Australian Government
Department of Defence
Defence Science and
Technology Organisation

Mobile Agents for Battlespace Information Exchange

Christos Sioutis and Yulin Ding

Air Operations Division
Defence Science and Technology Organisation

DSTO-TN-1179

ABSTRACT

This report provides an overview of Mobile Agent (MA) technology which is especially suited for use in networks with ad-hoc connectivity and fluid topology. This is still very much the case in Defence operations where consumer-level infrastructure is not available. The report provides an overview of MA characteristics and follows with a description of the implementation architecture of a specific MA framework. It then proposes their relevance in application to battlespace information exchange.

Published by

Air Operations Division
DSTO Defence Science and Technology Organisation
506 Lorimer St
Fishermans Bend, Victoria 3207 Australia

Telephone: 1300 DEFENCE
Fax: (03) 9626 7999

AR-015-614
May 2013

APPROVED FOR PUBLIC RELEASE

Mobile Agents for Battlespace Information Exchange

Executive Summary

An agent is an abstraction, or a concept that provides a convenient and powerful way to describe a complex software entity that is capable of autonomously accomplishing tasks on behalf of its owner. More specifically, a Mobile Agent (MA) is an agent which is able to migrate (move) from one computer to another and to continue its execution on the destination computer. MA technology was invented in a time when internet connectivity was not constantly available. The general use case for using an MA is to instantiate one, send it out to the world in order to achieve something and then disconnect from the network. The MA returns with the results when the user reconnects sometime in the future.

The research described in this report has been performed because in the current Australian battlespace there is no fixed infrastructure for constant network connectivity. Communication is typically ad-hoc, dropping in and out and with limited bandwidth. As a result, many critical tasks are still performed on the radio with no computer support. Other than a satellite link (which is only available to very few platforms) there is only one other (non-voice) way for military aircraft to communicate externally, a Tactical Data Link (TDL). TDLs have been around for many years and use standards to provide communication between platforms via radio waves.

This report proposes that MA technology could provide a means to introduce new information exchange paradigms and computer automation opportunities in the battlespace. It is proposed that MAs could be used to traverse over a TDL in order to achieve specific tasks like mission negotiation and remote data processing. Future directions of this work would initially involve development (or enhancement) of a MA context in order to make it able to interoperate with a TDL gateway. When this capability is available it will become possible to conduct experimentation to identify how MAs would be used to achieve the scenarios described in this report and/or other scenarios identified in consultation with ADF stakeholders.

This page is intentionally blank

Authors

Christos Sioutis
Air Operations Division

Dr Sioutis is a research scientist working in the Mission Systems Integration branch of Air Operations Division at DSTO. He is currently the Project S&T Advisor for JP2089 Ph3A (Tactical Information Exchange Domain Infrastructure) and leading the TDL research within AOD. Christos's research interests are focused in the areas of software architectures and middleware, artificial intelligence and tactical data links.

Yulin Ding
Air Operations Division

Yulin currently works on the Multi-TDL network and data link team for JP2089 project. She has been working on areas related to software architecture, simulation of data link and network management, agent technology, Human Machine Interface (HMI) and Service Oriented Architecture (SOA) in DSTO. Before joining DSTO, she worked as a senior research associate at The University of Adelaide. She has also worked in industry as a software engineer. She obtained her PhD and masters by research degree in computer science from University of Western Sydney and The University of Sydney respectively. She also obtained a master's degree in computer application and bachelor's degree in management engineering in China. Her research interests include software architecture and modelling, formal methods (model checking, model update), applied mathematics (classic logic, set theory, fuzzy logic, kinematics), mission system software architecture, data link, network management, agent technology, HMI and now SOA.

This page is intentionally blank

Contents

This page is intentionally blank

1. Introduction

This report describes exploratory research conducted under DSTO Task CIO 07/042 Tactical Information Exchange. The aim of this research is to investigate new technologies for realising information exchange within constraints posed by the Australian battlespace environment.

Mobile Agent (MA) technology was invented in a time when the internet for the most part was not always available; that is, people typically utilised slow dialup connections for a limited time. It was envisaged that MAs could be created offline, populated with data/instructions and "sent" to roam the internet when briefly connected. MAs would then autonomously gather information and coordinate activities (e.g. meetings, e-commerce transactions) on behalf of their owners. Sometime in the future owners would again connect to the internet and their MAs would return with results from their instructions.

Research applications with MA technology have included [EAKC05] network monitoring and management, information search and retrieval, integration with business services, intrusion detection for security, telecommunications and the military. With the advent of broadband communication (fixed and wireless) a typical consumer is now always connected to the internet. This means that it has become possible for internet services and client applications to assume or even require a constant internet connection which is especially prevalent with the latest trends of cloud computing and mobile apps. Constant connectivity has reduced the need and therefore research thrust behind MAs.

The research described in this report was performed because the military still work in environments where there is no fixed infrastructure for constant network connectivity. Communication is frequently ad-hoc, dropping in and out and with limited bandwidth. As a result, many critical tasks are still performed on the radio with no computer support. This report theorises that the current battlespace environment has very similar characteristics to the pre-broadband consumer world. Consequently the application of technologies like MAs could provide the means to introduce new information exchange paradigms and automation opportunities.

1.1 Relevance to Defence Operations

In the current Australian battlespace environment there is no fixed infrastructure for constant network connectivity. Communication is frequently ad-hoc, dropping in and out with a limited bandwidth. As a result, many critical tasks are still performed on the radio with no computer support. Approaches in bringing the cloud to the tactical edge [LEG11] have been considered, these however pose high associated costs which are a roadblock against wide implementation.

The battlespace environment has similar characteristics to the consumer pre-broadband environment. Additionally, due to the stringent security requirements imposed in the battlespace it is likely that this will remain the case for some time. Consequently the application of technologies like MAs could provide a means to introduce new information exchange paradigms and automation opportunities. That is at least until innovations allow for

the battlespace environment to catch up to today's consumer world in terms of network connectivity and bandwidth.

2. Literature Review

An Agent as an abstraction or a concept provides a convenient and powerful way to describe a complex software entity that is capable of autonomously accomplishing tasks on behalf of its owner. Agents are typically designed in terms of their desired behaviours as opposed to their methods and attributes. Being autonomous implies that they can operate without direct human (or other) intervention or guidance [WOO92]. Agents perform a service by either being reactive (ie. responding to changes in their environment) or proactive (ie. seeking to fulfil goals) [CZ98]. Finally, agents have social ability which means they are able to communicate and coordinate with other agents in order to achieve a given task. Depending on their intended application the following types of agents have been identified:

- intelligent agents: exhibiting aspects of artificial intelligence theories such as learning and reasoning,
- swarms: utilising large numbers of simple agents where the desired behaviour is achieved collectively, and
- mobile agents: agents that can relocate their execution onto different hosts.

2.1 General Concepts

An MA is a composition of computer software and data which is able to migrate (move) from one computer to another and to continue its execution on the destination computer. MAs are able to decide when and where to move. MA implementations need to comprise a life-cycle model, a computational model, a security model, a communication model and a navigation model. MAs are required to be implemented over a mobility framework that supports agent-related functions and additionally provide facilities for storage and retrieval of agents, instantiation, transfer and method invocation [BPL98].

Portability is fundamental because MAs should be able to move in heterogeneous networks between machines with different operating systems and hardware architectures in order to be really useful. In [BR05], characteristics of MAs are described as:

- MAs are typically used in wide area and heterogeneous networks in which no assumptions can be made concerning either the reliability of the connected computers or the security of the network connections.
- The MAs migration is initiated by the agent itself, in contrast to other systems where migration is initiated by the underlying operating system or middleware.
- Migration of MAs is performed to access resources available at other computers in the network.
- MAs are able to migrate more than once (ie. multi-hop ability). After an MA has visited the first host, it may choose to migrate further to other hosts to continue its task.

An attempt was made by the Mobile Agent System Interoperability Facility (MASIF) to standardise the definitions and interfaces of MA frameworks. They are defined using the

Interface Definition Language (IDL) and utilise the Common Object Request Broker Architecture (CORBA) standards. MASIF defines two interfaces `MAFAgentSystem` for MA management tasks and `MAFFinder` for MA discovery [MAL98].

AN MA framework is required to provide the programming constructs and also a multi-threaded execution environment which allows multiple MAs to be hosted and executed in parallel. The framework must control their execution and protect the underlying operating system and other services from unauthorized access. The major technical advantages of MAs are [BR05]:

Delegation of tasks: Instead of using computer systems as interactive tools that are able to work only under direct control by a user, autonomous MAs aim at undertaking entire tasks and working without constant control. As a result, the user can devote time and attention to other more important things.

Asynchronous processing: Once MAs have been initialized and set up for a specific task, they physically leave their owner's computer system and from then on migrate freely through a network. Only for this first migration must a network connection be established. This is more stable than a client-server architecture due to the independence from network connectivity.

Adaptable service interfaces: MAs can offer a chance to design a client driven interface that is optimized for the client user. The complex and user-driven interactions can then be translated into straight forward comprehensive requests with remote services.

Transfer of algorithms/behaviour: Instead of sending unprocessed data and making multiple requests, only MAs are transmitted. MAs encapsulate the required algorithms/behaviour to be performed remotely and only carry the associated results back. This reduces network traffic and saves time when operating on networks with high-latency and low bandwidth if the MA code is smaller than the data that must be processed.

2.2 Security Considerations

Security is a major concern when dealing with MAs from three perspectives. Firstly, MA hosts effectively hand over execution rights to a foreign program with unknown behaviours. Secondly, due to the fact that MAs are fully transferred to remote hosts they also offer themselves to the mercy of that host (e.g. they could be forcibly suspended and their contents inspected). Finally, the nature of MA systems encourages interaction which introduces a risk of malicious MAs hacking into other MAs whilst communicating. Some specific considerations in regards to security are [BR05]:

Authenticity: This is a major requirement and the foundation of many security solutions. It demands that each MA is able to prove their identity. Similarly, MAs must authenticate on each host in order to decide whether the MA is trusted.

Confidentiality: This demands that information contained within MAs is protected against unauthorised access. An example of this problem is when an MA is performing information gathering. At each stop the MA may be gathering confidential information which cannot be revealed to other hosts.

Integrity: This is necessary in order to ensure that information contained within an MA has not been modified at some point in time without detection. For example, on instantiation an MA could be provided with an itinerary of hosts it needs to visit. This itinerary must not be able to be changed otherwise the MA could be made to visit arbitrary hosts.

Accountability: This requires hosts and/or MAs to maintain a record of actions performed during a visit. If accountability is not maintained it becomes possible for MAs to take actions and to deny the responsibility of their effects.

Availability: This ensures that access to services required by MAs cannot be forcibly restrained. Conversely, it guarantees reliable and prompt access to data and resources for authorized MAs. Examples include malicious MAs causing problems to other MAs in a host and/or a malicious host refusing to let an MA migrate out of it.

Anonymity: This is the antithesis of authenticity however it is required in certain applications where hosts offer limited services to unauthenticated MAs. In such cases the owners of MAs may also wish to remain anonymous.

2.3 Applications in Network Management

Much of the research effort in MA technology occurred nearly a decade ago which was a booming time for the area. More recent research in MAs is combined with newer technologies such as web services [NG11] and distributed systems integration [SJ02].

Several groups have focused on the applicability of MAs for network management tasks. [RD99, RAL02, RAL03] described a network management system based on MAs. The authors compared the MA paradigm with client-server based approaches for typical management tasks, with regards to performance and network load. The response time results showed that MAs are less sensitive to the latency and the bandwidth of a bottleneck link that connects the management station to the managed hosts, but was more influenced by the task to be performed. In addition, the MA paradigm performed well depending on the following factors: a) the number of messages that traverse the bottleneck link, b) the incremental size of the MAs when returning to the management station after visiting a fixed number of nodes.

[KAL97] presented a design and implementation of an Intelligent Mobile Agent (IMA) framework for distributed network management. The authors delegated part of the management responsibility to the managed entity and used the response time as a measure to compare the performance of the MAs. The performance results indicated a significant improvement in response time for the tasks performed by the managed entity. When the

number of nodes was greater, the MA response time decreased compared to traditional methods like that of the Simple Network Management Protocol (SNMP).

[GAL99] presented a secure and fault tolerant management framework based on MAs, which addresses the limitations of traditional centralised network management by introducing two efficient, lightweight polling modes. Results indicate a significant improvement in both response time and traffic overhead when comparing the introduced polling modes to traditional centralised polling. The choice of transport protocol used for MA transfers has proven a critical factor regarding the polling modes' performance.

In [BAL05] the authors designed a two level architecture where MA-based servers within a sub-network were allowed to build and evolve the logical network dynamically. MAs enlisted at a central server called domain manager which was responsible for coordinating connectivity. The major usage of this approach was an enhanced ability to recover from failure situations. Specifically, if the connection to the domain manager was broken a new one was automatically elected. The author used the same toolkit as in [BR05] to implement its behaviour.

3. Implementation Overview

3.1 Software Characteristics

There are two kinds of mobility: strong and weak mobility. Both strong and weak mobility involve the transferring the MA's *runtime information* and *execution code*. Additionally, strong mobility includes the MA's *execution information*. The practical difference between the two is that with weak mobility execution effectively restarts and program flow always begins from a defined entry point method. Conversely strong mobility allows an MA to continue processing exactly from where it left off. Strong mobility is similar in concept to an operating system context switch where the execution information of a process (e.g. stack, program counter, registers) is stored such that the process can be resumed at the next scheduled slot.

3.1.1 Mobile Agent Elements

An MA has five basic elements [LO98]:

State: The state is defined as the runtime information gathered whilst an MA is executing. The state is what makes an MA unique with respect to other MAs as it encodes experiences from its travels.

Implementation: The business logic or instructions/code that the MA executes. This needs to be executable on all hosts that the MA visits. The easiest way to achieve this is to utilise scripting or interpreted languages that provide hardware abstraction (e.g. a virtual machine). Use of traditional programming languages which compile code down to specific assembly instructions are not preferred because it requires MAs to be locked down to a specific hardware architecture and set of supporting runtime libraries.

Interface: This exposes a set of methods which can be signalled by other MAs in order to communicate. The specific infrastructure and protocol used for communication depends on the chosen framework. It is preferred however that communication is based on an open standard like CORBA or Knowledge Query Manipulation Language (KQML), which allows MAs implemented through frameworks from different vendors to communicate.

Identifier: This provides a means to specifically identify an MA. When the MA identifier is coupled with a directory service (e.g. CORBA Name Service) then a specific MA can be contacted whilst located anywhere on a network.

Principals: This provides metadata that is specific to an MA instance. Examples include: the MA's implementation framework vendor and version; the owner of the MA (ie. the person/system who instantiated it); its purpose for instantiation.

3.1.2 Execution Contexts

The execution contexts are used to host MAs. They provide a means to instantiate new MAs as well as send and receive other MAs. They have four elements [LO98]:

Engine: The engine is vendor specific and has two main responsibilities. First, it provides the necessary resources required for MAs to execute effectively. Second, it provides safeguards to ensure MAs are hosted without compromising other operations and/or security.

Resources: An MA is instantiated by assigning certain resources to it. At the very least this includes allocations of memory to load and CPU time to execute. Additional resources include access to local data and/or network communication.

Location: This provides the concept of a uniquely addressable identifier of a context which MAs can travel to. One of the biggest advantages of mobile MAs comes from the fact that not all locations must be contactable from every other location. An MA will search the network of hosts until it finds one where its target destination is directly contactable.

Principals: Similarly to MAs, contexts can be populated with metadata that describe additional information. One example is the owner of the hosts upon which the context is available.

3.1.3 Transfer and Communication

A key feature provided by MA frameworks is the necessary processing and communication required for MAs to transfer between different hosts. The transfer process has multiple steps as shown below, items shown in bold are performed by the MA whilst the remaining items are performed by the supporting framework:

Suspend ➔ Serialise ➔ Encode ➔ Transfer ➔ Decode ➔ De-serialise ➔ **Resume**

The transfer process can be initiated by the MA itself or another entity within the network. Firstly, the MA is notified that the transfer process has been initiated and it is given the choice to deny the transfer or to continue with it. If the MA chooses to continue with the transfer it secondly performs any required processing to prepare for the transfer and suspends its execution, at this point the host context takes over. The MA is thirdly serialised into a persistent byte representation that can be stored and/or transferred. The serialised data are subsequently segmented into chunks, encoded within the required transport protocol and transferred over the network. Once the MA arrives at its destination the new context assembles the data, decodes it into its original form, de-serialises the MA then loads it into memory. Finally, the context adds the MA into its processing schedule and begins executing it when the CPU becomes available. In addition to transfer, MAs are able to communicate over the network. Communication in this context is simply an MA remotely calling a method from another MA's interface. The following types of communication operations can be used depending on the desired effect:

Synchronous: This is used when the MA needs to know if communication has succeeded. The calling operation blocks until a response is received. The response itself can simply be an indication that the communication has successfully completed at which point the MA's processing continues.

Asynchronous: This is used when the MA does not need to know if communication was successfully achieved. This type of communication is best suited for high volume periodic communication. That is, repeatedly sending updated data (e.g. monitoring a sensor) whereby such data becomes obsolete at the next update sent shortly afterwards.

Multi-point: This is a special case of asynchronous communication where the MA sends out a single message which is simultaneously received by multiple receivers. This approach is more efficient in comparison to sending a message to each MA individually.

3.2 Software Architecture

This section describes the software architecture of the *Aglets* [LO98] MA framework which is written in Java. It has been selected for this research because it provides a full working implementation and is open source.

The Aglets name is a composite word from *agent* and *applet*. This provides a hint to the fact that Aglets are designed based on the principles of Java applets. On reflection one can deduce that applets provide many of the features of MAs. This is true, except that applets provide a generic way for executing Java code on remote clients within very strict security restrictions. Aglets provides specific functionality and security functions specifically designed for MA applications. The Aglets framework was originally developed by IBM as a proprietary research system. It was subsequently made open source. Development had stagnated since

2002 with version 2.0.2 but has resumed in 2012 with version 2.5. The following sections detail our view of the Aglets architecture as described by Lange and Oshima [LO98].

One of the drawbacks in using Java as the language for a building an MA framework is that the JVM does not allow an application to explicitly access its runtime data (e.g. processing stack) and hence it becomes impossible to obtain the full execution state of an MA. Therefore, Aglets can offer only weak mobility whereby execution is restarted rather than resumed after every transfer. Another drawback is a lack of resource control by the JVM which means that MAs could potentially consume all host resources.

3.2.1 Core Constructs

The Aglets framework makes extensive use of the observer/listener pattern [GHJV95]. This is a software design pattern in which one software object maintains a list of other observer objects and notifies them about specified events during execution through calling one of their methods. These methods, termed *callbacks,* are typically pre-specified in an abstract listener interface. Applications provide object adaptors that implement the listener interface. By convention callback names typically begin with an "`on`" or "`handle`". For example, `Aglet.onCreation` is called when an MA is first created.

3.2.1.1 Aglet Class

A MA is defined by extending the `Aglet` class and building upon its methods. Code Listing 1 lists the commonly used subset of these methods. The `Aglet.onCreation` and `Aglet.onDisposing` callbacks are used for initialisation and destruction respectively. The `Aglet.run()` method is the execution entry point and should contain the main instructions that the MA performs. Other methods are provided for accessing elements of the MA's runtime infrastructure, dispatching custom events, receiving messages, as well as managing adapter objects for persistency, cloning and mobility.

```
1  public class Aglet {
2     //lifecycle callbacks
3     void onCreation(Object init)
4     void onDisposing()
5
6     //runtime callbacks
7     void run()
8     boolean handleMessage(Message message)
9
10    //runtime methods
11    AgletContext getAgletContext()
12    AgletProxy getProxy()
13    void dispatchEvent(AgletEvent ev)
14
15    //listener management methods
16    void addPersistencyListener(PersistencyListener listener)
17    void removePersistencyListener(PersistencyListener listener)
18    void addCloneListener(CloneListener listener)
19    void removeCloneListener(CloneListener listener)
20    void addMobilityListener(MobilityListener listener)
21    void removeMobilityListener(MobilityListener listener)
22 }
```

Code Listing 1: Aglet class

```
1  public interface AgletProxy {
2     //lifecycle methods
3     void activate()
4     Object clone()
5     void deactivate(long duration)
6     void dispose()
7
8     //mobility methods
9     AgletProxy dispatch(Ticket ticket)
10    String getAddress()
11
12    //aglet inspection methods
13    Aglet getAglet()
14    String getAgletClassName()
15    AgletID GetAgletID()
16    AgletInfo getAgletInfo()
17    boolean isActive()
18    boolean isRemote()
19    boolean isState(int type)
20    boolean isValid()
21
22    //communication methods
23    Object sendMessage(Message msg)
24    void sendOnewayMessage(Message msg) //async
25    FutureReply sendFutureMessage(Message msg)
26 }
```

Code Listing 2: AgletProxy Interface

3.2.1.2 AgletProxy Interface

As shown in Code Listing 2, the `AgletProxy` interface provides lifecycle methods; mobility methods; inspection methods and communication methods. MAs are required to use the `AgletProxy` interface instead of directly accessing objects in memory; there are two reasons for this:

a) It provides a security layer whereby all accesses are consulted against a security management service to ensure that they are allowable.
b) It keeps track of the MA as it migrates around the network and consequently offers location transparency. When an MA is located on a remote host the proxy will forward requests and return the result over the network.

MAs communicate by exchanging `Message` objects which are sent using the `AgletProxy` interface. The `AgletProxy.sendMessage()` method implements synchronous communication as described in section 3.1.3. MA execution halts at this point until a response is received. On the other side the MA receives a `Aglet.handleMessage()` callback with the relevant message. The return value of this callback is a Boolean that indicates if the message was successfully handled.

Conversely, the methods `AgletProxy.sendOnewayMessage()` and `AgletProxy.sendFutureMessage()`are used to send messages asynchronously allowing the originating MA to continue execution. The latter method utilises a `FutureReply` object which is filled out by the remote MA sometime in the future and is returned to the originating MA with the result upon request.

3.2.1.3 AgletContext Interface

The context is a special software container where MAs can be created, execute and be disposed. Moving MAs hence means transferring them between contexts. The `AgletContext` interface, shown in Code Listing 3, provides ways for MAs to query information about the context they are operating within; modify aspects of the context; obtain proxies to other MAs; and even to create new MAs.

3.2.2 Event Model

Aglets employs an event model to notify MAs when certain things have occurred. MAs can make use of the event model by overriding the relevant callback methods with new implementations. The `Aglet` class itself provides lifecycle callbacks whilst cloning, mobility and persistence callbacks are provided by listener interfaces as shown in Code Listing 4. MAs keep track of their state through handling different combinations of the event model callbacks as described as follows.

```
1  public interface AgletContext {
2     //context inspection methods
3     void addContextListener(ContextListener listener)
4     void removeContextListener(ContextListener listener)
5     URL getHostingURL()
6     String getName()
7     void setProperty(String key, Object value)
8     Object getProperty(String key)
9
10    //aglet control methods
11    AgletProxy createAglet(URL codeBase, String code, Object init)
12    Enumeration getAgletProxies()
13    AgletProxy getAgletProxy(AgletID id)
14    ReplySet multicastMessage(Message msg)
15    AgletProxy retractAglet(URL url, AgletID aid)
16 }
```

Code Listing 3: AgletContext Interface

```
1  public interface CloneListener {
2     void onClone(CloneEvent event)
3     void onCloning(CloneEvent event)
4     void onCloned(CloneEvent event)
5  }
6
7  public interface MobilityListener {
8     void onDispatching(MobilityEvent event)
9     void onArrival(MobilityEvent event)
10    void onReverting(MobilityEvent event)
11 }
12
13 public interface PersistenceListener {
14    void onActivation(PersistencyEvent event)
15    void onDeactivation(PersistencyEvent event)
16 }
```

Code Listing 4: Event Model Listener Interfaces

```
1  EnterpriseQueryAdaptor implements MobilityListener {
2     onArrival() {
3        if(localContext.equals(EnterpriseContext)){
4           getAgletContext().getEnterpriseHandle().performSomeQuery()
5           dispatch(MyContext)
6        } else {
7           getAgletContext().getMyApplicationHandle().notifyResults()
8           dispose()
9        }
10    }
11 }
12
13 MyAglet extends Aglet {
14    onCreation(){
15       addMobilityListener(new EnterpriseQueryAdaptor)
16       dispatch(EnterpriseContext)
17    }
18 }
```

Code Listing 5: Enterprise Query Pseudocode

Creation

- External application (or another MA) calls `AgletContext.createAglet()`
- MA constructor is called on instantiation
- New MA receives callback `Aglet.onCreation()`
- New MA receives callback `Aglet.run()` to start normal execution

Disposal

- MA calls `AgletProxy.dispose()` on its proxy
- MA receives callback `Aglet.onDispose()` to perform final processing
- Execution is halted
- MA is destroyed

Cloning

- MA calls `AgletProxy.clone()`
- MA receives callback `CloneListener.onCloning()` when about to be cloned
- Framework clones the MA
- Original MA receives callback `CloneListener.onCloned()`
- Clone of MA receives callback `CloneListener.onClone()`

Persistence

- MA calls `AgletProxy.deactivate()`
- MA receives callback `PersistenceListener.onDeactivation()`
- Framework deactivates the MA and stores it
- Another MA or application calls `AgletProxy.activate()`
- Frameworks retrieves MA from storage and activates it
- MA receives callback `PersistenceListener.onActivation()`

Mobility

- MA calls `AgletProxy.dispatch()` with required destination context
- MA receives callback `MobilityListener.onDispatching()` when about to be transferred
- Framework transfers the MA
- MA receives callback `MobilityListener.onArrival()` when instantiated in remote host context
- MA receives callback `Aglet.run()`to resume execution

The pseudo-code shown in Code Listing 5 illustrates how an MA is implemented that travels to a remote context in order to query an enterprise system and then returns to provide the result. A specific point of interest is that MA does not make use of the `run()` method because it does not perform any persistent processing.

4. Application in Battlespace Information Exchange

MA technology has already been considered as one option to enhance the utility of systems that enable network-centric warfare [CER01]. The vision in this case is that MAs are used to provide assistance in command and control, intelligence acquisition and tactical information dissemination. MAs could potentially switch their operating context from a real-time tactical domain (ie. troops, planes and ships) to the Web/enterprise domain (ie. command centres, enterprise facilities). Example scenarios of this include:

Tactical domain: A troop party on the ground can instantiate an MA and push it to a nearby aircraft/UAV. The MA would make its way to a distant ship and/or command centre to exchange information about the mission progress. It would then return to its owner to report any new information and/or updated mission parameters.

Enterprise domain: Upon detection of an unknown entity a tactical aircraft can instantiate an MA and send it to a command centre with access to enterprise military intelligence systems. The MA could exchange information with other MAs and collectively perform a range of queries on intelligence databases in order to identify the unknown entity. The MA would then return to the originating aircraft and offer its new information for fusion into the mission system.

4.1 Mobility over Tactical Data Links

The enterprise domain example in the previous section mentions a tactical aircraft sending and receiving MAs. For this to occur there must be some sort of connectivity between aircraft and/or the ground. Other than a satellite link (which is only available to very few platforms) there is only one other way at present for military aircraft to communicate externally; a Tactical Data Link (TDL). TDLs have been around for many years and provide communication between platforms via radio waves.

The most widely used TDL for military aircraft currently is Link 16 (L16). One of the limitations of L16 is that due to the frequencies it uses for transmissions its range is limited to within Line-Of-Sight (LOS) between platforms. This limitation requires that some platforms must be configured to provide a *relay* function in order to establish end-to-end connectivity. Such platforms will effectively re-transmit information they receive therefore pushing it further along the network.

A mock-up example of a L16 network is illustrated in Figure 1. The icons illustrate platform locations of which the shape indicates their environment. It is important to also understand that the network topology is not static. As aircraft and ships move around at different speeds the topology and hence connectivity of the network will constantly change. L16 will automatically maintain a cohesive network as long as all platforms are within LOS of their neighbour. It is also possible for the network to be segmented if a relay platform drops out until it reconnects and/or another platform takes its place.

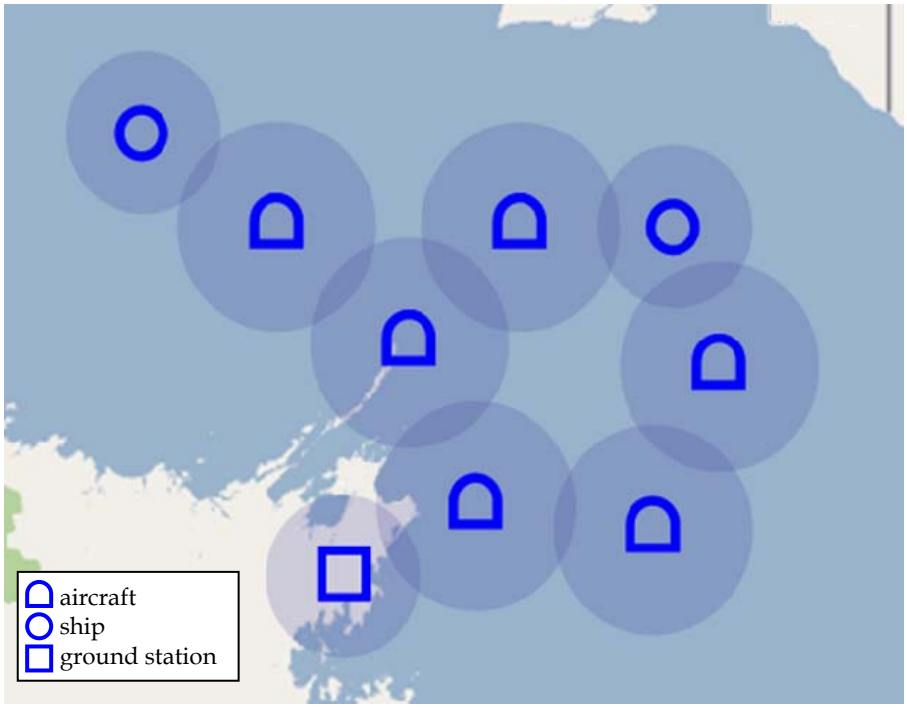

Figure 1: Mock-up of a Link 16 Tactical Data Link Network

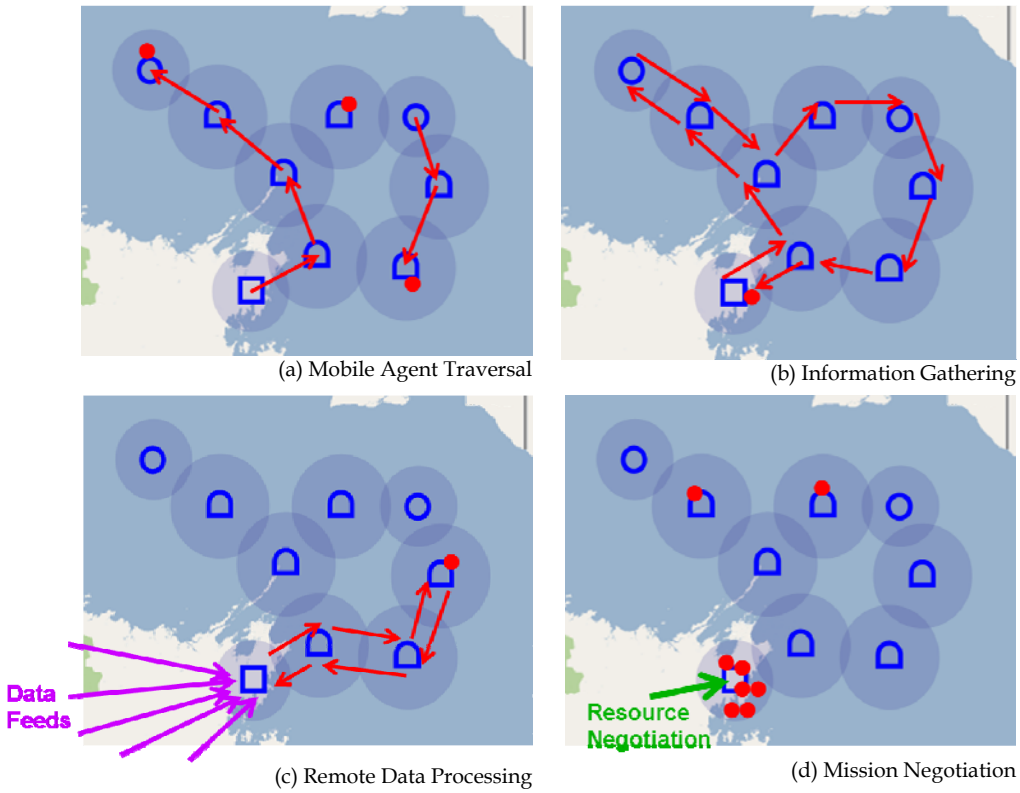

Figure 2: Mobile Agents through Tactical Data Links

L16 information is encoded into a number of fixed (J-series) messages which are broadly grouped based on their function, for example: Participant Location and Identification, Surveillance, Command & Control. Of particular interest in this context is a message called *J28.2-FreeText*. This is a variable length message which does not have a predefined structure. The payload of this message is simply a sequence of octets. It is proposed that MAs could be encoded into a String and transmitted through L16 using such messages. Figure 2 provides a number of examples of how MAs could be envisaged to operate over a L16 network. The red dots illustrate MAs instantiated within platforms and the red arrows illustrate the path MAs have followed within the network.

Mobile Agent Traversal: Illustrates three instantiated MAs, two of which have moved through the network and are executing on remote platforms.

Information Gathering: Illustrates an information gathering example. The ground node has instantiated an MA and sent it out into the network with the goal to execute on each and every platform in order to gather specific information from the operators of each platform.

Remote Data Processing: Illustrates a scenario whereby the ground station has access to a number of high-speed data feeds from relevant enterprise systems. One of the tactical platforms requires some information and sends an MA over to the ground station in order to query the enterprise data feeds and to consolidate the required information. The MA then returns and provides the results.

Mission Negotiation: The last example illustrates a scenario where there is a need to update mission parameters. In this case the ground station provides a *meeting place* for MAs to converge and engage with the mission commander on how to proceed sharing any relevant information. The MAs could then return and provide the updated mission parameters to each platform.

5. Conclusions and Future Work

The MA technology was specifically designed for achieving tasks over networks with low bandwidth ad-hoc connectivity. A general use case for using an MA is to configure one offline, send it out to achieve something when briefly connected and obtain the results sometime in the future. It is proposed that MAs could be used to traverse over a L16 TDL in order to achieve specific tasks like mission negotiation and remote data processing. Implementing MAs requires heavy use of an underlying framework to provide specialised resources and functions as well as events to notify MAs as they encounter different situations. Future directions of this work would initially involve development (or enhancement) of an MA context in order to make it able to interoperate with a L16 TDL gateway. When this capability is available it will become possible to conduct experimentation to identify how MAs would be used to achieve the TDL scenarios described in this report and/or other scenarios identified in consultation with ADF stakeholders.

6. References

[BAL05] Peter Braun et al. (2005). *Multi-agent approach to management a network of mobile agent servers*. Informatica 29(2005) 111-121.

[BPL98] Andrzej Bieszczad, Bernard Pagurek and Tony White. (1998). *Mobile agents for network management*. IEEE communications surveys, Vol. 1 No.1.

[BR05] Peter Braun and Wilhelm Rossak. (2005). *Mobile Agents – Basic concepts, Mobility models & the Tracy toolkit*. Elsevier Inc. Pp.441.

[CER01] Marion Ceruti (2011), *Mobile Agents in Network-Centric Warfare*, IEICE Transactions on Communications, Vol.E84-B No.10.

[CZ98] William R Cockayne and Michael Zyda. (1998) *Mobile Agents*. Manning Publications. Manning publications. Pp.312

[EAKC05] Mohamad Eid, Hassan Artail, Ayman Kayssi, and Ali Chehab (2005). *Trends in mobile agent application*. Journal of Re-search and Practice in Information Technology, 37(4).

[GAL99] D. Gavalas, D. Greenwood, M. Ghanbari and M. O'Mahony. (1999). *Using Mobile Agents for Distributed Network Performance Management*, In Proc.of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99), LNCS Vol. 1699, Springer-Verlag, Albayrak, S. (Ed.), Pp. 99-115, Stockholm, Sweden, 9-11 August 1999. [LEG11] Mary Legere (2011), *Land ISRNet Path to Network Centric Warfare*, presented at Network Centric Warfare Conference, Canberra.

[GHJV95] E. Gemma, R. Helm, R. Johnson, R. Vlissides (1995), *Design Patterns, elements of reusable software architecture,* Addison Wesley, Massachusetts.

[KAL97] Hosoon Ku, Gottfried W.R.Luderer and Baranitharan Subbiah. (1997). *An Intelligent Mobile Agent Framework for Distributed Network Management*, In proc. Of Global Telecommunications Conference (GLOBECOM'97). Vol.1 Pp.160-164. IEEE

[LO98] Danny Lange and Mitsuru Oshima (1998), *Programming and Deploying Java Mobile Aglets with Aglets*, Addison Wesley Longman, Massachusetts.

[MAL98] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. *MASIF: The OMG Mobile Agent System Interoperability Facility*. In Proceedings of the Second International Workshop on Mobile Agents, volume 1477 of Lecture Notes in Computer Science, pages 50–67, Stuttgart, Germany, September 1998. Springer-Verlag.

[NG11] Mydhili K. Nair and V. Gopalakrishna. (2011). *Applying Web Services With Mobile Agents for Computer Network Management*. International journal of computer networks and communications (IJCNC) Vol.3 No.2, Pp.125-143, March, 2011

[RAL02] Marcelo G. Rubinstein et al. (2002). *Evaluating the performance of a network management application based on mobile agents*. In proc. Of the 2nd International IFIP-TC6 Networking conference: Networking Technologies, Services, and Protocols, Performance of Computer and Communication Networks and Mobile and Wireless Communications (Networking 2002), Pisa Italy, Vol. 2345 Pp.515-526. Springer0Verlag.

[RAL03] Marcelo G. Rubinstein et al. *Scalability of a network management application based on mobile agents*. Journal of Communication and Networks, IEEE/Korean Institute of Communications Science (KICS), 5(3):240-248.

[RD99] Marcelo G. Rubinstein & Otto C. M. B. Duarte. (1999). *Evaluating tradeoffs of mobile agents in network management*. Networking and Information Systems Journal. 2(2):237-252.

[SJ02] Nils P. Sudmann & Dag Johansen. (2002). *Software deployment using mobile agents*. In procs. Of IFIP/ACM Working conference on component deployment (CD 2002), Berlin Germany. Vol. 2370 Pp.97-107. Springer-Verlag.

[WOO92] Michael Wooldridge. (1992). *The Logical Modelling of Computational Multi-agent Systems*. Ph.D Thesis, The University of Manchester

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Mobile Agents for Battlespace Information Exchange

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Christos Sioutis and Yulin Ding

5. CORPORATE AUTHOR

DSTO Defence Science and Technology Organisation
506 Lorimer St
Fishermans Bend Victoria 3207 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TN-1179	AR-015-612	Technical Note	May 2013

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
2012/1227872/1	CIO 07/042	A/DGEA (CIO)	17	17

DSTO Publications Repository

http://dspace.dsto.defence.gov.au/dspace/

14. RELEASE AUTHORITY

Chief, Air Operations Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DSTO RESEARCH LIBRARY THESAURUS

Agents, Tactical Data Links, Information.

19. ABSTRACT

This report provides an overview of Mobile Agent (MA) technology which is especially suited for use in networks with ad-hoc connectivity and fluid topology. This is still very much the case in Defence operations where consumer-level infrastructure is not available. The report provides an overview of MA characteristics and follows with a description of the implementation architecture of a specific MA framework. It then proposes their relevance in application to battlespace information exchange.