A Survey on Agent Systems Supporting Java

Oskari Koskimies

Department of Computer Science, University of Helsinki

P.O.Box 26 (Teollisuuskatu 23)
FIN-00014 UNIVERSITY OF HELSINKI, Finland

E-mail: Oskari.Koskimies@cs.helsinki.fi

Abstract

The aim of the MONADS project is to develop an architecture, based on mobile agents, which supports nomadic users. Intelligent agents can adapt to changes in terminal equipment and network connection, and collaborate with service agents to provide an optimal service in changing circumstances. The project will develop a prototype based on an existing agent system, which will be extended for mobile environments.

In order to choose the system to be used, we evaluated the leading Java-based agent platforms available today. We found that each system had its strengths, and thus no one system could be recommended as best for all kinds of use. However, Voyager is arguably the best overall system.

1 Introduction

Internet offers an ever–increasing amount of information services. People become increasingly dependent on having access to those services, and need them even when they do not have network access. By using wireless

The financial support of Nokia, Sonera and the Finnish Development Centre of Technology is gratefully acknowledged.

networks, people can utilize network information services even when they are on the move. However, current communication services do not work well in a mobile environment, because of the different characteristics of fixed and mobile networks. Applications that were designed for fixed, reliable networks often perform poorly in a mobile environment. The nomadic user is also limited by the capabilities of his terminal, which are often inferior to desktop machines. For example, a terminal may be unable to store or display large, high–quality video clips.

These problems can be solved by using light–weight, mobile–aware protocols and applications in the mobile environment. In addition, a high degree of adaptability is needed to cope with changing circumstances, for example when a user switches off his GSM connection and plugs his computer in the office Ethernet network.

Autonomous software agents have been seen as the next big step in distributed systems. A lot of research has already been done in this area. However, the use of agent technology to support nomadic users has not yet received much attention, although it is often mentioned as a potential application area.

The aim of the Monads[6] project is to develop an architecture, based on mobile agents, which supports nomadic users. The emphasis is on adaptation. Intelligent agents can adapt to changes in terminal equipment and network connection, and collaborate with service agents to provide an optimal service in changing circumstances.

The project will develop a prototype based on an existing agent system. The existing system will be extended for mobile environments, leveraging software and experience from the Mowgli[7] [1] project. In order to choose the system to be used, we had to evaluate currently available systems. In this article we present a summary of the results of that evaluation, Of each system, key features such as communication facilities, security and reliability are evaluated. However, one should bear the following things in mind when reading the evaluation:

1. The field is evolving quickly: New systems appear, and existing ones improve. The evaluation was done during Spring 1998.

2. JDK1.1 itself provides CORBA support [29], and in this regard all

[6]http://www.cs.Helsinki.FI/research/monads/
[7]http://www.cs.Helsinki.FI/research/mowgli/

systems except D'Agents (aka AgentTCL) [4, 11, 10, 12] support CORBA. However, Voyager additionally allows CORBA objects to be used in the same way as objects remote–enabled by Voyager.

3. JDK1.1 also provides RMI support [31], so all systems except D'Agents can use Java RMI for communication. However, RMI only provides for synchronous messaging, and is not very efficient. Most agent systems also provide other forms of communication.

4. The Java Security Manager architecture [5] can be used to implement security in all systems. Most systems have provided for security in this way. However, this will only provide for access control, not encryption and authentication, which some systems (like Concordia [20, 32, 21] and D'Agents) provide additional support for.

2 Initial selection criteria

We have limited ourselves to evaluating agent systems that could, at least in theory, be used in the Monads project. This leads to the following four constraints:

1. **Linux Platform.** The agent system must run on the Linux operating system. All pure Java systems are acceptable. This requirement stems from the fact that Linux is the development platform for the Mowgli Data Channel Service [1], which is necessary for this project.

2. **Java Language.** The agent system must be able to run agents written in the Java language. Due to its portability, widespread use and support, Java is the most likely contender for a general agent language.

3. **Evaluation version available.** An evaluation version of the system is crucial for a reliable estimate of the system's characteristics. In addition, if none is available, the software is in all probability too unstable to use.

4. **Generality.** The agent system must be suited to developing all kinds of mobile agents, not just, for example, Web searching agents.

Limitations on the kinds of agents it is possible to create using the agent system could prove fatal later in the project.

The first constraint is the most restraining. It effectively constrains the choice of commercial systems to those that are implemented in pure Java. Fortunately, this includes most important systems. Academic systems (most notably D'Agents) are more likely to support Linux even if they are not written in pure Java.

The performance of these systems has not been evaluated, due to time constraints. It should, however, be noted that the systems which utilize Java RMI for communication are bound by its perhaps less than optimal performance. Also, the Voyager [23, 24] system is undergoing performance optimization before the production version is released.

3 Candidate systems

3.1 Aglets

The Aglets Software Development Kit (ASDK) [15, 17] is developed by the IBM corporation. The ASDK 1.0 production version has just been released (April 24, 1998). It runs on all platforms which support JDK 1.1, and currently consists of about 190 core classes (ca. 700 kB).

The name is not just an idle pun on agents and applets; the aglets programming paradigm closely resembles that of applets, with significant events in an aglets life having callback methods (like `onCreation` and `onArrival`) in the aglet class.

Still following the applet model, aglets access their environment through an *AgletContext* object—similar to the AppletContext for applets—which corresponds to an execution context. For example, to create a new Aglet in a given context, `createAglet` is invoked on the context.

Movement

IBM is pushing its Agent Transfer Protocol (ATP) [16] as a standard for transferring agents between networked computers. ATP is very much like HTTP 1.0 [3], with the same advantages and disadvantages. The main difference is the methods it supports:

DISPATCH Transport an agent to a remote host.

RETRACT Retrieve an agent from a remote host.

FETCH Retrieve executable code required to execute an agent.

MESSAGE Forward a message to an agent.

ATP can easily be wrapped up in a HTTP request, which enables ATP to be used from within a firewall, using HTTP tunneling and HTTP proxy servers.

While class loading is basically on–demand, JAR [27] files (Java class archives) will be supported in the future. When aglet class files are in a JAR file , the whole JAR file will be transferred to the target system when the agent moves.

3.1.1 Communication

Agents do not refer to other agents directly. Instead, they use *AgletProxy* objects. The AgletProxy interface defines methods for accessing the aglet it represents, for example, sending it messages. The interface is common to all Aglets, so an AgletProxy is not the same thing as a CORBA or RMI reference. Its main role is to shield the aglet from unauthorized access. Currently AgletProxies do not keep track of roaming agents, so once an agent moves all referencing AgletProxy objects become invalid.

Communication between agents is not based on RMI, but on *messages* (however, messages are probably implemented using RMI). Messages can be of three types:

1. **Synchronous messages** cause the sender to block until the result has been returned.

2. **Asynchronous messages** return immediately with a placeholder object that can be periodically queried for the results. An agent can also wait for the reply to arrive, specifying a time limit for the wait.

3. **Oneway messages** are asynchronous messages that do not return anything.

A message is sent to an agent using one of the `sendMessage` methods of its AgletProxy object. Messages received by an agent are stored in a priority queue (messages can be given priorities), and the agent's `handleMessage` method is then called for each message in turn, giving the message object as a parameter for the method.

3.1.2 Documentation

ASDK documentation consists of the as yet incomplete Aglets specification, JavaDoc API documentation, ATP specification, and some tutorials. In addition, there is some introductory material created by third parties. While not extensive, the available documentation is sufficient for learning the basics of the system.

3.1.3 Security

Aglets may be *trusted* or *untrusted*. An agent that is launched locally and only uses local code is trusted, all other agents are untrusted. Read and write access to files, object instantiation and window access can be specified separately for trusted an untrusted agents.

The security measures are implemented using the standard Java Security Manager architecture. There is no encryption, or trust based on secure agent ID's.

3.1.4 Reliability

The only available reliability feature is the preliminary `snapshot` method in the aglet class, which saves a checkpoint of the aglet's state to secondary storage, and reactivates the aglet from this checkpoint if it is accidentally killed.

3.1.5 User Interface

One of the ASDK's strong points is it easy-to-use user interface. Setting up the system is easy, and all example aglets are controlled via GUI windows. The agent manager interface includes functions for creating, dispatching, retracting, killing and interrogating agents, as well as for setting security privileges.

3.1.6 Standard Support

The Aglets development team participates in MASIF [8]. Correspondingly, IIOP [25] may be supported in the future as a transport protocol. However, currently MASIF is not supported, and there is no explicit CORBA support.

3.1.7 Distinguishing features

There is some additional agent collaboration support in the form of a set of templates called Java–based Moderator Templates (JMT) that define the basic collaborative behaviors of mobile agents. JMT allows developers to build a complex plan by simply combining them so that multiple agents can work together toward a common goal. This is done using *moderator agents*, which manage and combine the work of other agents.

The JKQML [14] framework from IBM alphaWorks supports KQML communication between aglets.

3.1.8 Openness

Source code is provided for the some of the core classes, but not for ones containing important implementation details. It is probably only meant as additional documentation. The Aglets API does not provide hooks for extending ASDK functionality.

3.1.9 Evaluation

Aglets is fairly reliable medium–range agent system. While it does not provide all the advanced features of Voyager, or the security and reliability of Concordia, it is also a lighter system. Its main advantage is its simplicity and user–friendliness. On the other hand, there is no CORBA support, and providing the necessary extensions to the system can become hard when neither source code or extension hooks are available.

Advantages:

- Nice, easy-to-use graphical user interface
- MASIF involvement
- Good selection of message types

- Basic security support
- Firewall support via HTTP tunneling for ATP
- Familiar applet–like programming model
- Collaboration support
- Backing of a large corporation

Disadvantages:

- No CORBA support
- No hooks for extending system functionality
- Agent references become stale when an agent moves
- Reliability is only beginning to get attention

3.2 Concordia

Concordia [20, 32, 21] has been developed by Mitsubishi Electric Information Technology Center America. Version 1.0 has been available since January 19th, 1998. It requires JDK 1.1.3, and contains about 480 core classes (ca. 1.9 Mb), which makes it the largest system for the moment. Also, the above figure does not include the third party classes (about 600 classes, taking up ca. 2.4 Mb) that the Concordia package includes.

3.2.1 Movement

The Concordia agent programming model is based on itineraries. An itinerary contains a list of target systems, and for each target system there is method that will be invoked once that target system is reached. The itinerary of an agent may be modified by the agent itself, by the agent system, or by the user via a graphical user interface.

Class loading in Concordia is dynamic, but after a class has been loaded, it is packaged in a special data structure which travels with the agent. While this keeps things efficient, it does not help in the case where the agent does not need a class before long after it has been dispatched.

3.2.2 Communication

For agent-to-agent communication, Concordia provides distributed synchronous and asynchronous events. The events differ from Aglet messages in the sense that they cannot return a value; otherwise the two mechanisms have a lot in common. However, Concordia events can be multicast to a group of agents, and they allow a primitive form of location–transparent communication by using non–mobile event manager objects.

3.2.3 Documentation

Documentation consists of an architecture description and a separate document about security and reliability. The installation package also has a developers guide in HTML format, including instructions for getting started. The startup documentation is poor, though, containing errors and omissions.

Generally speaking, the documentation is somewhat below par, but does explain the basics of agent programming in the Concordia system.

3.2.4 Security

Security is where Concordia shines. Secure communications are implemented using SSL [6]. Agent data is encrypted during transfer and storage, and security permissions for an agent depend on the user who launched the agent, in contrast to, for example, Aglets where only locally created agents are trusted. However, a user is authenticated by a password that the agent carries, not by a certificate with a secure hash of agent code. This means that user identification does not guarantee that the agent contains the same code as when the user launched it.

3.2.5 Reliability

Reliability is another of Concordia's strengths. Agent movement is fully transactional, meaning that agent state is saved to secondary storage immediately upon arrival, and the agent transfer does not complete before the save is successful.

Agents are automatically checkpointed just before departure in the source system, and just after arrival in the destination system (as outlined above). If an agent server crashes, agent state can be restored from the most recent checkpoint. The checkpoint just before departure is to save the results of processing, in case the server crashes while attempting to transfer the agent.

The weakness of this automatic checkpointing is that it fails in the case where agent operations are not idempotent. This can partly be handled by explicitly requesting the agent to be checkpointed at key points in the processing.

3.2.6 User Interface

While the graphic user interface is probably the most advanced of the systems presented here, it is also the most unreliable. However, this may be because of JDK problems.

Problems aside, the GUI has perhaps the best general agent control method. The itinerary can be modified by the user without any programming knowledge. All that is required is that the agent programmer has defined a separate method for each type of host. E.g. there can be one method for querying data, and another for returning results.

The GUI also has a wealth of configuration options, allowing different services to be started or stopped, agents installed and launched, security options specified etc.

3.2.7 Standard Support

Concordia does not support any agent standards.

3.2.8 Distinguishing Features

Like the ASDK, Concordia provides a collaboration framework, although Concordia's framework is much simpler, at least conceptually. It supports the common case where a single master agent waits for multiple slaves to accomplish their tasks and then processes the results. The structure may extend to several levels: slaves can themselves be masters for other slaves.

3.2.9 Openness

Aside from examples, Concordia does not come with any source code. The *Service Bridge*, which is meant for providing an interface from Concordia agents to the services available at the various machines in the network, is the only extension API provided. It is only suitable for adding new services, not for modifying Concordia behaviour.

3.2.10 Evaluation

Concordia is a large system, with comparatively extensive reliability and security features. However, agent communication is limited, and needs RMI to be complete. Even so, there is no way to invoke a remote method asynchronously.

Concordia documentation is barely adequate, and not much thought has been put into the installation procedure: Concordia installation requires Korn shell to be installed on your system and was, with the possible exception of D'Agents which was not yet available, the most difficult system to install. The overall impression is that of an unfinished product.

Advantages:

- Good security and reliability features
- Nice user interface
- Collaboration support
- Distributed events
- Flexible generic agent control

Disadvantages:

- Unfinished system and documentation
- No messaging system (although events are provided). Instead, RMI has to be used, which means asynchronous (future) messages are not possible
- Messaging is not migration–transparent
- Large system size
- No extension APIs or source

3.3 D'Agents (AgentTCL)

D'Agents [4, 11, 10, 12] (formerly AgentTCL) is developed by Dartmouth College. It is in several ways unique with respect to the systems evaluated here. Firstly, it is an academic system, with full source available. Secondly, it implements agent movement in a way which allows execution state to be captured. And last (and least pleasant), there is no publicly available version which supports Java, although there are internal versions that do.

The last point should by rights have excluded the system from this evaluation, but a public version with Java is promised "Any Day Real Soon Now", and the source availability makes D'Agents too good a possibility to ignore.

3.3.1 Movement

By using a modified Java runtime, D'Agents is able to capture agent execution state. This makes it possible for D'Agents to continue execution immediately after the move command at the new node. While this programming model is not intrinsically more powerful than the one used by other systems, and could actually lead to code that is harder to understand, it does make agent programming more straightforward.

The major disadvantage of using a custom Java runtime is that the Java runtime version that D'Agents supports will unavoidably lag behind—currently it is in version 1.0.2, whereas the most recent version is 1.1.6. One is also required to license the JDK if one wants to get the sources for the modified runtime.

Once an agent has been transferred, it cannot load any classes from its origin host. Also, since only Java 1.0.2 is supported, RMI is not available. This means that *all* non–system classes that the agent might need must be transferred with the agent. Serialization does not capture classes which have no instances, so it is necessary for the agent to provide a list of classes it needs.

It should perhaps be noted that while all other agent system use Java Object Serialization to move agents, D'Agents uses its own state capture code, partially borrowed from the Sumatra [2] system. This becomes an issue if agent movement is standardized, because the bytestream created by Java Object Serialization is in all likelihood incompatible with the one created by the custom code in D'Agents. On the other hand, the D'Agents agent programming model is unique in any case, because of the execution state capture, which means that D'Agent agents are unlikely to be compatible with other systems.

3.3.2 Communication

D'Agents implements synchronous messages and events (they are essentially the same thing in D'Agents, as messages do not return anything). Stream communication is also supported. There is no location transparency: Messages must be sent to the host where the receiver agent resides.

3.3.3 Documentation

Since D'Agents is an academic project, there is a lot of written material available. Unfortunately the most important one, the tutorial, has not been updated since version 1.1. The is therefore not known for certain what kind of Java API the system will have.

3.3.4 Security

D'Agents uses the Pretty Good Privacy (PGP) [33] software as an external module to accomplish encryption. Data to be encrypted is saved in a file, encrypted and/or signed using PGP, and the result is then sent to its destination. Both agents and messages can be encrypted to avoid interception, and digitally signed to reliably identify their owner [7].

PGP uses RSA [26] public key cryptography for authentication, and the IDEA [19] algorithm for encryption. The problem with these security features is the slowness of the RSA algorithm; using PGP introduces a significant performance penalty. In addition, implementing PGP use as a separate process introduces additional overhead when compared to a more tightly integrated encryption system.

Agent can be either anonymous or owned. Like in Concordia, agent permissions depend on the owner of the agent. Anonymous agents are untrusted and are allowed minimal access to resources if they are accepted at all.

3.3.5 Reliability

At the moment, the only reliability–oriented mechanism is a *docking system* (see Figure 1, where mobile computers are assigned a docking station where agents can dock to wait for the mobile computer to be connected to the network. When the mobile computer connects to the network, it alerts the docking station, which wakes up all docked agents. However, the docking system currently only handles migration—it does not help when an agent tries to send a *message* to a disconnected mobile computer.

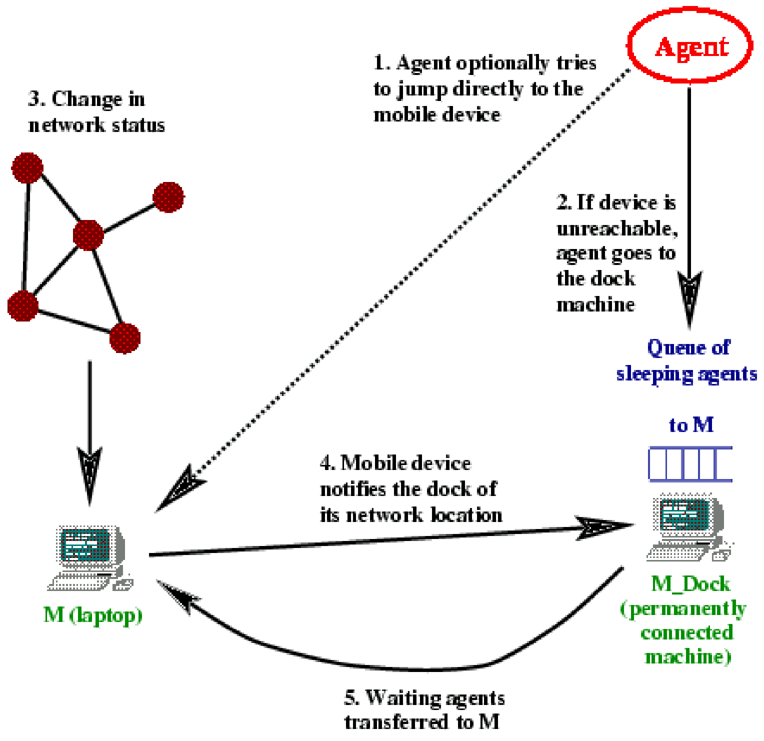
Agent
1. Agent optionally tries to jump directly to the mobile device
3. Change in network status
2. If device is unreachable, agent goes to the dock machine
Queue of sleeping agents
to M
4. Mobile device notifies the dock of its network location
M (laptop)
M_Dock (permanently connected machine)
5. Waiting agents transferred to M

Figure 1. D'Agents docking system

Reliability is one of the main deficiencies in the current version of D'Agents, and thus also one of the most important directions for further development. A checkpointing practice similar to that in Concordia is planned, and partially implemented. In addition, the system designers are considering methods to guard against unacceptable delays due to server crashes.

3.3.6 User Interface

While graphical user interfaces for agents can easily be constructed using Agent Tcl and Tk, there is no graphical user interface for server management. Agents can be tracked by a special debugging tool, which allows agent movement and communications to be monitored. However, the tool only supports Tcl and works by instrumenting agent code.

3.3.7 Standard Support

Currently D'Agents does not support any standards, but the people behind the system express an interest in supporting MASIF once it becomes accepted. However, the unique state capture model of D'Agents may present problems here.

3.3.8 Distinguishing features

D'Agents supports multiple languages: Tcl, Java and Scheme. In this regard, the system is unique of those reviewed here. D'Agents is also the only system reviewed here that has at least partially taken into account mobile computers which are only intermittently connected to the network. This is accomplished through a *docking system* [9] (see **Reliability** above).

A *debugging tool* [13] can be used to instrument agent code, which allows execution, movement and communications to be monitored. However, the tool currently only supports Tcl.

A *Yellow Pages* service [12] allows an agent to identify, locate and use previously unknown services. A service agent announces a service by posting an interface description of the service.

For non–programmers, D'Agents offers a simple *graphical agent construction program* [12], which allows Tcl agents to be constructed from predefined components. The principle is similar to Java Beans [30].

An important justification for mobile agents is the performance improvement they provide by migrating code and using local messaging. However, when bandwidth is abundant and CPU time at the premium, it may be more efficient to use remote messaging instead of agent movement. Deciding on the *optimal migration strategy* is a difficult issue, and subject to ongoing research by the D'Agents team [12]. Network monitoring, machine monitoring and a decision–making library are under development.

3.3.9 Openness

Since source code is available, D'Agents can be regarded as a fully open system. The only caveat is that code for the modified Java runtime requires a license from Sun Microsystems.

3.3.10 Evaluation

D'Agents is first and foremost an open, academic system. For example, while it offers a very limited feature set (e.g. with respect to messaging), on the other hand it is the only agent system presented here to explicitly support mobile computing. This is typical of academic systems, for full functionality is not necessary for academic purposes. Instead, the area covered by the system is very broad and branching, spiraling into whatever directions the developers have thought interesting, or a suitable student project.

It should be noted that while D'Agents is no longer called AgentTCL, it is still a very Tcl–centric system. In particular, features are first implemented in Tcl, and then moved to Java. This means that Java language features are not taken advantage of.

Advantages:

- Full source code available (but license from Sun Microsystems is required)
- Academic developer makes co–operation easier
- Agent movement captures execution state
- Language support for multiple languages: Tcl, Java and Scheme

- Simple agents are often easier to code in Tcl
- Security is on the same level (and partially above) as that of Concordia
- Docking system takes mobile computing into account

Disadvantages:

- Java version is not yet available
- Only supports Java 1.0.2: No RMI or Object Serialization
- Java is a secondary language in the project
- Agent communication is very primitive, only synchronous messages and stream transfer
- Reliability is only beginning to get noticed
- No standard support

3.4 Voyager

Voyager [23, 24] is developed by Objectspace, the people behind the Java Generic Library [22] (Java version of STL), which has been incorporated into the Java Foundation Classes [28]. The Voyager system is now in version 2.0 Beta, which consists of about 280 core classes (ca. 770 kB). It is by far the most professional and feature–rich product evaluated here. It is worth noting that Voyager is the only system that was developed as product (as compared to a research project) from the beginning. One of the main Voyager features is its striving towards distribution transparency; where possible, distribution has been hidden from the programmer.

3.4.1 Movement

The move command takes as parameters the target system and the name of the method that should be called once the target system has been reached. Although itineraries like those in Concordia are not explicitly supported, they are extremely simple to implement. However, there is no GUI for managing itineraries.

Classes are not fetched on–demand; instead, all agent classes are moved when the agent moves. Unnecessary class transfer can be avoided by using interfaces.

3.4.2 Communication

Voyager provided a comprehensive array of messaging alternatives: Synchronous, asynchronous, one–way and (software) multicast messages are all catered for. Messaging is both location– and migration–transparent (via a federated directory service and message forwarders), and timeouts can be specified for asynchronous messages. The hierarchical "Space" messaging architecture (see Figure 2) provides for scalable group communication. In addition, "smart messages" [24], which are essentially miniature agents, can implement things like store-and-forward functionality.

Messaging looks like Java invocations to the programmer, but RMI is not used. The necessary stubs are created automatically and on–demand. Distributed events compatible with the Java Beans Event model are also available.

3.4.3 Documentation

Voyager has by far the best documentation of the agent systems. The provided documentation includes a 375–page Userguide, API documentation, a Technical Overview with information on how to get started, a comparison of Voyager 1.0 and other agent systems, and some preliminary documentation on Voyager's upcoming CORBA and transaction services.

3.4.4 Security

Security is one of Voyager's main deficiencies. It only provides for the standard applet–like security model implemented by the Java Security Manager architecture.

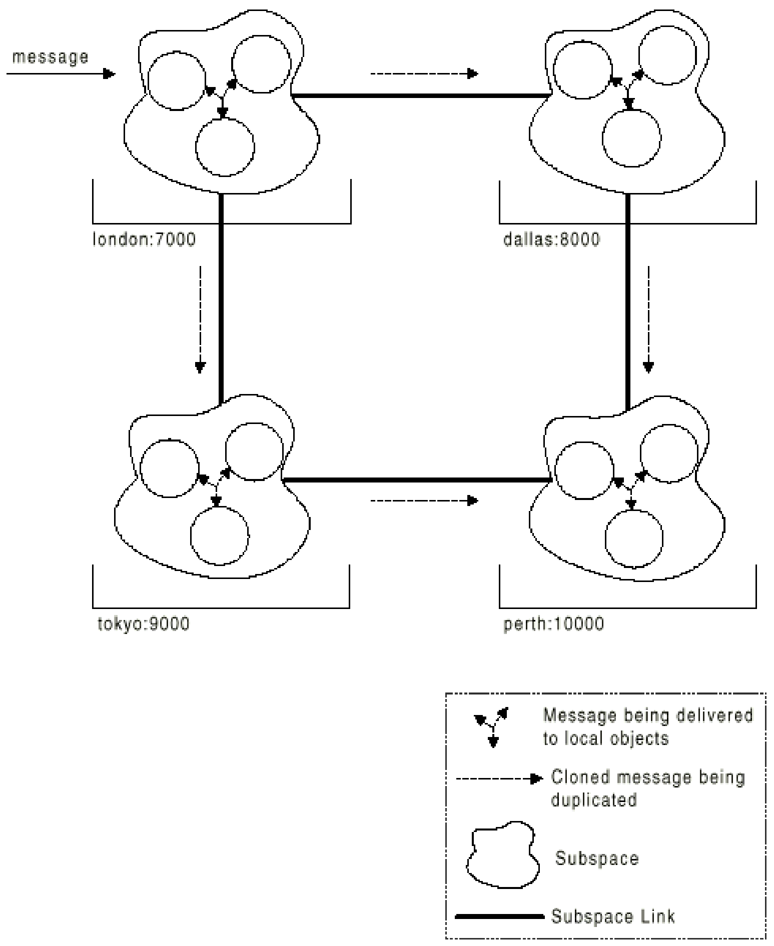
message
london:7000
dallas:8000
tokyo:9000
perth:10000
Message being delivered to local objects
Cloned message being duplicated
Subspace
Subspace Link

Figure 2. The Voyager “Space” messaging architecture

3.4.5 Reliability

Voyager provides its own database service, which can be used to make objects persistent. Persistent objects can be automatically loaded to memory during server startup, which allows a kind of recovery mechanism. Reliable agent movement is also supported.

3.4.6 User Interface

Voyager has no graphical user interface. There are also no utilities for creating and destroying agents, comparable to the management GUIs of other agent systems; while these are not very useful in serious development, where such utilities would probably have to be specially built anyway, it would simplify trying out the system.

3.4.7 Standard Support

The production version of Voyager 2.0 (said to be available in Summer 1998) will provide full CORBA support. Unofficially, Voyager developers have said that "Once MAF becomes a standard it is likely that we will become compliant."

3.4.8 Distinguishing features

The features that set Voyager apart from the rest are its Transaction Service (in development, and not free), CORBA support, superior documentation, and flexible communication system.

3.4.9 Openness

No source code is provided, apart from example programs. However, Voyager does provide a few hooks for extending system functionality. One such hook is the possibility to use your own transport service instead of traditional sockets. A different service can be used for different destination hosts. Other hooks are the "smart" messaging system and a possibility to extend the class loading mechanism. Voyager developers have also talked about more hooks being available in the production version; whether this is true or not remains to be seen.

3.4.10 Evaluation

Two things characterize Voyager:

1. Professional standard. Unlike other systems, Voyager is more a product than an experiment. This is not to say that it does not contain bugs.
2. The Voyager philosophy seems to be "by coders, for coders". There is no attractive GUI, the example agents are simple and unassuming, each highlighting a specific feature of the system. But substantial effort has been invested into making agent programming as painless as possible—assuming you are a coder.

Advantages:

- Professional product.
- Superior documentation.
- Advanced messaging facilities—only system to support migration transparency.
- Distributed Java Beans compatible events.
- Reliability support.
- Extension APIs.
- Transaction Service (in development, and not free).
- Full CORBA support (production version).

Disadvantages:

- It is for the moment unsure exactly how extensible the production version will be, and if that will be enough for the purposes of the project. It may be necessary to license the source code, the cost of which is unknown.
- Security is nonexistent, and reliability support is not all that extensive either.
- While not huge, the system is certainly large.
- No graphical user interface.

3.5 Comparison and Conclusions

Table 1. Comparison of agent systems

System	**Advantages**	**Disadvantages**
Aglets	Good Messaging	Extension APIs
	Good GUI	Reliability
	MASIF involvement	
	Basic security	
	KQML support	
Concordia	Good Security	Extension APIs
	Good reliability	Unfinished system
	Good GUI	Unfinished docs
	Agent control	Only RMI comm.
		Large system
D'Agents	Source code	*No Java yet*
	Many languages	Only Java 1.0.2
	Good security	Primitive comm.
	Mobility support	Primarily TCL
	Academic project	No reliability
Voyager	Professional product	Extensibility?
	Superior docs	No security
	Advanced comm.	No GUI
	Reliability support	No management utils
	Full CORBA support	Large system
	Transaction service	

As Table 1 shows, there is no single best system. Each system is good in some areas, and poor in others. However, Voyager comes very close to achieving supremacy. The suitability of the systems is outlined below:

Aglets Aglets is the best introductory system, suitable for those who want to try out agent programming. It is easy to install, and has an simple but effective GUI for controlling the agents.

Concordia The designers of Concordia have paid special attention to reliability and security, and in this area Concordia has the upper hand.

D'Agents D'Agents is an academic system, and thus much more "open" than the other systems. Full source code is freely available, making the system uniquely suitable for research work. In addition, the system supports several languages, whereas the other systems only support Java.

Voyager Voyager is by far the most advanced system of the four, offering a vast array of features. It also goes to great lengths to make agent programming easy and intuitive. For generic, commercial agent applications it is without doubt the best choice.

References

[1] *T. Alanko, M. Kojo, M. Liljeberg, and K. Raatikainen.* Mowgli: Improvements for Internet Applications Using Slow Wireless Links. In *Proceedings of the 8th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Helsinki, Finland, September 1997.

[2] *Anurag Acharya, M. Ranganathan, and Joel Saltz.* Sumatra: A Language for Resource-aware Mobile Programs. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet*, volume 1222, pages 111–130. Springer-Verlag: Heidelberg, Germany, 1997.

[3] *T. Berners-Lee, R. T. Fielding, and H. F. Nielsen.* Hypertext Transfer Protocol – HTTP/1.0. Internet draft, IETF, HTTP Working Group, December 1994. Available at
<URL:http://www.ics.uci.edu/pub/ietf/http/
draft-fielding-http-spec-01.ps.Z>.

[4] Dartmouth College. D'Agents. Available at
<URL:http://www.cs.dartmouth.edu/~agent/>.

[5] *Marlena Erdos, Bret Hartman, and Marianne Mueller.* Security Reference Model for the Java Developer's Kit 1.0.2. Technical report, Sun Microsystems, Inc., 1996.

[6] *Alan O. Freier, Philip Karlton, and Paul C. Kocher.* The SSL Protocol Version 3.0, 1996. Available at <URL:http://www.netscape.com/eng/ssl3/draft302.txt>.

[7] *Gray R. S., Cybenko G., Kotz D., and Rus D.* D'Agents: Security in a multiple-language, mobile-agent system. In Giovanni Vigna, editor, *Mobile Agent Security*, Lecture Notes in Computer Science. Springer-Verlag, 1998. To appear.

[8] GMD FOKUS and IBM Corporation. Mobile Agent System Interoperability Facilities Specification, November 1997. Standard proposal.

[9] *Gray R. S., Kotz D., Nog S., Rus D., and Cybenko G.* Mobile Agents for Mobile Computing. Technical Report PCS-TR96-285, Dartmouth College, Computer Science, Hanover, NH, May 1996.

[10] *Gray R., Kotz D., Nog S., Rus D., and Cybenko G.* Mobile agents: The next generation in distributed computing. In *Second Aizu International Symposium on Parallel Algorithms/Architectures Synthesis (pAs '97)*, pages 8–24, Fukushima, Japan, March 1997. IEEE Computer Society Press.

[11] *Gray R. S.* Agent Tcl: A Transportable Agent System. In *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, Maryland, 1995.

[12] *Gray R. S.* Agent Tcl: A flexible and secure mobile-agent system. Technical Report PCS-TR98-327, Dartmouth College, Computer Science, Hanover, NH, January 1998.

[13] *Hirschl M. and Kotz D.* AGDB: A Debugger for Agent Tcl. Technical Report PCS-TR97-306, Dartmouth College, Computer Science, Hanover, NH, February 1997.

[14] IBM alphaWorks. JKQML. Available at <URL:http://www.alphaWorks.ibm.com/formula/jkqml>.

[15] IBM Corporation. Aglets Software Development Kit. Available at <URL:http://www.trl.ibm.co.jp/aglets/>.

[16] *Danny B. L. and Yariv A. Agent Transfer Protocol – ATP/0.1* . IBM Tokyo Research Laboratory, March 1997.

[17] *Danny B. L. and Daniel T.* Programming Mobile Agents in Java – A White Paper. Technical report, IBM Corp., 1996.

[18] *Liljeberg M., Helin H., Kojo M., and Raatikainen K.* Enhanced Service for World-Wide Web in Mobile WAN Environment. Technical Report C-1996-28, Department of Computer Science, University of Helsinki, 1996. Revised version published in *Proceedings of the IEEE Global Internet 1996 Conference*, London, England, November 20-21, 1996.

[19] *Lai X., Massey J. L., and Murphy S.* Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *Proceedings of Advances in Cryptology (EUROCRYPT '91)*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer-Verlag: Heidelberg, Germany, 1991.

[20] Mitsubishi Electric Information Technology Center America. Concordia. Available at <URL:http://www.meitca.com/HSL/Projects/Concordia/>.

[21] Mitsubishi Electric Information Technology Center America. Mobile agent computing. A White Paper, January 1998.

[22] ObjectSpace, Inc. ObjectSpace JGL – The Generic Collection Library for Java. Version 3.0. Available at <URL:http://www.objectspace.com/jgl/>.

[23] ObjectSpace, Inc. ObjectSpace Voyager. Available at <URL:http://www.objectspace.com/voyager/>.

[24] ObjectSpace, Inc. ObjectSpace Voyager Core Package Techical Overview, December 1997. Version 1.0. Available at <URL:http://www.objectspace.com/voyager/whitepapers/ VoyagerTechOview.pdf>.

[25] Object Management Group. The Common Object Request Broker: Architecture and Specification, February 1998. Version 2.2. Available at <URL:http://www.omg.org/corba/c2indx.htm>.

[26] RSA Laboratories. RSA Encryption Standard, 1993. Available at <URL:http://www.rsa.com/rsalabs/pubs/PKCS/ps/pkcs-1.ps.gz>.

[27] Sun Microsystems. JAR – Java Archive. Available at <URL:http://java.sun.com/products/jdk/1.1/docs/guide/jar/>.

[28] Sun Microsystems. Java Foundation Classes. Available at <URL:http://java.sun.com/products/jfc/>.

[29] Sun Microsystems. Java IDL. Available at <URL:http://java.sun.com/products/jdk/1.2/docs/guide/idl/>.

[30] Sun Microsystems. *Java Beans(TM)*, July 1997. Graham Hamilton (ed.). Version 1.0.1.

[31] Sun Microsystems. Java Remote Method Invocation – Distributed Computing for Java. White Paper, March 1998.

[32] *Wong D. et al.* Concordia: An Infrastructure for Collaborating Mobile Agents. In K. Rothermel and R. Popescu-Zeletin, editors, *Proc. First Int. Workshop on Mobile Agents*, volume 1219 of *Lecture Notes in Computer Science*, pages 86–97, Berlin, 1997. Springer-Verlag, Berlin.

[33] *Zimmermann P.* Pretty Good Privacy. Available at <URL:http://www.pgpi.com/>.