and sometimes better, than some of the more "intelligent" migration patterns. Variations of a random walking agent can be given some basic guidelines, such as avoiding migration back to the host from which it just came.

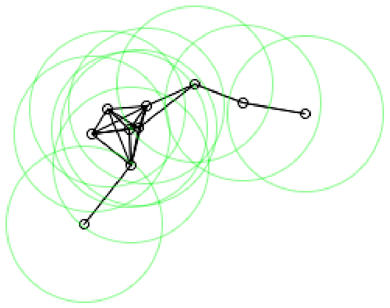

Figure 1. A network snapshot from the discrete event simulator

4. Experiments

The example applications of the mobile agent swarm presented earlier in Section 3.1 are all instances of a data synchronization problem. There is a set of mobile network hosts, each of which require some piece of data about the distributed system state. The state of the hosts is changing over time as they move about in their physical environment. The hosts require a view of the others that is as up-to-date as possible. For example, if the hosts are a team of mobile robots mapping an area, then the task is to synchronize mapping information. If the hosts are Tablet PCs on which a shared whiteboard application is functioning, then the task is to synchronize the state of the whiteboard.

Mobile agents are used to synchronize the data across the set of hosts. An agent carries with it the latest copy of the data that it has received about the state of each individual host. When the agents migrates to a host, it merges its data with the host's latest data, updating both the host and the agent. The agent then migrates to another host and continues this process. Timestamps are used in this data merge process to ensure that the agent keeps the most up-to-date information.

The experiments run in a discrete event simulator, known as MATES [23], specifically designed for testing mobile agent control logic in a simulated MANET environment. Figure 1 shows a representative network topology from the simulator. The simulation uses the following parameters and assumptions:

- The simulation uses a mobility model that moves a host randomly at each iteration. Hosts can wander out of radio range, but network partitions are prevented. There is always a route between two hosts, although it may consist of several hops.
- Routing data is made available to the agents. This means that an agent is aware of the neighbors of its current host as well as information on how to reach non-neighbor hosts.
- The term *destination* means the next host on the agent's itinerary.
- If hosts are within radio range, a link exists between them. Agents take exactly one simulation iteration to travel across a link.
- Agents have no knowledge of each other and do not communicate.
- Two experiments are considered—one with 10 hosts and one with 20 hosts. Each simulation lasts for 100,000 iterations.
- The number of agents working collaboratively is varied from one to four.
- Four types of agents are considered as described below. Each type differs by the mobility pattern used.

The four types of agents used in the experiments include:

FixedAgent. This agent used a *Pre-defined Itinerary* as it mobility pattern. The *FixedAgent* tries to visit every host on the network, in a pre-specified order. In our simulation, we assume that an ad hoc routing protocol is used that provides multi-hop routes. The simulation is also designed to ensure that the MANET never severs. Additional experiments are in process that removes this assumption. Given these assumptions, the *FixedAgent* requires some fixed number of simulation steps to reach the next host in its itinerary, depending on the length of the current route between its current location and its destination. Once the *FixedAgent* reaches the last host on its itinerary, it tries to migrate back to the first host and repeats the process iteratively. Note that the *FixedAgent* may migrate through hosts to reach its next destination if it cannot be reached in a single hop.

ReorderableAgent. Like the *FixedAgent*, the *ReorderableAgent* agent tries to visit every host on its itinerary. However, the *ReorderableAgent* migrates to the closest (fewest network hops) unvisited host on its itinerary instead of simply choosing the next host. If more than one host is considered closest, it randomly chooses one of them. Essentially, the ReorderableAgent follows a greedy heuristic in determining the order of hosts for migration. Once the *ReorderableAgent* has visited all of the hosts on its itinerary, its itinerary is reset and it begins touring the network again. The ReorderableAgent does not update intermediary hosts on its way to its next destination. These intermediary hosts are used only to route the agent to the next host on its itinerary.

RandomAgent. The *RandomAgent* chooses its destination randomly from its neighbors. This agent does not maintain any record of hosts it has visited, nor does it have a set itinerary. The *RandomAgent* synchronizes on every migration since it always chooses its next destination from its neighbors.

Type	NumAgents	Mean	Std. Dev.
Random	1	1577.780	1021.12
Random	2	720.707	460.67
Random	3	496.132	346.77
Fixed	1	537.770	103.45
Fixed	2	560.080	108.03
Fixed	3	625.183	128.46
Reorderable	1	348.172	72.97
Reorderable	2	252.339	62.47
Reorderable	3	206.102	56.14
Gradient	1	443.909	134.32
Gradient	2	287.917	107.75
Gradient	3	213.227	89.00

Table 1. 10-host experiment results

GradientAgent. This agent is an improvement upon the *RandomAgent*. The *GradientAgent* does not have an itinerary, but keeps track of visits to each host by incrementing a counter after a successful migration. The next destination of this agent is the neighbor with the least number of prior visits. If more than one host has an equal number of visits, the *GradientAgent* chooses randomly from among them. Like the *RandomAgent*, this agent updates every host to which it migrates.

System performance is measured in terms of the team of agents. That is, for runs that involve a single agent, performance is given for that single agent. For runs in which a group of N agents of a single type is used for data synchronization, results reported is for the collective performance of the group. The objective that we call out-of-dateness is defined as:

$$\text{OutOfDateness} = \sum_{h \in H} \max_{g \in H}(T - T_h(g)), \qquad (1)$$

where H is the set of hosts in the MANET, T is the current time, and $T_h(g)$ is the time that host h was most recently updated about the state information of host g. We report the OutOfDateness for different numbers of agents for each of the types of agents. The results are averaged across the 100,000 simulation time units and presented for networks consisting of 10 and 20 hosts (Table 1 and Table 2, respectively).

Some observations can be made from these results. First, the two best migration patterns for this problem from those that have been evaluated are the Re-orderable Itinerary and the Gradient-based migration. In the ten host problem, these two types of agents appear to be converging in performance. The ReorderableAgent performs slightly better. This can be attributed to this agent type's usage of an itinerary to ensure it visits each of the hosts. The GradientAgent always chooses a neighbor as its next destination. The next best agent type in this example is the RandomAgent. This

Type	NumAgents	Mean	Std. Dev.
Random	1	12915.547	8605.03
Random	2	5804.275	3584.82
Random	3	3911.574	2745.25
Random	4	2791.396	1859.25
Fixed	1	3877.396	775.72
Fixed	2	4300.934	947.05
Fixed	3	4667.739	1021.92
Fixed	4	4584.596	996.77
Reorderable	1	1593.642	335.38
Reorderable	2	1127.592	271.91
Reorderable	3	909.967	229.27
Reorderable	4	780.212	202.48
Gradient	1	2346.064	722.29
Gradient	2	1474.223	512.70
Gradient	3	1127.283	434.06
Gradient	4	893.248	349.83

Table 2. 20-host experiment results

leads one to speculate the possible reasons that the FixedAgent performs so much worse than random. Recall that the FixedAgent uses a Pre-defined Itinerary, using multi-hop routes if necessary to visit and update the set of hosts in a pre-specified order. If the network topology is static, then one can handcraft an efficient pre-defined itinerary. Or if the network is fully-connected then any pre-defined itinerary would be as good as any other since every host would be a single hop away from any other. But given the dynamic nature of a MANET, a fixed itinerary can lead to migration involving long routes between consecutive stops on the itinerary.

5. Conclusion and Future Work

In this paper, the challenges to designing distributed software applications for a MANET environment have been considered. Our approach to tackling these problems has focused on groups of relatively simple mobile agents used for tasks such as data synchronization and resource monitoring. This paper has taken our general approach and presented it within the context of a software design pattern that we have called a Mobile Agent Swarm. In using this design pattern in practice, it becomes necessary to design the migration logic for the individual agents. We have begun to consider this design decision, considering a catalog of migration logic patterns in a simulation of a MANET data synchronization problem. Ongoing work includes:

Expanding the selection of migration patterns. This paper has considered only a few of the possible alternatives for the migration logic of the swarming mobile agents. The possibilities are not restricted to itineraries. Myopic approaches such as the Gradient-based Migration presented here that consider only its direct neighbors show promise and additional such patterns will be considered.

Choosing from alternative migration patterns. Once a thorough evaluation of the possible migration patterns has been conducted, one can envision an approach that empowers the mobile agents with a menu of migration logics from which to choose. The agents can then potentially make use of locally available network state information to select an appropriate migration strategy.

Benchmarking migration patterns for a mobile agent swarm on a live MANET. The results in this paper have been for a simulated MANET environment. These tests will be repeated in the live MANET environment of the PA-UWNT. Certain assumptions have been made in the design of the simulation, some of which may not hold in reality. For example, in real MANET environments, one cannot hope to maintain a network state free of islands and partitions. The simulations presented in this paper deliberately dealt with what may seem like small networks. This, however, is characteristic of the known physically implemented MANETs (in academic settings) that typically consist of no more than dozens of hosts.

References

[1] Y. Aridor and D. B. Lange. Agent design patterns: Elements of agent application design. In *Proceedings of the International Conference on Autonomous Agents*, 1998.

[2] D. Artz, M. Peysakhov, and W. Regli. Network metareasoning for information assurance in mobile agent systems. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1455–1457, August 2003.

[3] BBN Technologies. Cougaar architecture document. http://docs.cougaar.org, February 2003.

[4] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, 1999.

[5] D. Chacon, J. McCormick, S. McGrath, and C. Stoneking. Rapid application development using agent itinerary patterns. Technical Report TR-01-01, Lockheed Martin Advanced Technology Laboratories, March 2000.

[6] V. A. Cicirello, M. Peysakhov, G. Anderson, G. Naik, K. Tsang, W. C. Regli, and M. Kam. Designing dependable agent systems for mobile wireless networks. *IEEE Intelligent Systems*, 19(5):39–45, September/October 2004. Special Issue on Dependable Agent Systems.

[7] T. T. Do, M. Kolp, and A. Pirotte. Social patterns for designing multiagent systems. In *Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering*, 2003.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[9] R. Grabowski, L. Navarro-Serment, C. Paredis, and P. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots - Special Issue on Heterogeneous Multirobot Systems*, 1999.

[10] A. Helsinger, R. Lazarus, W. Wright, and J. Zinky. Tools and techniques for performance measurement of large distributed multiagent systems. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, pages 843–850, July 2003.

[11] E. A. Kendall, P. V. M. Krishna, C. V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In *Proceedings of the International Conference on Autonomous Agents*, 1998.

[12] R. Lentini, G. P. Rao, J. N. Thies, and J. Kay. Emaa: An extendable mobile agent architecture. In *AAAI Workshop on Software Tools for Developing Agents*, July 1998.

[13] K. Lim and R. Stadler. Devloping pattern-based management programs. In *Proceedings of the 4th IFIP/IEEE International Conference on Management of Multimedia Networks and Services: Management of Multimedia on the Internet*, number 2216 in LNCS, pages 345–358. Springer, 2001.

[14] J. Lind. Patterns in agent-oriented software engineering. In *Proceedings of the 3rd International Workshop on Agent Oriented Software Engineering*, 2002.

[15] A. Liotta and G. Knight. Decomposition patterns for mobile-code management. In *Proceedings of the HP-OVUA, the Hewlett-Packard Openview University Association Plenary Workshop*, April 1998.

[16] Q. H. Mahmoud. Security policy: A design pattern for mobile java code. In *PLOP 2000: 7th Pattern Languages of Programs Conference*, 2000.

[17] H. V. D. Parunak. Go to the ant: Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[18] M. Peysakhov, D. Artz, E. Sultanik, and W. C. Regli. Network awareness for mobile agents on ad hoc networks. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-2004)*, July 2004.

[19] M. Peysakhov, V. A. Cicirello, and W. Regli. Ecologically inspired agent control model. In *The Third NASA-Goddard / IEEE Workshop on Formal Approaches to Agent-Based Systems (FAABS III)*, 26-28 April 2004. Greenbelt, Maryland.

[20] O. F. Rana. Performance prediction from agent design patterns. In *Software Performance Prediction Extracted From Designs Workshop Program*, November 1999.

[21] Y. Shang, W. Ruml, and Y. Zhang. Localization from mere connectivity. In *MobiHoc '03*, pages 201–212, June 2003.

[22] E. Sultanik, D. Artz, G. Anderson, M. Kam, W. Regli, M. Peysakhov, J. Sevy, N. Belov, N. Morizio, and A. Mroczkowski. Secure mobile agents on ad hoc wireless networks. In *The Fifteenth Innovative Applications of Artificial Intelligence Conference*, pages 129–136. American Association for Artificial Intelligence, August 2003.

[23] E. A. Sultanik, M. D. Peysakhov, and W. C. Regli. Agent transport simulation for dynamic peer-to-peer networks. Technical Report DU-CS-04-02, Department of Computer Science, Drexel University, September 2004.

[24] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development method based on agent patterns. In *Proceedings of the 21st International Conference on Software Engineering*, pages 356–367, 1999.

[25] Y. Tahara, N. Yoshioka, A. Ohsuga, and S. Honiden. Secure and efficient mobile agent application reuse using patterns. *ACM SIGSOFT Software Engineering Notes*, 26(3):78–85, May 2001.

[26] S. Thayer. Four generations of robotic mapping and exploration in extreme enviroments. In *International Symposium on Artificial Intelligence, Robotics and Human Centered Technology for Nuclear Applications (AIR'02)*, pages 85–92, January 2002.

[27] A. I. Wang, A. A. Hanssen, and B. S. Nymoen. Design principles for a mobile, multi-agent architecture for cooperative software engineering. In *The 4th IASTED International Conference on Software Engineering and Applications*, November 2000.

[28] M. Weiss. On the use of patterns in agent system design. In *AAMAS-2002 Workshop on Agent-Oriented Information Systems*, 2002.

[29] M. Weiss. Pattern-driven design of agent systems: Approach and case study. In *Conference on Advanced Information Systems Engineering*, volume LNCS 2681 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.