APM

FollowMe

Comparison of autonomous mobile agent technologies

Takanori Ugai ,Michael Bursell

Abstract

Comparison between four different Java-based software agent technologies - Aglets (IBM), Kafka (Fujitsu) , Voyager (ObjectSpace) and Odyssey (General Magic)

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE CB3 0RD UNITED KINGDOM

+44 1223 515010 • Fax +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

Distribution:
Supersedes:
Superseded by:

1989.02.00

Approved
Technical Report

10 October, 1997

TABLE OF CONTENTS

I. AGENTS

A. Agents background

Agent technology is a growing area of interest and research, and a number of competing technologies are currently making their presence felt in the field. In fact, there are a number of different definition of what constitutes an agent. This paper is concerned with *software agents*, and, more specifically, those that can be mobile - that is, move between execution environments on different host machines on a network - and autonomous - that is, have some control over their execution and movement.

We adopt, for the purposes of this document, part of the definition proposed by OMG CFTF RFP3 in *Section B2.4 Technical Issues*.

> "Agents are small to medium grained objects, they can move, and they can start or stop their execution autonomously."

Narrowing the field of interest still further, the discussion will focus on four agent technologies which are implemented in the Java language. These four technologies are - **Aglets** (from IBM Japan), **Kafka** (from Fujitsu, formally called CHOAS), **Voyager** (from ObjectSpace) and **Odyssey** (from General Magic) The reasons for the choice of these four specifically are -

1. they are available in the public domain

2. Java (their implementation language) is of particular interest as a means of producing applications that can be executed on a variety of platforms.

3. Java can provide access to system resources such as databases and other processes on a host machine using methods such as Java Beans, RMI and JDBC. This can also provide legacy access to non-Java systems.

Other Java-based agent systems are available, but the authors are not aware of any others that are other than as research projects, and one of the aims of this document is to provide information on technologies that are, or are expected to become, commercially available.

Although the main focus of interest is, as stated above, on these technologies in Java, questions of interoperability are do figure, and will be raised briefly in the examination.

1. Some agent resources

The Agent Society - a "international industry and professional organization established to assist in the widespread development and emergence of intelligent agent technologies and markets" - is an excellent source of information, on agents of all kinds, including intelligent and information agents, as well as those of the autonomous, mobile kind addressed by this document.

- Agent Society homepage - http://www.agent.org/
- Agent Standards Activities - http://www.agent.org/pub/standards.html
- Agent Events - http://www.agent.org/pub/events.html
- Agent Discussion Groups - http://www.agent.org/pub/discussions.html

The WWW Consortium also has some useful resources on agents and mobile code -

- WWW Consortium homepage - http://www.w3.org/
- W3C mobile code review -http://www.w3.org/pub/WWW/MobileCode/

1. OMG MAF

The OMG (Object Management Group) has a Common Facility Task Force (CFTF) which has published a RFP3 for Mobile Agent Facility (http://www.omg.org/docs/1995/95-11-03.txt), which is largely concerned with providing a framework for mobile agents which implements a very open platform- and language-independent standard for agent implementations. RFP3 is in fact concerned with two areas - Data Interchange Facility and Mobile Agent Facility. This document is concerned with only the latter of these two, and submissions to the RFP3 have been accepted addressing only one of the two areas. The Data Interchange Facility and Mobile Agent Facility RFP (CF RFP3) homepage is available at - http://www.omg.org/library/schedule/CF_RFP3.htm.

There have been three responses (from IBM, General Magic and Crystaliz) to the MAF RFP, following which, the three originators of these responses have agreed to put together a joint submission. This is expected to be evaluated by the CFTF, which has already evaluated the earlier submission from IBM.

The OMG MAF page (username and password required) can be found at - http://tick.genmagic.com/~cynthia/mafeval/. It contains information about the joint submission and links to related resources.

It should be noted that neither *Aglets*, *Kafka* nor *Voyager* come close to fulfilling the requirements specified by the RFP3. Specifically, the RFP states that agents should be able to be written in different languages, and

should provide basic Common Facilities in the context of OMG's Object Management Architecture. Furthermore, the RFP states -

"…Common Facilities will need to provide CORBA-compliant interfaces and be consistent with the OMG Object Model…"

None of the three technologies currently comes close to providing this sort of support. This is not necessarily a condemnation of the development teams, however, as they have all made the decision to design a system which will allow the user community access to agent technology, and there are clearly market advantages to being early to release. The best systems will be those which are best suited in the long run to take on board the issues raised by MAF and similar standards bodies.

However, certain aspects of the MAF evaluation process can be applied to each, and although it does not represent a suitable framework for fully evaluating either technology, useful information can be gained by awareness of its aims and some of the questions raised.

The authors have chosen this RFP as the basis for their comparison of the technologies as it conforms quite closely to many of the issues raised in the FollowMe project. This does not mean that they share an exact match and, in fact, one issue that is absent from the MAF RFP which is important to FollowMe is that of security.

It is to be expected that the *Aglets, KAFKA* and *Odyssey* teams will work towards a state in which they could fulfil whatever requirements are laid down by the final accepted response to RFP3. There is no information available from ObjectSpace as to their intentions in this area. In each case, the development teams have made the decision to proceed in the absence of a definitive framework.

The position held by Mitsuru Oshima (of the *Aglets* development team) is of interest. In an email exchange the authors asked -

> "what plans are there to bring aglets into line with whatever MAF specification is accepted (the joint one currently being prepared, for instance)? "

The response (9th April, 1997) was -

> "The focus of MAF submission is on the interoperability, that is, the specification is independent of agent architecture. Although OMG/MAF is not on the top of the agenda, there is no technical barrier to bring aglets into the line. I cannot promise you that we finally join the submission, but it looks promising.
>
> "Once we join the submission and that submission is accepted, we're under obligation to develop a generally available offering that implements OMF/MAF specification."

On the Odyssey WWW pages, General Magic stress their involvement in MAF, and although they make no commitment to offering a MAF-compliant system, they state in their *Odyssey* FAQ -

> "General Magic, in conjunction with a number of other companies, is writing the specification for the Object Management Group's Mobile Agent Facility. The Mobile Agent Facility will enable agents to travel between agent systems. Odyssey is one example of an agent system. IBM Aglets is another."

Although not an official statement from Fujitsu, Takashi Nishigaya, one of the Kafka development team gave his personal opinion to the authors, writing that -

> "The objective of MAF is a language independent mobile agent, however, currently KAFKA API depends on Java API and Kafka API's are under construction. He is investigating how he can satisfy the MAF's requirements. In future versions of Kafka, the agent architecture might be separated from communication platform and language. If the architecture satisfies the MAF requirements, he may submit to MAF. Right now he has no strong motivation to support CORBA. ORBs (RMI, HORB) is regarded as implementation platforms of Kafka agents."

ObjectSpace state in their *1.0 Beta User Guide* -

> "… market feedback will drive the creation of seamless bridges from Voyager to existing technologies such as CORBA, DCOM and RMI."

Although this does not address the issue of OMG/MAF compliance, there is clearly an intention to leverage other technologies. A recent reply from David Glass (Senior Architect, distributed computing group, ObjectSpace, inc.) to a question about IIOP and Voyager on the Voyager mailing list (objectlist@objectspace.com) read -

> "our goal is to allow voyager to communicate with objects that understand different protocols as easily as possible. this includes DCOM and IIOP. we are therefore looking at voyager performing automatic protocol conversions so that it can talk to CORBA objects using IIOP, COM objects using DCOM, and voyager objects using smart messengers.
>
> "so rather than running voyager "over IIOP", the idea is that voyager should be smart enough to use the right on-the-wire protocol depending on what kind of object it wishes to talk to."

Interworking with legacy systems and agents is likely to become a more important issue in the future, and the authors believe that the importance of ORB support will show in this area as well as in OMG MAF compliance.

1. Other agent systems

There are a number of agent systems currently available, many as research projects, but some in more stable forms - a useful resource for general agent-related links on the WWW can be found at http://www.cl.cam.ac.uk/users/rwab1/ag-pages.html and the Agent Society's Agent Product and Research Activities page is also of interest - http://www.agent.org/pub/activity.html

APM has done previous work in the area of mobile code, and the document entitled *Status of Industry Work on Signed Mobile Code* (APM document 1879) and the presentations *Signed Mobile Code* (APM document 1779) and *Scripts and Mobile Agents* (APM document 1593) have direct relevance in this area. Work in the E2S project has also looked at issues of mobile code, but not in the context of autonomous agents, the accent being establishing secure links, and mobile code as a possible means to establishing this end.

Of specific interest is the mobile agent system *Telescript* from General Magic (http://www.genmagic.com/Telescript/index.html), a proprietary commercial system not based in Java, which has provided at the very least a yardstick against which to measure mobile agent systems, and possibly a model for certain types of agent interaction. The *Odyssey* system is gives a subset of *Telescript* functionality, with additional functionality possibly to appear in future version. Take-up of *Telescript* was not very great, partly, it seems, due to the commercial nature of the offering and its relatively high cost, and it has now been discontinued.

A. Agents in Java

1. Terms

As the area of agent technology is still a relatively new one, the terminology has not yet settled down. As a consequence, there is some real divergence in terminology between different systems - particularly noticeable in the areas of the three technologies under evaluation. For this reason, it is useful to set some terms of reference for the purposes of this document.

Host machine: a machine on which an *agent* resides, and on which the *host* and *environment* run

Host: the software controlling the *environment*, running on the *host machine*. The *host* has access to the O/S and other resources on the *host machine* which may not be available to *agents* residing in the *environment*.

Agent: an instance of some code that represents a user (may or may not be mobile)

Environment: the execution environment for an *agent*, it provides the software link between the *host* and the *agent*

1. FollowMe requirements

The FollowMe project is an ANSA project to allow mobile users access to information, and requires mobile software agents for its implementation. A number of issues are important to the project, and the decision as to which agent technology to use for initial implementation. Some of the most important of these are:

- **security**: the ability of agents to enjoy different levels of security based on their status, and for hosts to differentiate between different agents and to grant them different levels of access to resources. The ability to be able to implement security strategies is therefore important to the project.

Example - an agent from an untrusted domain might be refused access by a host being run by a bank if, for instance, the host contained a service providing financial data. In the same case, although some agents (those belonging to bank employees) might be allowed full access to all the data, other agents (agents belonging to customers) would only be allowed partial access.

- **Java support**: the language for implementation should be Java. Also important is that the execution base should be available for configuration for the needs of the project, to be able to enforce security strategies, for instance. Although this does not preclude the use of other languages, if the entire system were in Java, this type of configuration would clearly be easier.

Rationale - Java was chosen as the base language for the FollowMe project for three main reasons -

1. it provides a sandbox, allowing resources and access to them to be controlled within that context

2. it provides a great platform independence - a pure Java application should be 'compile once, run anywhere'.

3. there are a growing number of APIs providing access to legacy systems and information, allowing businesses to leverage their current systems.

- **movement of data**: agents should be able to move data with them, and not solely to move themselves

Example - If an agent is charged with moving between different mortgage companies for information about their products, that agent will need to 'report back' to with the information that it has gleaned.

- **control of agents**: agents should be able to move themselves (or cause themselves to be moved), and hosts should be able to move agents (or cause them to be moved). Agents must be aware of their environment, as their behaviour may well be dependent on their location.

Example - If an agent is finding information about new books from a publisher and visits a host maintained by the publisher in the U.K., the host might move it to a related site (if, for example, the newest information were kept at the publisher's U.S. headquarters). In this case, the agent would need to realise that it had moved and to change the currency in which it was working accordingly.

- **directory services**: there should be ways of naming, identifying and tracking agents so that agents can both find new resources and keep track of those which already exist.

- **communication**: agents should be able to communicate with each other and with their host(s).

Example- an agent which has gone to a host to arrange a meeting for its owner would need to talk to other people's agents to arrange the meeting. It might also need to talk to the host to determine the local time - otherwise a meeting might have been arranged at the wrong time!

- **public availability**: the technology chosen should be publicly available, allowing users to create their own agents. For this reason, choice of a short-term or unsupported research project would be inappropriate.

Rationale - the FollowMe project is an ESPRIT project to help European businesses to make better use of existing technology. Choosing a technology which would be unavailable to European businesses and the wider business community would not help to achieve this aim.

Other issues that may be of importance in the longer term include:

- **ORB support**: it is expected (in particular with reference to the MAF RFP3) that support for new ORBs will be important for agent systems in the long run, and it is not the intention of the FollowMe to exclude improvements and innovations in the field. For this reason, the extent to which the technologies are capable of supporting new ORBs will be important.

1. Aglets background

Aglets homepage - http://www.trl.ibm.co.jp/aglets/

Aglets are a technology made available by IBM Tokyo in alpha release in late 1996/early 1997. At the time of writing, it is in alpha 5b release, and the AWB (Aglets WorkBench) available on an platform supporting Java 1.0.2 with RMI, or Java 1.1+. Java 1.0.2 will not support any more.

There is a lively mailing list (aglets@cosimo.com) and WWW site (http://www.javalounge.com/), and a fair amount of documentation, including a book which is currently being written and is being considered for publication by Addison-Wesley.

The AWB is written entirely in Java, and is fairly simple to use. Tahiti, a GUI-based environment, acts both as environment and host to aglets - which are agents - although aglets can be serialised by hand.

1. KAFKA background

Kafka homepage - http://www.fujitsu.co.jp/hypertext/free/kafka/index.html

Kafka is a technology made available by Fujitsu in a beta release in August 1997. At the time of writing, it is in beta 1.4 release, and available on an platform supporting Java 1.1+. There is currently very little documentation in either English or Japanese. There is a mailing list (kafka-users@flab.fujitsu.co.jp).

Kafka is written entirely in Java, and the model for Agent control, instantiation and lifetime is very different to that for Aglets. Each KAFKA.Agent lives in an instance of the JVM (Java Virtual Machine) on the host machine, and although kafka.Agents can not migrate to other machines, the usual way to move code is by KAFKA.Actions, which can be moved between KAFKA.Agents using reflective techniques, and executed at each.

For these reasons, one reasonable model for implementing a system would be for KAFKA.Agents to act as hosts, providing environments for KAFKA.Actions, which are executed as separate, mobile code. This is the model adopted for the purposes of comparison with the Aglets system, unless specifically noted.

1. Voyager background

Voyager homepage - http://www.objectspace.com/voyager/

Voyager is an agent technology currently available in product release from ObjectSpace, and running on an platform under Java 1.1+. It is clearly aimed at commercial implementation, and the level of documentation is generally excellent, and there is a mailing list (objectlist@objectspace.com) on which the technical team seem to be active.

Voyager is currently available for free commercial use, and, according to ObjectSpace “the voyager core will remain 100% free for commercial deployment in "regular" computer systems like pcs, minis, etc. no loopholes, no caveats, 100% free.” Use for embedded applications is subject to licensing.

Voyager is written in completely in Java, and is fairly simple to use. There are two main types of object in Voyager - applications (which act as hosts) and agents (which can move between them), exchanging messages, etc.. Both

applications and agents can instantiate objects or call methods on remote hosts.

1. Odyssey background

Odyssey homepage - http://www.genmagic.com/agents.

Odyssey is an agent technology currently available in beta release from General Magic, running on any platform under Java 1.1+. As Odyssey is the result of a research effort, and not a commercial product, it may not be deployed in a commercial product at this time, though General Magic notes that they are willing to discuss commercial licensing. The level of documentation is generally poor, and there is some confusion as to which documentation applies to (the now defunct) *Telescript* system, and which to Odyssey.

At the beginning of May 1997, Danny B. Lange, inventor of Aglets, left IBM to join the Odyssey team.

Odyssey is written completely in Java, and there are two types of classes - agents and places, which acts as hosts. The source code is available.

I. AGENT FEATURES

A. MAF evaluation

In the MAF evaluation of the IBM RFP3 response by Jin. W. Chang (jin.w.chang@ac.com), several useful points are made. As they do not relate specifically to the technologies under review here, they are lifted from context and paraphrased. Those interested in the full evaluation are encouraged to reference the full evaluation, can be found on the OMG WWW site at - http://www.omg.org/archives/mafeval/0001.html. (MAF in this context is any Mobile Agent Facility).

1. MAF needs to support agent identification and agent location. It is not certain whether URL can be accepted as the standard for all different MAFs.

2. MAF needs to support basic agent transportation i.e., can agents be sent and received?

3. Some MAFs may utilise proprietary internal encoding schemes for the transport of an agent, others may not. As a result, how do we support various transport systems between different MAFs?

4. Since agents are moving from one node [*host* in the terminology of this document], MAF needs to support the disconnected situation.

5. MAF needs to send agents to different nodes. What is the appropriate abstraction of these nodes? URL may be too biased to an implementation - other MAF may choose to use the notion of place (which is simply represented as a string) and resolve its physical address using a Trader service.

6. As an agent moves, it may need to interact with other external applications. If an external application is a proactive entity which initiates the execution with an agent, it has to be able to monitor the arrival / departure / disposal / etc. of an agent. This may be accomplished using an Event service.

7. If an agent moves, MAF needs to track its location.

8. MAF needs to support run-time discovery and monitoring of agent execution engines and agents by name, or their capabilities.

9. MAF needs to support the cloning of an agent and the merging of the cloned agents.

10. MAF needs to delete all cloned agents.

The current RFP does not actually address issues 3, 4, 5, 6, 9, or 10. The interesting final paragraph of the evaluation concludes -

Regarding RFP; Basically, there are number of important issues which have not been addressed in the initial RFP. For example, there are no explicit requirements for interoperability between different MAFs. If MAFs can not interoperate with each other, what is the point of RFP? Furthermore, the initial RFP does not discuss about any requirements for supporting the interaction between agents and other external applications. In practice, mobile agents rarely work as a standalone program. They tends to collaborate with other applications.

A. Aglet features

1. Environment

Aglets are most commonly based in the Tahiti environment, provided with the AWB (Aglets WorkBench) as an execution environment. This provides security features, messaging and migration. Tahiti is 100% Java, but the source is not currently released by IBM (there seem to be no plans to make it available) - and this causes a number of problems.

1. Security

The first of these is security: Tahiti applies security policies, which are partly configurable by the user. However, they do not deal with issues of exactly which agents can talk to each other, whether agents can be blocked from entering the host, or whether aglets can be restricted from moving from a host once they have entered it. All these issues are likely to be important for systems with any level of security greater than a very basic one. Some of these security issues are addressed by the use of Proxies, but control of all method invocation is through messaging, for which no standards currently exist.

HTTP tunnelling via a proxy server has been implemented for Aglets, to allow access to information behind a firewall. Although the ability to access data sources in such a situation is important, this solution might be seen as rather simplistic, and to create dangerous opportunities for a technology whose security features are not very well-defined in the first place.

Another issue is the lack of a directory service. Although Aglets are provided with unique global identifiers, there is no easy method to track the their location. Suggestions for forwarding with the use of Proxies would not seem to scale, and although there is some work within the user community in this

area, there is no official line from IBM, and there does not seem to be a much likelihood of a directory service being provided in a release in the near future.

1. ORBs and messaging

There is currently no support for any ORBs in the AWB.

Some discussion on issues such as trading services, directory services and message-passing formats, KQML and KIF use, etc., has taken place within the user community, but lack of interest from IBM and the number of teething problems associated with the alpha software seem to have distracted discussion away from these topics. Any work being done seems to be on an individual basis by users unconnected with IBM.

1. General

The great plus point for Aglets is their ease of use. Tahiti provides a very simple environment in which it is easy to place Aglets, and although the API is not immediately accessible, the documentation currently being produced is simplifying the process of Aglet programming. The user community around Aglets is enthusiastic and growing, and this is largely a reflection of the ease of use of the technology and the fact that it arrived on the scene relatively early.

IBM have been unwilling to give any information about their plans for a commercial release of Aglets, and are insisting that the current alpha license - which is for evaluation purposes only, and allows no commercial use of the technology - be followed. However, the amount of effort currently being spent on research and the visibility of Aglets, coupled with IBM's public stance on the importance of Java as a technology, point firmly towards commercial availability in the future.

In conclusion, Aglets provide a relatively simple way to implement agents in Java, and have the advantage of a wide and growing user base. The disadvantages revolve round the fact that some important issues have not been dealt with in as much detail as might be appropriate - in particular security. IBM has clearly made the decision to produce an easy-to-use system quickly, and seems to have succeeded. This does mean, however, that some fundamental issues have not been fully resolved, whilst the ability to make changes to affected parts of the system has been put outside easy control of developers.

It is the opinion of the authors that Aglets may well gain a good deal of user acceptance and therefore market share, although there has been some rumbling within the user community that changes to important systems take a long time to be resolved by the development team. It may be that IBM made a bad decision to go live with an early alpha system, expecting a small band of testers to give them feedback which they could use, when, in fact, a large number of users have jumped at this technology, overwhelming the development team. It will be interesting to see whether the book currently

being written is published, and if so, what impact it makes on the project and its acceptance.

1. Methods

Aglets are provided with an API which is already very rich. Some of the methods available to programmers include (see Aglet documentation for details and other methods) -

clone(), deactivate(long), dispatch(URL), dispose(), getAgletContext(), (URL), getCodeBase(), getIdentifier(), getMessageManager(), getProperty(String), getPropertyKeys(), handleMessage(Message), onActivation(), onArrival(), onClone(), onCloning(), onCreation(Object), onDeactivating(long), onDispatching(URL), onDisposing(), onReverting(URL), run() , setItinerary(Itinerary), setProperty(String, String), setProxy(AgletProxy), subscribeMessage(String)

A. KAFKA features

As explained above, in *section 1, KAFKA background*, the model for agents in KAFKA is very different to that in Aglets. This examination of the features of KAFKA assumes an implementation such as that outlined above, with KAFKA.Agents acting as hosts for KAFKA.Actions, thus allowing a better comparison with Aglets. Indeed, the methods provided for KAFKA.Actions have very similar functions to those for Aglets. However, some other features which do not rest in this implementation are picked out specifically.

It should again be noted that statements and conclusions reached here should be regarded as provisional until a full software release is available.

1. Environment and security

Kafka.Agents do not require an execution environment other than an RMI registry (the one provided with the release is an adapted version which implements a directory service), and as the implementation being proposed would involve using the kafka.Agent as a host, the developer clearly has almost complete control over policies with regard to agents (here kafka.Actions). In fact, although in the (sparse) documentation currently available there is discussion of security implementations, not all the three levels discussed have been implemented to the authors' knowledge. APM has done some related work in implementing a security model using RMI which should be closely relevant to the kafka system.

Much of the security implementation, and, indeed the power of the KAFKA system, comes from the use made of reflection. Classes and, specifically, kafka.Actions, can be added to a kafka.Agent, thus extending the capabilities of a kafka.Agent, and allowing behaviour to be adaptive, depending on context. Each kafka.Action added to a kafka.Agent has its method access

mediated the kafka.Agent, according to a value associated with the agent, which could be applied as method-by-method policy.

Kafka.Agents are enable to register with a Directory service, but this is not the case for Kafka.Actions - the agents in this study. The development team at Fujitsu have confirmed that there is no current method to track Kafka.Actions, but the fact that the lower level classes are very open to development means that building a good distributed system incorporating a directory service enabling tracking of agents.

The design for Kafka is strong, and, as mentioned several times above, allows much scope to the developer, whilst providing the framework to implement a basic system without much difficulty.

1. ORBs and messaging

There is currently support for HORB, and it is expected that other ORBs will be added - it is certainly possible for developers to produce their own ORBs, and the HORB classes would serve as a model.

1. General

The state of play regarding licensing for Kafka is currently unclear. Fujitsu should be approached for more information. It is hoped that commercial licensing may be a possibility.

The major drawback to the Kafka system is also one of its strongest features - much of the responsibility for the system lies with the developer. Although this can obviously means that the Kafka system is not very accessible, this is likely to change as more documentation becomes available. On the other hand, this does give the developer the chance to implement a system 'from the bottom up', implementing security policies, etc. as needed.

From the viewpoint of extensibility, the great plus point of the Kafka system is that there is great scope to tailor the technology to the specific needs, as there is access to all the constituent classes. The system seems well designed to take this sort of development, and the fact that it is based firmly in Java's RMI registry, without precluding developer adaptation is a strong point in its favour. However, before serious development can go on in Kafka, the lack of documentation must be addressed.

1. Methods

The number of methods available to programmers in Kafka API is currently quite small, and those that are of interest are split across the Kafka.Agent and the Kafka.Action classes. Some of the methods available to programmers include -

Kafka.Agent - add(String, Object), add(String, Object, int), call(String, Object[]), callAsync(String, Object[]), eval(byte[], String, Object[]), evalAsync(byte[]), get(String), getResult(int), init(), listAll(), moveTo(RemoteAgent), remove(String), run(), set(String, Object), start(), waitForKilled()

Kafka.Action - destroy(), getBytes(String), getClassData(), getInputStream(String), ignoreResult(), init(), moveTo(RemoteAgent), printInfo(), self(), start(Object[]), startThread(), stop()

A. Voyager features

1. Environment

There are two types of Voyager object - *Voyager.agent*s and Voyager applications, both of which are based in a Voyager host object, the services for which are started with the creation of a Voyager applications, and both of which are written entirely in Java. However, the source is not available for the Voyager object, and new instances may not be constructed, leaving the host out of the control of the developer, which causes problems.

1. Security

The Voyager security comes complete with an optional Voyager Security Manager. The Voyager Security Manager is a pluggable security manager that restricts the operations of foreign objects. A foreign object is an object whose class was loaded across the network from another program.

The Voyager Federated Directory Service allows programmers to build and link together hierarchies for the management of objects in a distributed system. Voyager agents do, however, have an automatically assigned Globally Unique Identifier (GUID).

1. ORBs and messaging

Although the Voyager system is referred to as an ORB throughout the documentation, there is no support for any other ORBs currently supported - though note ObjectSpace's statements in *Section 1, OMG MAF*.

The messaging system provided with Voyager is quite sophisticated, including 'Smart Messengers', which can track agents around the network, and messaging forwarding (via the secretary objects). There is also support for the remote invocation of remote agents and applications, including the ability to instantiate new objects remotely. Any Java class can be made a 'virtual class', allowing it to be instantiated on remote system.

Lifespans can be controlled, allowing objects to self-destruct after a certain elapsed time.

1. General

Though not supplied with any GUI, Voyager is quite a simple system, and is extremely well documented. Also in its favour is the fact that it is in a relatively stable release, and may be used to build commercial applications. ObjectSpace has issued a statement to the mailing list (objectlist.@object.com) which is interesting, and may be important in certain situations -

> "if you wish to embed voyager in a consumer device, network card, etc. etc. or wish to obtain a source license to do your own customization, then you will have to be granted a license from objectspace. the terms for this license would vary based on the situation"

It is this ease of use and its commercial opportunities which might help Voyager to acceptance in the user community.

1. Methods

The Voyager API includes quite a large number or classes and methods available to the programmer. Many of the functions of Voyager are spread across different classes; some of the methods available to programmers in the Agent class include -

die(), dieAfter(int), dieAt(Date), dieWhenNoReferences(), getAddress(), getAlias(), getApplicationAddress(), getForwarding(), getId(), liveForever(), move(VObject, String), release(), setForwarding(boolean), toString()

A. Odyssey features

The model for Odyssey is part-way between that for Aglets and that for KAFKA, and much of the design is clearly modelled on that of the *Telescript* system - also from General Magic. *Place* objects act as hosts for *Agent* object, and can track entry and exit of different agents. They also deal with all access to resources such as databases on the host machine.

1. Environment and Security

Place objects do not require an execution environment other than an RMI registry, and the security model is based on the Java security model and security manager. While there are no explicit restrictions on a method calls

basis, the fact that the source is available to the developer allows much control over security policies to be implemented.

There is no directory service either for Places or Agents, but Places possess a Location object which is not URL-based, and can be used for identification.

The design for Odyssey seems strong, based as it is on the *Telescript* model. Jim White (of the Odyssey development team) notes in the Odyssey FAQ -

> "Odyssey 1.0 provides only a subset of Telescript's agent-and-place functionality. Additional functionality may appear in future versions. Some Telescript functionality cannot be provided in 100% pure Java using version 1.1 of the Java virtual machine. Also, Telescript was a complete programming language and included much functionality that was not inherently related to mobile agents (e.g., string operations and dictionaries). Odyssey does not need to provide these classes as they are already provided within the Java Development Kit."

There seems to be no specific support for messaging, which means that Odyssey must use those capabilities supported by RMI. This means that Odyssey supports synchronous, but not asynchronous messaging. Although not an insuperable obstacle, and a feature which could be provided by the developer, this is a drawback to an otherwise excellent system.

1. ORBs and messaging

There is currently no support for any ORBs, though there are implements of Odyssey transfer by IIOP and DCOM. As noted above in *section 1, OMG MAF*, General Magic are very involved in the work going on in the OMG MAF, and it seems likely that support for other ORBs will be built in if General Magic decide make Odyssey their offering for a MAF-compliant system.

1. General

The source for Odyssey is available in beta form, and though not supported directly, General Magic welcomes feedback. Odyssey is not yet available for commercial products, and no decision has yet been made as to whether to make it available on a commercial basis, but General Magic is asking those interested in commercial licensing to contact them.

Odyssey sits at an important point in its life-cycle. Although the system seems quite mature (probably due to its deriving much of its structure from *Telescript*), the lack of documentation is clearly a problem. On the other hand, the availability of the source, the fact that the system is based firmly in Java's RMI registry and that there is low level access means that it is highly extensible.

1. Methods

The number of methods provided in the Odyssey API is small, and they are spread over a number of classes. The methods available to programmers in the genmagic.odyssey.Place and genmagic.odyssey.Agent classes (intended only to provide a kernel of functionality) include -

genmagic.odyssey.Place - entering(Class, ProcessName, String), exiting(Class, ProcessName, String), location(), returnTicket(), run(), waitForever()

genmagic.odyssey.Agent - go(Ticket), here(), nextAgentPlace(), run(), setNextAgentPlace(Place)

I. COMPARISON

A. Feature comparison

	Aglets	*Kafka*	*Voyager*	*Odyssey*
Can an agent migrate?	Yes	Yes	Yes	Yes
Can a host force an agent to migrate?	Yes	Yes	Yes	Yes
100% Java?	Yes	Yes	Yes	Yes
Security policies?	Yes	Yes	Yes	Yes
Security policies configurable?	Partly, through Tahiti	Yes	Yes	Yes
Security policies modifiable?	No	Yes	No	Yes
Status?	Alpha 5b	Beta 1.4	1.0.1	Beta 2.0
Documentation?	Good - growing	Poor	Very Good	Poor
Ease of use?	Very accessible	Low level programming	Very accessible	Quite accessible
Extensibility?	Medium	High	Medium	High
Other ORB support?	None explicit	Yes (HORB)	None explicit	None explicit
Transport layer protocol	ATTP	RMI or HORB		RMI / IIOP / DCOM
Can agents communicate with host?	Yes	Yes	Yes	Yes
Can agents communicate with each other?	Yes	Only via host	Yes	Only via host
Synchronous and Asynchronous messages?	Yes	Yes	Yes	Only synchronous
Can agents transport data?	Yes	Yes	Yes	Yes
Directory service?	No	Yes	Yes	No
Persistent Agent	No	No	Yes	No
Event service?, Event Delegation?	Yes	No	Yes	Yes
User acceptance?	High	not applicable	?	?
Source available?	Partially	Yes	No	Partially
Commercial license available?	No (no plans)	Not yet determined	Yes	Not yet determined
Lifespan tracking?	No	No	Yes	No
UID (Unique Identifier)	Yes	No	Yes	No
GUI provided?	Yes	Simple tool	No	No

A. Other issues

The early release of Aglets into the community stands it in both good and bad stead against the opposition. Whilst there is a wide user community, the

amount of work that the community generates for the development team does seem to be hindering development of the technology. There is also some unrest being felt over the time to a commercial product - users have got so used to using Aglets that they want to use it for real projects and products. The ease of use of Aglets counts in their favour, but the difficulty associated in tailoring the system for specific projects is likely to stand against it. All these issues could bode well or badly for Aglets, and only time will tell how well they are accepted.

It has in its favour that it is maybe a better designed system, and is certainly more easily extensible than Aglets. Whether it is robust enough to stand up to the opposition, and whether the documentation will provide enough information to allow much serious development in the short term are questions that we cannot yet answer. Another point in its favour is the fact that commercial licenses may well be available for development. This, in conjunction with the availability of the source code, places Kafka in a very strong position.

The picture painted by ObjectSpace in their marketing literature puts Voyager at the intersection of Java, Agent Technology and ORBs, and mentions ORBIX, TELESCRIPT and JGL specifically. However, this picture does not seem fair, as the technology does not really implement the best of any of the other technologies it lists, and has no obvious connection with any of the specific products. The current lack of security and lack of thought to matters of scalability are two very important drawbacks, and less though seems to have gone into the design of the system than, for instance, Kafka. The main points in its favour are its excellent documentation and the availability of commercial licenses.

Odyssey is quite a mature technology, thanks to General Magic's experience with *Telescript*, and is let down only by its lack of documentation and commercial licensing. Also strongly in its favour is General Magic's involvement in the MAF submission, where the Odyssey team seem more involved that do IBM's Aglet team. It is as yet too early to see what the user community's take-up of Odyssey will be, but its extensibility should play in its favour, particularly if the lack of documentation can be remedied. General Magic also seem very aware of the importance of other ORBs, and continued development in this direction should be expected, and would clearly count in the technology's favour. The lack of explicit messaging capabilities is also a clear drawback.

I. CONCLUSION

A. Aglets, KAFKA, Voyager or Odyssey for FollowMe?

1. Security

The FollowMe project has some important requirements, and most of these are met by both systems. However, the issue where there is divergence in the suitability for the project lies in extensibility. None of the technologies yet provides security to the level required for the FollowMe project, and where the Kafka and Odyssey systems will allow its implementation, it seems that the Aglets system would make this a great deal more difficult, particularly given the lack of access to source code. The same goes for the Voyager system - possibly even to a greater degree.

1. ORB support

Another important issue is that there is some ORB support for KAFKA, where there is as yet none for Aglets, Voyager or Odyssey, though some support seems likely in the last. It is our opinion that any serious agent system will need to support ORBs in the longer term, and it seems likely that this will be more easily implemented in Kafka or Odyssey than in Aglets or Voyager.

1. Availability, users and licensing

KAFKA suffers from its lack of exposure to the user community, and questions remain as to its robustness, and therefore its likely adoption, and if it were not for these issues, it would probably be more easily recommended over Odyssey if it were not for the latter's maturity, though it falls down in the area of messaging. One point which is likely to become important, and where KAFKA, Voyager and Odyssey stand in better stead than Aglets is their stance on licensing, as noted above in *section A, Other issues*, whereby they are all open (or are expected to be open, in the case of KAFKA) to commercial licensing possibilities. In this area, it may well be that Voyager's earlier release may well stand it in good stead.

1. Final choice

The final decision for the FollowMe project must, in the authors' opinion, be between Kafka and Odyssey. Both are very extensible and should have their source fully available to the developer, and neither precludes implementation of the sort of security required for the project. Both teams seem very aware of the necessity of ORB support, are keeping abreast of developments in OMG MAF which, the authors believe, is likely to have a defining influence on autonomous agent technologies in the near future. General Magic are willing to discuss commercial licensing of their respective technologies, and it is hoped that the same will be true for Fujitsu.

Counting against each is the lack of documentation, and need for low level development, which may lead to less user acceptance. It is the author's recommendation that the project team monitor user acceptance of each technology, and see how each development team addresses the issue of documentation before making a concrete choice. The authors also believe that the two systems may well become interoperable in the longer term as each embraces future MAF specifications.

I. REFERENCES

A. APM documents

- *Status of Industry Work on Signed Mobile Code* (APM document 1879)
- *Signed Mobile Code* (presentation, APM document 1779)
- *Scripts and Mobile Agents* (presentation, APM document 1593)

B. Worldwide Web pages

- Agent Discussion Groups - http://www.agent.org/pub/discussions.html
- Agent Events - http://www.agent.org/pub/events.html
- Agent resources page - http://www.cl.cam.ac.uk/users/rwab1/ag-pages.html
- Agent Society Agent Product and Research Activities page - http://www.agent.org/pub/activity.html
- Agent Society homepage - http://www.agent.org/
- Agent Standards Activities - http://www.agent.org/pub/standards.html
- Aglets homepage - http://www.trl.ibm.co.jp/aglets/
- Aglets user community WWW site - http://www.javalounge.com/
- Data Interchange Facility and Mobile Agent Facility RFP (CF RFP3) homepage - http://www.omg.org/library/schedule/CF_RFP3.htm
- Odyssey homepage - http://www.genmagic.com/agents
- OMG MAF evaluation of IBM response to RFP3 http://www.omg.org/archives/mafeval/0001.html
- OMG MAF page (password required protected) - http://tick.genmagic.com/~cynthia/mafeval

- RFP3 for Mobile Agent Facility - http://www.omg.org/docs/1995/95-11-03.txt
- Telescript homepage - http://www.genmagic.com/Telescript/index.html
- Voyager homepage - http://www.objectspace.com/voyager/
- Kafka homepage - http://www.fujitsu.co.jp/hypertext/free/kafka/index.html
- W3C mobile code review -http://www.w3.org/pub/WWW/MobileCode/
- W3C homepage - http://www.w3.org/

C. Mailing lists

Voyager mailing lists -

- objectlist@objectspace.com
- voyager-interest@objectspace.com

Aglets mailing list - aglets@cosimo.com

Kafka mailing list - kafka-users@flab.fujitsu.co.jp