# An Adaptive and Efficient Peer-to-Peer Service-oriented Architecture for MANET Environments with Agile Computing

Niranjan Suri[1,2], Massimiliano Marcon[1,3], Raffaele Quitadamo[1,4], Matteo Rebeschini[1], Marco Arguedas[1], Stefano Stabellini[1,3], Mauro Tortonesi[3], Cesare Stefanelli[3]
[1] Florida Institute for Human & Machine Cognition
[2] Lancaster University
[3] University of Ferrara
[4] University of Modena and Reggio Emilia
{nsuri,mmarcon,rquitadamo,mrebeschini,marguedas,sstabellini}@ihmc.us
{mtortonesi,cstefanelli}@ing.unife.it

*Abstract*–**Realizing adaptive and efficient peer-to-peer Service-oriented Architectures for MANET environments is a challenging problem. In particular, robust and efficient service discovery and service migration are critical in the constantly changing and bandwidth limited MANET environments. In these scenarios, service migration lays the foundation for self-adaptive architectures. This paper describes the agile computing approach to peer-to-peer service discovery and service migration and provides a performance evaluation of these functions in the context of the Agile Computing middleware. The experimental results presented in the paper show that applications built on top of the Agile Computing middleware are capable of opportunistically exploiting transient computing resources in the MANET environment.**

*Index Terms*–**Agile Computing, Peer-to-Peer, Service Discovery, Service Migration, Service-oriented Architectures, JXTA.**

## I. INTRODUCTION

Service-oriented Architectures (SoAs) are a popular approach for designing and building networked and distributed systems. SoAs allow the realization of complex distributed applications through the composition of services according to the definition of a specific business process. This approach offers the opportunity for both extensive service reuse and the integration of heterogeneous services, with significant savings in distributed applications development costs and time.

Most SoAs were designed to operate on either corporate networks or in the Internet environment, with the purpose of separating business processes and rules from the implementation of basic service functions. Hence, traditional SoA implementations are based on centralized service directories and make strong assumptions about relatively static network topologies, large bandwidth availability, and high network stability.

However, a growing interest in running SoAs in MANET environments has recently emerged. In fact, SoAs allow the dynamic (re)composition of services at run-time, thus enabling the ad-hoc realization of complex distributed applications. In addition, SoA-based applications build on top of lean and modular services, which are better suited for deployment on mobile and resource-constrained nodes than traditional heavyweight services. Finally, the modular architecture of SoA-based applications allows for the dynamic replacement of services, therefore enabling application adaptation to the constantly changing network topology of MANETs.

Unfortunately, the realization of SoAs in MANETs is a very challenging task which requires a significant re-engineering of existing SoA platforms. In particular, the MANET environment is not well suited for the deployment of traditional centralized service broker(s) and bandwidth intensive protocols such as SOAP. One requirement is a decentralized and distributed peer-to-peer approach to service discovery, which can efficiently find service instances and select the best suited ones for exploitation, such as those satisfying topological proximity and/or reliability constraints. Also, dynamic service migration is an effective approach to improving performance and availability of services in the face of frequent variations in network topology and resource availability. Finally, in MANET environments, bandwidth is typically very scarce, thus imposing strict efficiency constraints on signaling protocols involved in service discovery, migration, and invocation.

There is a growing effort from both industry and academia to develop solutions for SoAs in MANET environments [1] [2]. However, it is still unclear whether these proposals enable the SoA architectural style to deliver satisfying performance levels in MANETs. Moreover, they do not address service migration as a means of adaptation. As a result, there is still a need to gather some insights on the performance and efficiency of state-of-the-art solutions.

This paper describes and experimentally evaluates two aspects of SoAs – service discovery and service migration. The Agile Computing middleware is our solution for the realization of SoAs specifically designed to operate in MANETs. It is based on a peer-to-peer architecture and on an adaptive and opportunistic paradigm for resource/service discovery and exploitation, which we have called *agile computing*. In particular, the paper focuses on the dynamic service discovery and service deployment, activation, and migration functions of the Agile Computing middleware,

978-1-4244-2067-4/08/$25.00 ©2008 IEEE.

which are respectively provided by the Group Manager [3] and AgServe components [4].

This paper experimentally compares the performance of the Group Manager with Sun Microsystems' JXTA Technology [5] [6]. While JXTA was not designed for MANET environments, and other papers describing its inadequacies have been published [7], it continues to be a popular platform for peer-to-peer systems [8]. Moreover, JXTA has been selected to provide the peer-to-peer discovery service for the System of Systems Common Operating Environment (SOSCOE) for the U.S. Army's Future Combat Systems (FCS) initiative[1], which is also one of the target environments for the Agile Computing middleware, thereby making such a comparison fair.

This paper also presents some performance results for service migration in AgServe, which enables load-balancing, fault-tolerance, and the application of autonomic computing principles to SoAs.

## II. AGILE COMPUTING

Agile computing is a novel metaphor for distributed and networked systems. It emphasizes designing systems to be opportunistic in discovering and exploiting resources in a dynamic environment as well as being able to quickly adapt to changes in such an environment. The word *agile* is used to highlight both the rapid discovery and exploitation of resources and the ability to take advantage of highly transient resources.

The Agile Computing middleware is a specific implementation of the agile computing metaphor that proposes a peer-to-peer approach for the realization of SoA-based distributed applications in the MANET environment. The middleware consists of six major components: a) the Agile Computing Infrastructure (ACI) Kernel, b) the Group Manager, c) the Service Manager, d) Mockets, e) AgServe, and f) FlexFeed. Figure 1 shows the overall architecture for the middleware.

The ACI Kernel provides the container functionality for hosting and executing services. It also instantiates and contains the Group Manager component, which supports dynamic resource and service discovery. The Service Manager provides service matching functions on top of the Group Manager, providing applications with a convenient interface based on XML and XPath. Mockets is a communications library providing several advanced features that were purposely designed for the MANET environment. The endpoint migration capability of Mockets is particularly relevant to the service migration capability described in this paper. AgServe provides support for dynamic deployment, activation, and migration of services. Finally, FlexFeed is a publish-subscribe system that handles hierarchical data dissemination, policy-based transformation of data, and in-stream data processing.

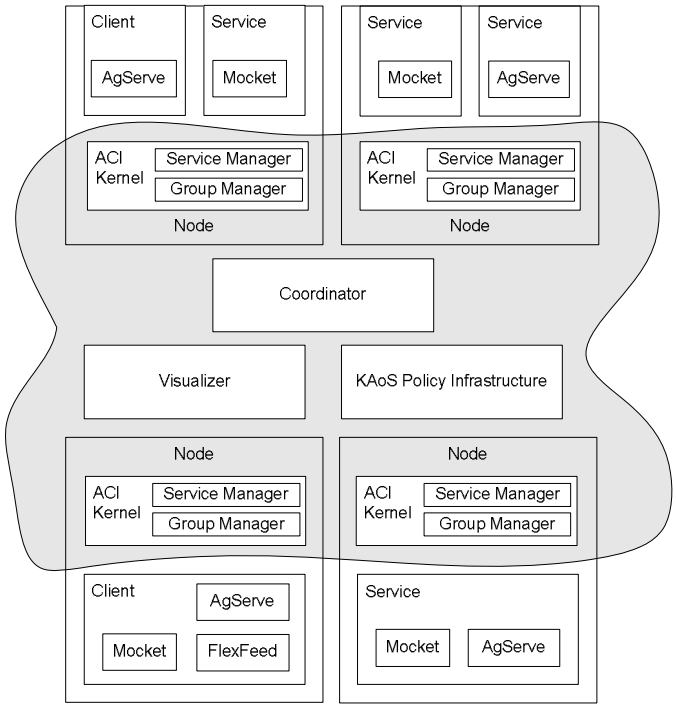


Figure 1: Agile Computing Middleware Architecture

This paper focuses on the Group Manager and AgServe components, which are further described in sections III and IV. More information about agile computing, Mockets, and FlexFeed is available in [9], [10], and [11] respectively.

## III. GROUP MANAGER

The Group Manager is an application-level component that supports resource and service discovery. It enables the agile and opportunistic exploitation of resources by optimizing queries to find nodes in network proximity and/or nodes that are resource rich or have excess capacity.

The Group Manager supports proactive advertisement, reactive search, or a combination of the two. The search is realized using a Gnutella-like or probabilistic search mechanism. In addition, the radius (in terms of network hops) of the advertisement or search can be controlled on a per request basis, providing a powerful mechanism to control how strongly or weakly a service may be advertised and how far a search request may travel. In the simplest case, distance is defined as the number of hops in a MANET environment, but can be a more application relevant parameter such as bandwidth or latency.

Propagation of the advertisement and search messages occurs via one of three mechanisms – UDP broadcast (the simplest case), UDP multicast, or via the XLayer Framework that provides bandwidth-efficient flooding [12]. In addition, tunneling via TCP supports bridging multiple networks. In all of these four cases, each node may selectively rebroadcast an incoming message to provide control over the radius.

[1] See Sun Microsystems' Announcement "Global Governments and Industry Partners Rely on Sun Microsystems to Achieve Security and Scalability" at http://www.sun.com/smi/Press/sunflash/2005-05/sunflash.20050503.5.xml

These capabilities enable applications to make tradeoffs between discoverability, bandwidth, and latency. Proactive advertisement uses more bandwidth but reduces the latency when a client needs to find a service and vice-versa. On another dimension, a service that is widely present in a network does not need to advertise strongly (or, consequently, a client looking for such a service does not need to search widely) as opposed to services that are scarce.

Groups may be used to partition network nodes into different sets thereby restricting advertisements and queries. The Group Manager provides support for two different group types: peer groups and managed groups. Peer groups are completely decentralized: they do not have an owner or manager, but instead maintain node membership independently from the perspective of each node. This design choice also implies that there is no attempt at maintaining a consistent group view for all nodes that are members of a peer group. In addition, no special mechanism is required in order to join a peer group. Nodes can simply query or register resources and services in the context of a specific peer group and will implicitly be treated by the Group Manager as members of the same peer group.

The decentralized and dynamic nature of peer groups make them very well suited for resource and service sharing in MANET environments. However, for additional flexibility, the Group Manager also supports centralized resource and service management by means of managed groups. A managed group is created by a particular node and is owned by that node. Other nodes need to explicitly join a managed group by sending a membership request to the group owner node.

Access to groups (of both peer and managed types) may optionally be restricted using a password, thereby preventing nodes that do not possess the necessary authorization credentials from joining a specific group.

The Group Manager API has been designed to be extremely simple and generic, facilitating its use in a wide range of applications. For example, the ACI Kernel uses the Group Manager to propagate node resource information, the Service Manager uses it to publish services in XML and search for services using XPath queries, and FlexFeed uses the Group Manager to find data sources for subscribers. Versions of the Group Manager are available for both Java and C++, and operating at the application layer facilitates easy, piecewise integration into existing applications. All of the above features combine together to make the Group Manager well suited to MANET environments. More information about the Group Manager component is available in [3].

## IV. AGSERVE

The AgServe component supports a Service-oriented Architecture on top of the Agile Computing Middleware. AgServe supports dynamic definition, instantiation, invocation, relocation, and termination of services. The underlying middleware monitors service resource utilization, invocation patterns, and network and node resource availability and uses that information to determine optimal initial placement and subsequent migration of services.

AgServe supports dynamic deployment of services by exploiting mobile code. Clients may define new services dynamically, thereby injecting new capabilities into the network. These services are packaged as self-contained archives that include all the necessary Java class files and other related JAR files. These service archives are dynamically distributed (pushed) to other nodes using the ACI Kernel functions for remote service installation. Policies may be used to control and manage service implementations as required for security purposes.

AgServe also allows activating services on remote nodes. Service activation is normally triggered by application-driven service invocation requests. AgServe takes advantage of the ACI Kernel functions to instantiate the specified service on the target node. Once activated, the service is identified by a unique identifier that is returned to the client stub, which is subsequently used for service invocation.

In addition, AgServe extends the ACI Kernel service container functions to provide transparent service migration. A service running inside the service container can be asynchronously stopped, its execution state captured, moved to a new service container (usually on a different node), and then restarted. This migration process is transparent to both the service itself and the client utilizing the service.

AgServe can take advantage of two Java-compatible ACI Kernel service containers, respectively based on the Aroma VM [13] and the JikesRVM [14], which support transparent service migration for services implemented in Java. The JikesRVM container uses Mobile JikesRVM [15], an enhanced version of the IBM JikesRVM with the ability to capture the execution state of Java threads and re-establish it transparently on another node. The execution of the service will then continue at the very next bytecode instruction on the new node.

The ACI Kernel builds a resource utilization profile for each service, keeping track of the CPU utilization and of the bytes sent and received over the network for each invocation. The ACI Kernel includes a coordination component that uses this resource profile information, along with the resource availability information from other nodes, to migrate service instances between nodes. In particular, it exploits a heuristic coordination algorithm to perform service (re)allocation on a continuous basis. The checkpointing capabilities of Aroma and Mobile JikesRVM, combined with the endpoint migration capability of Mockets, allows service instances to be migrated in a manner completely transparent to the clients, even in the midst of a service invocation.

Service migration differentiates AgServe from other SoAs. Service migration allows AgServe to react to environmental changes and is crucial to realize the goal of agility. For example, as nodes with free resources become available, service instances might be migrated from heavily loaded server nodes to other nodes, thereby improving the overall performance of the system (self-optimizing behavior). Service migration also allows the system to react to accidental events,

such as a power loss or an incoming attack (survivability behavior).

AgServe uses the Service Manager functions to support dynamic service definition, registration, lookup, and invocation. In fact, the dynamic activation and migration of service instances raises the need to augment traditional service descriptions with meta-information such as location of service implementation code, resource utilization profile, and the communications profile. Currently, each ACI Kernel maintains a database of services and their resource utilization information. While this aspect has not yet been realized, the resource utilization profile and other meta information could be embedded into the WSDL description for the service and packaged as part of the service archive. In addition, service registration, lookup, and invocation operations have been modified in order to support dynamic changes in service locations. More information about AgServe can be found in [4].

## V. EXPERIMENTAL RESULTS

Three experiments were conducted to evaluate the performance of the agile computing middleware. The first two experiments compare the service discovery capability in the Group Manager component with JXTA. The third experiment evaluates the service migration capability provided by AgServe.

### *A. Service Discovery Experiments*

The first experiment used a scenario with three nodes in an ad-hoc network to perform the evaluation of service discovery. The scenario is based around two UAVs and one ground vehicle, as shown in Figure 2. The two UAVs are providing a service that the client on the ground wishes to use. The UAVs advertise the service periodically and the client looks up the service periodically. As the UAVs move, they come into and go out of communications range with the ground vehicle.

The independent variables in this first experiment were the connectivity between the three nodes, the time interval between service advertisements by UAV1 and UAV2, and the time interval between service lookups by the client. The dependent variables were the ability for the client to discover the service and the bandwidth utilized. Since there was no discernible difference between JXTA and the Group Manager in terms of the client's ability to discover the service, only the bandwidth utilization results are reported.

The experiment was set up using three machines running NISTNet [16] to simulate the connectivity between the three nodes as the UAVs move. The UAVs complete one loop in 60 seconds. The UAV paths were synchronized in such a way that when UAV1 is closest to the ground vehicle, UAV2 is farthest from the ground vehicle. Note that in this setup, there is no connectivity between UAV1 and UAV2.

The connectivity between the UAVs and the client has been broken into four phases:

1. 40 seconds: the UAV is unreachable by the client (packet drop ratio set to 100%)

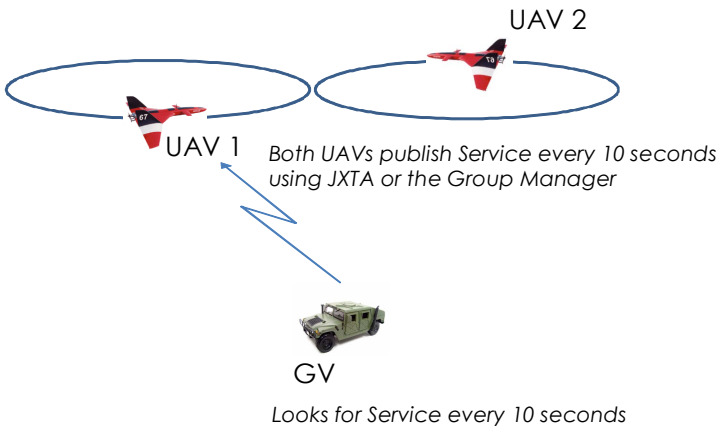


Figure 2: Scenario for Experiment One

2. 7 seconds: the UAV is approaching the area of maximum radio covering (packet drop ratio of 30%)
3. 6 seconds: the UAV is closest to the client (packet drop ratio 10%)
4. 7 seconds: the UAV is leaving the area of maximum radio covering (packet drop ratio of 30%)

JXTA has been configured in two different modes. In the first configuration, each node has been configured as a JXTA Rendezvous Peer, and therefore it can share its advertisements cache with other Rendezvous Peer nodes and propagate queries. In the second configuration, each node has been configured as a JXTA Edge Node and relies exclusively on multicast queries for service discovery. The Service Manager was used over the Group Manager for service matching. Peers running JXTA or the Group Manager have been configured without service caching.

The results are shown in Table 1. JXTA-RV is the configuration using Rendezvous Peers. JXTA-E is the configuration using only Edge Nodes. GM-PI is the default group manager configuration, with PING and INFO packets enabled. GM is with the PING and INFO packets disabled. The PING packet is used to detect node loss / disconnection, whereas the INFO packet is used to proactively push data. Neither of these capabilities is provided by JXTA, so it would be reasonable to disable them in the Group Manager for a fair comparison.

The results show the network traffic in bytes per second introduced by service discovery operations in each of the four configurations. Each node may generate traffic that is multicast to all the other nodes (shown in the multicast column) or addressed to a specific target node. The totals indicate the overall bandwidth utilization. The results show that JXTA uses 2.98 to 18.84 times more bandwidth than the Group Manager. In particular, in the best-suited configuration modes for this scenario, GM and JXTA-E respectively, JXTA uses 6.94 times more bandwidth than the Group Manager.

The second experiment used a scenario with five nodes instead of three – two UAVs and three ground vehicles. Also, in this scenario, the UAVs were the clients looking for services while the ground vehicles were the servers

advertising the services. The ground vehicles advertised the services at different times, in order to vary the results obtained by the UAVs when they queried for a service. In particular, GV1 advertised the service every 10 seconds. GV2 advertised a service for 10 seconds and stopped for 3 seconds before repeating the cycle. GV3 advertised a service for 10 seconds and stopped for 5 seconds before repeating the cycle. The UAVs queried for the service every 10 seconds.

TABLE 1
THREE NODE JXTA – GROUP MANAGER COMPARISON

| System/ | Destination | | | |
|---|---|---|---|---|
| **Sender** | Multicast | GV | UAV1 | UAV2 |
| **JXTA-RV** | | | | |
| GV | -- | -- | 731 | 581 |
| UAV1 | -- | 1817 | -- | -- |
| UAV2 | -- | 1206 | -- | -- |
| **Total** | **4335** | | | |
| **JXTA-E** | | | | |
| GV | 430 | -- | 20 | 12 |
| UAV1 | 188 | 370 | -- | -- |
| UAV2 | 188 | 389 | -- | -- |
| **Total** | **1597** | | | |
| **GM-PI** | | | | |
| GV | 183 | -- | -- | -- |
| UAV1 | 103 | 73 | -- | -- |
| UAV2 | 103 | 73 | -- | -- |
| **Total** | **535** | | | |
| **GM** | | | | |
| GV | 84 | -- | -- | -- |
| UAV1 | -- | 73 | -- | -- |
| UAV2 | -- | 73 | -- | -- |
| **Total** | **230** | | | |

Bandwidth Utilization in Bytes per Second

The connectivity between the UAVs and the ground vehicles was the same as before – no connection for 40 seconds, followed by 7 seconds with 30% packet loss, 6 seconds with 10% packet loss, and another 7 seconds with 30% packet loss. There was no connectivity between UAV1 and UAV2 while there was continuous connectivity between the three ground vehicles.

The results are shown in Table 2. The four configurations, JXTA-RV, JXTA-E, GM-PI, and GM are the same as before. The results show the network traffic in bytes per second, both multicast traffic generated from each node as well as point-to-point traffic between any two nodes. Once again, the Group Manager significantly outperforms JXTA, with JXTA using 2.36 to 49.17 times more bandwidth. Again, using the best possible configuration for each component, JXTA uses 9.55 times more bandwidth.

While these two experiments provide only two data points, it is still worth looking at the trend in terms of bandwidth utilization when increasing the size of the network from three to five nodes. Figure 3 plots the bandwidth utilization for all four configurations. As the figure indicates, the bandwidth utilization increases substantially for JXTA in Rendezvous mode. Even JXTA in Edge mode shows a larger rate of increase than either of the Group Manager configurations. With the Group Manager in the optimal configuration, there is a very small increase in the bandwidth utilization.

TABLE 2
FIVE NODE JXTA – GROUP MANAGER COMPARISON

| System/ | Destination | | | | | |
|---|---|---|---|---|---|---|
| **Sender** | Multicast | GV1 | GV2 | GV3 | UAV1 | UAV2 |
| **JXTA-RV** | | | | | | |
| GV1 | 28 | -- | 762 | 1854 | 325 | 118 |
| GV2 | 10 | 638 | -- | 636 | 214 | 130 |
| GV3 | 214 | 4153 | 738 | -- | 303 | 972 |
| UAV1 | 206 | 130 | 385 | 299 | -- | -- |
| UAV2 | 215 | 104 | 77 | 522 | -- | -- |
| **Total** | **13031** | | | | | |
| **JXTA-E** | | | | | | |
| GV1 | 187 | -- | -- | -- | 19 | 25 |
| GV2 | 188 | -- | -- | -- | 205 | 243 |
| GV3 | 430 | -- | -- | -- | 20 | 318 |
| UAV1 | -- | 6 | -- | -- | -- | 430 |
| UAV2 | -- | 17 | 12 | -- | -- | 430 |
| **Total** | **2531** | | | | | |
| **GM-PI** | | | | | | |
| GV1 | 103 | -- | -- | -- | 73 | 72 |
| GV2 | 103 | -- | -- | -- | 58 | 103 |
| GV3 | 103 | -- | -- | -- | 49 | 49 |
| UAV1 | 177 | -- | -- | -- | -- | -- |
| UAV2 | 181 | -- | -- | -- | -- | -- |
| **Total** | **1071** | | | | | |
| **GM** | | | | | | |
| GV1 | -- | -- | -- | -- | 73 | 73 |
| GV2 | -- | -- | -- | -- | 58 | 58 |
| GV3 | -- | -- | -- | -- | 50 | 49 |
| UAV1 | 83 | -- | -- | -- | -- | -- |
| UAV2 | 83 | -- | -- | -- | -- | -- |
| **Total** | **265** | | | | | |

Bandwidth Utilization in Bytes per Second

As part of future work, we intend to measure the performance with additional nodes to develop a more accurate trend for bandwidth utilization as a function of the number of nodes.

### *B. Service Migration Experiment*

The third experiment shows the benefits of opportunistically taking advantage of transient resources and measures the overhead of service migration. Figure 4 shows the experimental scenario, which consists of a number of client nodes, and a smaller number of servers and transient nodes (i.e., nodes that become available for short periods of

time during which their resources are opportunistically exploited).

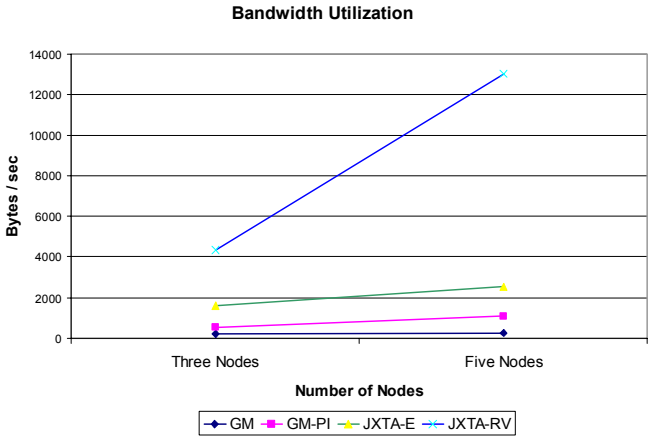


Figure 3: Bandwidth Utilization Trend

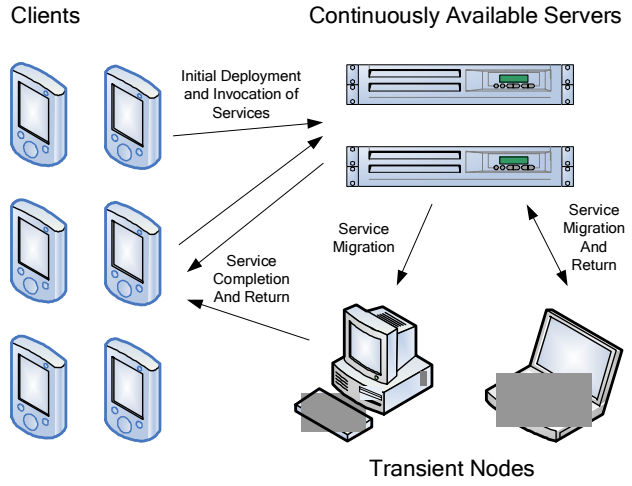


Figure 4: Service Migration Scenario

For this experiment, two client nodes, one server node, and one transient node were utilized. The two client nodes continuously invoke a CPU intensive factorization service, thereby overloading the only available server node. When a transient node becomes available, the middleware detects its presence and transparently migrates one of the service instances to the transient node. When the transient node is about to become unavailable (e.g., being shutdown), the service is migrated back to the server node. Note that the shutdown of the node is recognized by the middleware by means of a shutdown handler hook installed in the OS.

A transient resource may become unavailable in one of the following three ways:

- The node is undergoing a controlled shutdown, because some critical condition has been detected. In this case, the service migration must be as fast as possible to achieve survivability.
- The node transitions from a previously idle state to a busy state, e.g. because the owner of the node is now utilizing it. The resource is thus no longer free and the service must return back to a server node or move to another idle transient node.
- The quality of the network link (e.g. wireless) used by the node is degrading with respect to its past experience level, as reported by the XLayer framework. The service should then migrate immediately to keep on serving the client from another "more reachable" node.

Currently, an abrupt failure of the node or the communications link to the node is fatal to the service and the middleware does not support recovery from such an occurrence.

Table 3 shows the results of the service migration on the performance of the service from the client's perspective. The independent variable in this experiment is the duration of time for which the transient node is available. The dependent variables are the turnaround times for the service invocation from the two clients.

TABLE 3
SERVICE MIGRATION PERFORMANCE RESULTS

| Transient Node | Service #1 | Service #2 | Overall |
|---|---|---|---|
| Time (ms) | (Execution Time in ms) | | |
| 0 | 98362 | 98570 | 98570 |
| 5000 | 99421 | 100171 | 100171 |
| 10000 | 96683 | 97337 | 97337 |
| 15000 | 92019 | 93443 | 93443 |
| 20000 | 88990 | 89203 | 89203 |
| 30000 | 79324 | 81220 | 81220 |
| 40000 | 75322 | 78208 | 78208 |
| 50000 | 64870 | 70686 | 70686 |
| 60000 | 58614 | 66706 | 66706 |
| 70000 | 58218 | 64826 | 64826 |
| Baseline time for service execution - 54760 ms | | | |

Service Execution Time in case of Service Migration to a Transient Node

The time to execute the service, if there is no other load on the server node, is 54,760 ms. If two clients simultaneously execute the service and there is no transient node to exploit, the overall time for both clients to finish is 98,570 ms. When the transient node is available for 5 seconds, the overall time actually becomes worse (100,171 ms), which is an indication of the overhead of service migration. However, as the availability of the transient node increases, the overall performance improves. The break-even point is around 8 seconds, which is a measure of the overall agility of the system. It is the shortest length of time for which an opportunistic node may be exploited without any performance degradation. If a transient node is available for a longer length of time, the middleware shows a positive performance improvement. The ACI kernel decides whether a transient node is a suitable candidate for service migration using a

TABLE 4
OVERHEAD FOR DIFFERENT STAGES OF SERVICE MIGRATION

| Run | Detection | Local Coord. | RVM Capture | Packaging Dime | Transfer | Rem. Coord. | RVM Restore | Total |
|---|---|---|---|---|---|---|---|---|
| 1 | 53 | 1 | 192 | 5 | 16 | 2 | 240 | 509 |
| 2 | 31 | 1 | 213 | 3 | 19 | 2 | 241 | 510 |
| 3 | 702 | 1 | 264 | 5 | 21 | 2 | 218 | 1213 |
| 4 | 466 | 1 | 128 | 6 | 23 | 2 | 237 | 863 |

All Times in Milliseconds

TABLE 5
OVERHEAD FOR DIFFERENT STAGES OF SERVICE MIGRATION AS A PERCENTAGE OF TOTAL TIME

| Run | Detection | Local Coord. | RVM Capture | Packaging Dime | Transfer | Rem. Coord. | RVM Restore |
|---|---|---|---|---|---|---|---|
| 1 | 10.41 | 0.20 | 37.72 | 0.98 | 3.14 | 0.39 | 47.15 |
| 2 | 6.08 | 0.20 | 41.76 | 0.59 | 3.73 | 0.39 | 47.25 |
| 3 | 57.87 | 0.08 | 21.76 | 0.41 | 1.73 | 0.16 | 17.97 |
| 4 | 54.00 | 0.12 | 14.83 | 0.70 | 2.67 | 0.23 | 27.46 |

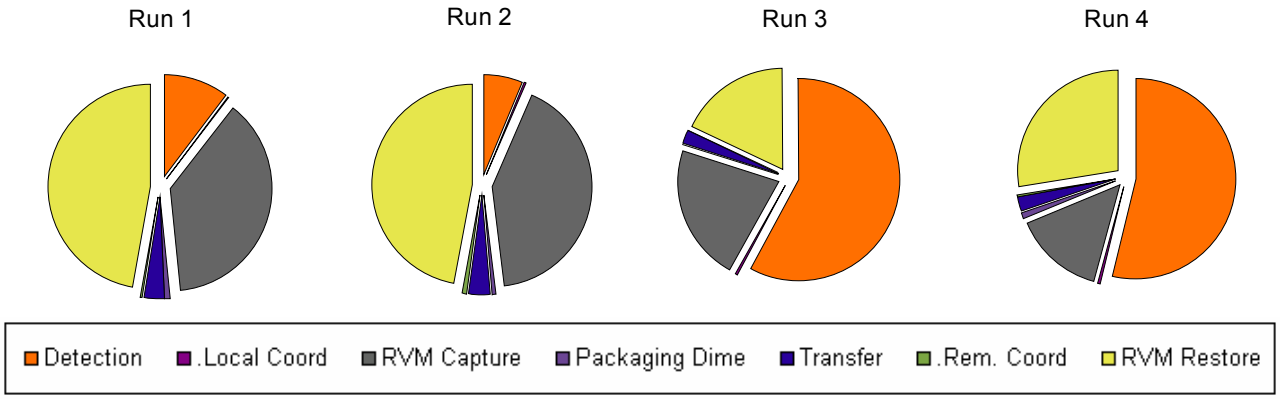


Figure 5: Visual Representation of Overhead for Different Stages of Service Migration

probabilistic model based on the past availability time of nodes and on the estimated execution time of the service.

Tables 4 and 5 and Figure 5 below provide details about the overhead costs of service migration. The overhead is characterized by the following operations:

- Time needed to detect a new node on the network (Detection time)
- Time needed by the server node to decide to migrate a service (Local Coordinator time)
- Time needed to capture the execution state of the service (RVM Capture time)
- Time needed to package the state as a DIME[2] message to send to the opportunistic node (DIME Packaging time)
- Time needed to transfer the DIME message (Transfer time)
- Time needed on the opportunistic node to receive and process the request to host the migrated service (Local Coordinator time 2)
- Time needed on the opportunistic node to restore and restart the service (RVM Restore time)

As the data indicates, the most variable component of the overhead (and the largest in some cases) is node detection. The proactive advertisement capability of the group manager is used to push information about resource availability at the opportunistic nodes. These nodes advertise via a periodic PING message generated by the Group Manager once every 1000ms. Therefore, the discovery time could range upto 1000ms, depending on when (within the advertisement cycle) the opportunistic node becomes reachable over the network. When an overloaded server node receives an advertisement from another node indicating resource availability, the server node triggers migration of a service, which includes the operations listed earlier.

[2] DIME (Direct Internet Message Encapsulation) is a proposed internet standard for the transfer of binary and other encapsulated data over SOAP or HTTP. A DIME message is composed of one or more DIME records. Each DIME record has a small header with meta-information that describes the record.

## VI. CONCLUSIONS AND FUTURE WORK

The experimental results presented in this paper show that the Agile Computing middleware enables the realization of SoA-based applications in the MANET environment, achieving good performance levels. The Group Manager service discovery mechanism is more efficient than JXTA. The service migration capability allows applications to effectively exploit transient computing resources in a dynamic, adaptive, and opportunistic fashion.

We are currently performing more extensive experimental analysis using different and more complex network topologies. We are also improving the performance of the state capture and restore operations of the middleware. We expect to conduct more experiments with the service migration to measure performance with a variety of services and with scalability. We will focus particularly on the question of the service state size. The size of the service state in Figure 5 is relatively small, but this may not be always true in more complex services. The migration time grows depending on the number of bytes of the service's state. For this reason, we are planning to explore using an analytical model, which calculates the expected benefit of migrating a service at a certain point in time. The latter calculation will exploit the Garbage Collector to determine the size of the service state in memory.

## ACKNOWLEDGEMENTS

This work is supported in part by the U.S. Army Research Laboratory under Cooperative Agreement W911NF-04-2-0013, by the U.S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0009, and by the Air Force Research Laboratory under Cooperative Agreement FA8750-06-2-0064.

## REFERENCES

[1] Halonen, T., Ojala, T., Cross-layer Design for Providing Service Oriented Architecture in a Mobile Ad Hoc Network. In Proceedings of the 5th International Conference on Mobile and Ubiquitous Multimedia, Stanford, CA, USA, 2006.

[2] Juszczyk, L., Lazowski, L., Dustdar, S., Web Service Discovery, Replication, and Synchronization in Ad-Hoc Networks. In Proceedings of the 1st International Conference on Availability, Reliability and Security (ARES'06), Vienna, Austria, April 2006.

[3] Suri, N., Rebeschini, M., Breedy, M., Carvalho, M., and Arguedas, M. Resource and Service Discovery in Wireless Ad-Hoc Networks with Agile Computing. In Proceedings of the 2006 IEEE Military Communications Conference (MILCOM 2006), October 2006, Washington, D.C.

[4] Suri, N., Rebeschini, M., Arguedas, M., Carvalho, M., Stabellini, S., and Breedy, M. Towards an Agile Computing Approach to Dynamic and Adaptive Service-Oriented Architectures. In Proceedings of the First IEEE Workshop on Autonomic Communication and Network Management (ACNM'07).

[5] Sun Microsystems JXTA Web Site: http://www.jxta.org/.

[6] Traversat, Bernard, Ahkil Arora, Mhoamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Sun Microsystems.

[7] Oliveira, R., Bernardo, L., Ruivo, N., and Pinto, P. Searching for PI resources on MANETs using JXTA. In Service Assurance with Partial and Intermittent Resources (SAPIR'05) at the Advanced Industrial Conference on Telecommunications.

[8] Tomarchio, O., Bisignano, M., Calvagna, A., and Di Modica, G. ExPeerience: A JXTA Middleware for Mobile Ad-Hoc Networks. In Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P2003).

[9] Suri, N., Bradshaw, J., Carvalho, M., Cowin, T., Breedy, M., Groth, P., and Saavedra, R., Agile Computing: Bridging the Gap between Grid Computing and Ad-hoc Peer-to-Peer Resource Sharing, in: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003).

[10] Tortonesi, M., Stefanelli, C., Suri, N., Arguedas, M., and Breedy, M. Mockets: A Novel Message-oriented Communication Middleware for the Wireless Internet, in Proceedings of International Conference on Wireless Information Networks and Systems (WINSYS 2006), Setúbal, Portugal, August 2006.

[11] Carvalho, M., Suri, N., Arguedas, M. (2005) Mobile Agent-based Communications Middleware for Data Streaming in the Battlefield. In Proceedings of the 2005 IEEE Military Communications Conference (MILCOM 2005), October 2005, Atlantic City, New Jersey.

[12] Carvalho, M., Suri, N., Arguedas, M., Rebeschini, M., and Breedy, M. A Cross-Layer Communications Framework for Tactical Environments. In Proceedings of the 2006 IEEE Military Communications Conference (MILCOM 2006), October 2006, Washington, D.C.

[13] Suri, N., Bradshaw, J.M., Breedy, M.R., Groth, P.T., Hill, G.A., and Saavedra, R. State Capture and Resource Control for Java: The Design and Implementation of the Aroma Virtual Machine. USENIX JVM 01 Conference Work in Progress Session.

[14] The Jikes Research Virtual Machine (RVM) project, at http://jikesrvm.org.

[15] Quitadamo, R., Cabri, G., and Leonardi, L. Enabling Java Mobile Computing on the IBM Jikes Research Virtual Machine. In Proceedings of The International Conference on the Principles and Practice of Programming in Java 2006 (PPPJ 2006). Mannheim, Germany.

[16] Carson, M. and Santay, D. NIST Net – A Linux-based Network Emulation Tool. In Computer Communication Review, June 2003.