OBJECTSPACE VOYAGER CORE TECHNOLOGY

THE

AGENT ORB

FOR JAVA

Voyager and Agent Platforms Comparison

ObjectSpace Voyager™ Core Technology (Voyager) is a simple yet powerful object request broker (ORB) for creating distributed Java applications. Voyager contains a superset of features found in other ORBs and agent platforms, including CORBA, JavaSoft's RMI, General Magic's Odyssey™, IBM's Aglets™, and Mitsubishi's Concordia. Although Voyager can be used to supplement these technologies, it effectively replaces them with a single, easy-to-use platform.

This paper compares Voyager Version 1.0 with the early beta version of Odyssey, the alpha version of Aglets, and the alpha version of Concordia. Although on the surface these products appear similar, there is a significant difference between the philosophy behind Voyager and its competitors.

OBJECTSPACE

© 1997 ObjectSpace, Inc. • All rights reserved • Version 1.0 • December 1997

Contents

Overview 1
Remote Messaging 3
Life Spans 4
Directory Service 5
Mobility 6
Itineraries 8
Persistence 9
Scalability 10
Multicast Messaging 11
Distributed Events 12
Publish/Subscribe 13
Applet Connectivity 14
Security 15
Availability and Pricing 16
Product Size 17
Browser Compatibility 18
Performance 19
Future Direction 20

Overview

ORBs that support fundamental distributed computing, such as CORBA, DCOM, and RMI, allow developers to create remote objects and send them messages as if they were local. They often include features such as distributed garbage collection, different messaging modes, and a naming service. However, none of them support object mobility or mobile, autonomous agents.

Agent platforms like Odyssey, Aglets, and Concordia allow developers to create an agent, program it with a set of tasks, and launch it into a network to fulfill its mission. However, they have minimal support for basic distributed computing and treat agents differently than simple objects. Aglets uses sockets, and Odyssey and Concordia use RMI to move agents between machines. But, none of these platforms allow sending a regular Java™ message to a stationary or moving agent. As a result, it is very difficult for objects to communicate with an agent after the agent has been launched and for agents to communicate directly with other agents.

Voyager is the first platform to seamlessly integrate fundamental distributed computing with agent technology. Voyager was designed from the ground up to support mobile objects and autonomous agents. The Voyager philosophy is that an agent is simply a special kind of object that can move independently, can continue to execute as it moves, and otherwise behaves exactly like any other object. Voyager enables objects and other agents to send standard Java messages to an agent even as the agent is moving. In addition, Voyager allows you to remote-enable any Java class, even a third-party library class, without modifying the class source in any way. Voyager also includes a rich set of services for transparent distributed persistence, scalable group communication, and basic directory services.

The following table summarizes the features of Voyager Version 1.0, the early beta version of Odyssey, the alpha version of Aglets, and the alpha version of Concordia. Each feature is described in detail in the following sections. For a detailed comparison of Voyager and RMI, please consult the technical white papers section on the Voyager home page at www.objectspace.com/voyager/technical_white_papers.html.

Feature	Voyager (Version 1.0)	Odyssey (Beta Version)	Aglets (Alpha Version)	Concordia (Alpha Version)
Remote Messaging				
Creating agents remotely	✓			
Sending Java messages remotely	✓			
Sending Java messages to mobile agents	✓ Transparent			
Messaging modes between agents	✓ Synchronous, one-way, future, one-way multicast		✓ Synchronous, future	
Life Spans	✓ Five modes	✓ Explicit	✓ Explicit	✓ Explicit
Directory Service				
Mobile directory service	✓ Integrated			
Federated directory service	✓			
Object Mobility	✓			✓ Limited
Agent Mobility				
Moving to a program	✓	✓	✓	✓
Moving to an object	✓			
Itineraries	✓ No special API needed	✓ Special API needed	✓ Special API needed	✓ Special API needed
Persistence				
Object persistence	✓			
Agent persistence	✓			✓
Database integration	✓ Integration with most popular databases			✓ Proprietary database only
Database independence	✓			
Flushing/autoloading	✓			
Scalability	✓ Space™ architecture			
Multicast Messaging	✓			
Distributed Events	✓			Unable to verify
Publish/Subscribe	✓			
Applet Connectivity	✓ Full	✓ Restricted	✓ Restricted	✓ Restricted
Security	✓ Security manager		✓ Security manager	✓ Security manager
Pricing	Free for most commercial uses	Free noncommercial	Free noncommercial	Free noncommercial
Product Size	270KB	242KB	120KB (without ORB)	175KB+
Browser Compatibility				
Netscape Navigator 4.0	✓			
Internet Explorer 4.0	✓			
Performance	See "Performance" on page 6.			
Future Direction	Rich feature set publicly stated	Not stated	Not stated	Not stated

Remote Messaging

One of the most compelling visions of the future is a world in which specialized agents collaborate with each other to achieve a goal that an individual agent cannot achieve on its own. These agents would need to move, when appropriate, to conduct high-bandwidth conversations that could not possibly take place over a low-bandwidth network. A platform for building these systems, therefore, would have to support a powerful messaging system that allows objects and mobile agents to communicate seamlessly with each other, even if they are moving across a network.

ObjectSpace Voyager

Voyager allows an agent to send regular Java messages to other agents, even if they are moving and regardless of where they are in the network. Voyager allows you to remote-enable a Java class without modifying it in any way. Once remote-enabled, a class can be easily instantiated anywhere in a network and can be sent regular Java messages. Voyager supports synchronous, one-way, one-way multicast, and future message modes. Each time an object moves, it leaves behind a forwarder object that forwards messages to the object's new location. When an object dies, its forwarders also die. Because Voyager treats an agent like any other object, you can also create remote agents and send them messages as they move.

The example below demonstrates sending a message to an agent that has moved.

```
//create MyAgent in tokyo
VMyAgent agent = new VMyAgent( "tokyo:8000" );

//roam network for best price
agent.findBestPriceFor( "widget" );

... //time period passes during which the agent moves

//send Java message to agent at new location
int price = agent.getBestPrice();
```

Odyssey, Aglets, and Concordia

Creating a remote agent in Odyssey, Aglets, and Concordia is not easy—an agent must be built locally. Once an agent has been launched, none of these platforms allow you to send the agent a regular Java message, even if the agent is stationary.

Aglets allows you to send a string command to a stationary agent at a well-known URL, but the agent is responsible for decoding and executing the command, which is a tiresome programming chore. Concordia provides a stationary AgentGroup object that agents must interact with to communicate with each other.

Both Aglets and Concordia make it difficult for objects to communicate with agents and for agents to collaborate across a network.

Life Spans

The life span of an object defines how long the object lives before it dies and is reclaimed by the system. ORBs typically contain a distributed garbage collector that kills objects when they have no more local or remote references. Agents, on the other hand, can roam a network independent of external references and thus require a more flexible life span scheme.

ObjectSpace Voyager

Voyager supports a variety of life spans:

- An object can live until there are no more local or remote references to it. (This is the default life span of a simple object.)
- An object can live for a certain amount of time. (The default life span of an agent is one day.)
- An object can live until a particular point in time.
- An object can live until it becomes inactive for a specified amount of time.
- An object can live forever.

You can change an object's life span at any time.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia do not support a garbage collection scheme. An agent lives forever until it either explicitly tells itself to die or is explicitly told to die.

Directory Service

Most platforms for distributed computing support a naming service that enables connecting to an existing object based on a name. This is particularly useful for launching a mobile agent from one application to another and then locating it after it moves.

ObjectSpace Voyager

Voyager contains an integrated directory service that allows you to associate an alias with any object you create and connect to the object via its alias at a later time, even if it has moved.

The following example demonstrates how to work with a Voyager alias.

```
// application 1
// create agent in tokyo with alias "David"
VMyAgent agent = new VMyAgent( "tokyo:8000/David" );
agent.findBestPriceFor( "widget" ); // remote Java message

// application 2
// connect to agent based on its last known location
VMyAgent agent = (VMyAgent) VObject.forObjectAt( "tokyo:8000/David" );
int price = agent.getBestPrice(); // remote Java message
```

Voyager's directory service allows you to create and connect network directories together to form a large federated directory service. You can associate an object with a hierarchical name such as sports\basketball\lakers or chemistry\symbols\calcium. The federated directory service is fully integrated with Voyager's persistence subsystem.

Odyssey, Aglets, and Concordia

Both Odyssey and Aglets include simple, local lookup tables that enable associating a string with an agent's URL. Neither platform contains a distributed directory service that enables connecting to a moving agent based on an alias. Without this feature, locating a moving agent is difficult.

Concordia has no support for naming except RMI's primitive, bootstrapping naming service.

Mobility

Most distributed computing platforms support the movement of agents between programs.

ObjectSpace Voyager

Voyager supports mobility of all serializable objects, not just agents. You need not modify a class in any way to get full mobility support. There are several reasons why object mobility is useful.

- If two objects need to have a high-bandwidth conversation, they can move closer to each other to speed communications and reduce network traffic. Local messaging is often 10,000 to 1,000,000 times faster than remote messaging.
- If an object requires features such as persistence or a fast processor, the object can move to a machine that has these features.
- If a machine containing an object is about to be disconnected from the network, the object can move to another machine to continue its execution.

You can move an object to a new program, even while the object is receiving messages. Because Voyager handles all synchronization and message-forwarding issues, the code required to move an object is very simple, as shown below.

```
// create instance of MyObject in tokyo
VMyObject object = new VMyObject("tokyo:9000" );
...
// move object to dallas
object.moveTo( "dallas:7000" );
```

Voyager also supports agent mobility. To move to another program, an agent can send itself the moveTo() message with the address of the destination and the name of a method. The method is executed with no arguments when the agent arrives.

```
class Traveller
 {
 public void travel()
  {
  moveTo( "tokyo:8000", "atTokyo" );
  }

 public void atTokyo()
  {
  System.out.println( "I am at tokyo" );
  }
 }
```

Unlike most platforms, Voyager allows agents to move to a particular *object* as well as to a particular program. To have a high-speed conversation with an object, the agent can send itself the moveTo() message with the address of the target object and the name of a

method. When the agent arrives, the method is executed with a reference to the target object.

```
class Librarian
 {
 String home; //where i came from
 Vector titles; //for storing titles

 public void collectTitlesFromLibrary()
  {
  home = Voyager.getAddress(); //remember home

//visit library object
  moveTo( "dallas:8000/library", "atLibrary" );
  }

 public void atLibrary( Library library )
  {
  System.out.println( "I am at the library in dallas" );

//store collection of book titles
  titles = library.getTitles();

//move back home
  moveTo( home, "atHome" );
  }

 public void atHome()
  {
  System.out.println( "titles = " + titles );
  }
 }
```

Voyager's mobile callback technique allows you to easily specify how an agent is resumed. An agent can move to any Java object for a high-speed conversation, even if the object itself is mobile.

Odyssey, Aglets, and Concordia

Odyssey and Aglets allow an agent to move itself to a well-known location. When the agent arrives, it is sent a single message that never varies, which often results in an agent's code resembling a large, multiway branch statement.

```
public void onArrival()
 {
 if( <at dallas library> )
  //...
 else if( <at home> )
  //...
 else if( <at tokyo> )
  //...
 }
```

Neither Odyssey nor Aglets allows an agent to move to an object, even if the object is stationary. In addition, neither platform supports mobility of simple Java objects.

Concordia's support for mobility is limited to itineraries, described in the next section.

Itineraries

An itinerary is a collection of tasks to be performed at a sequence of locations.

ObjectSpace Voyager

Programming an itinerary using Voyager is simple. The locations to be visited can be stored as a Vector of addresses, and an agent can move easily from location to location, performing tasks as it travels.

The following code defines a Voyager agent that visits a list of locations and displays a message at each one:

```
class MyAgent
 {
 Vector itinerary = new Vector(); //list of places to visit
 int index = 0;

 public void addLocation( String address )
  {
  itinerary.addElement( address );
  }

 public void launch()
  {
  //pass the current index as a callback argument
  visit( itinerary.elementAt( index ), "visit" );
  }

 public void visit()
  {
  System.out.println( "at " + itinerary.elementAt( index ) );

  if( ++index < index.size() )
    visit( itinerary.elementAt( index ) );
  else
    System.out.println( "finished" );
  }
}
```

The locations in a Voyager itinerary can consist of an arbitrary combination of target objects and programs. Voyager uses regular Java features to execute an itinerary, thus not requiring a special itinerary API within the Voyager platform.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia each include a special itinerary API that achieves the same behavior as Voyager, but requires more complex programming.

Persistence

A persistent object has a backup copy in a database. A persistent object is automatically recovered if its program is terminated or if it is flushed from memory to the database.

ObjectSpace Voyager

Voyager includes seamless support for database-independent persistence. In many cases, you can persist an object without modifying its source in any way.

Every Voyager program can be associated with a database. The type of database can vary from program to program and is transparent to a Voyager programmer. Voyager includes a high-performance object storage system called VoyagerDb, but Voyager is designed to work with most popular relational and object databases.

To save an object to its program's database, you can send it a saveNow() message. This method causes a copy of the object to be written to the database, overwriting any previous copy. If the program is shut down and then restarted, the persistent object is left in the database. An attempt to communicate with a persistent object causes the object to be immediately reloaded from the database.

If a persistent object is moved from one program to another, the copy of the object is automatically removed from the source program's database and added to the destination program's database.

To conserve memory, you can use one of the flush() family of methods to flush a persistent object from memory to a database. As before, a subsequent attempt to communicate with the flushed object causes the object to be reloaded immediately from the database.

An object can be programmed to restart automatically when a persistent server is rebooted. By default, Voyager's database system persists Java classes loaded into a program across a network so they need not be reloaded when the program is restarted.

Odyssey, Aglets, and Concordia

Neither Odyssey nor Aglets supports persistence. Concordia has proprietary support for persistence of agents only.

Scalability

Many distributed systems require features for communicating with groups of objects. Consider the following examples.

- Stock quote systems use a distributed event feature to send stock price events to customers around the world.
- A voting system uses a distributed, multicast messaging feature to send messages to voters around the world and ask them for their views on a particular matter.
- News services use a distributed publish/subscribe feature so that each broadcast is received only by readers interested in the topic of the broadcast.

Most traditional systems use a single repeater object to replicate the messages and events to each object in a target group. This approach works well if few objects reside in the target group, but does not scale well when large numbers of objects are involved.

ObjectSpace Voyager

Voyager uses an innovative architecture for message and event replication called Space™ that can scale to global proportions. Clusters of objects in the target group are stored in local groups called subspaces. The subspaces are linked together across a network to form a larger logical group, or Space. When a message or event is sent into one of the subspaces, the message or event is cloned to each of the other subspaces in the Space before being delivered to every object in every subspace. This results in a rapid, parallel fanout of the message or event to every object in the Space. A special marker mechanism in each subspace ensures that no message or event is accidentally processed more than once, regardless of how the subspaces are linked together.

Voyager's multicast messaging, distributed events, and publish/subscribe features all use and benefit from the same underlying Space architecture.

Odyssey, Aglets, and Concordia

Odyssey and Aglets do not include a scalable architecture for multicast messaging, distributed events, or publish/subscribe features.

Concordia claims a simple events system, but examples of such a system could not be obtained. Concordia does not support multicast messaging or publish/subscribe.

Multicast Messaging

ObjectSpace Voyager

Voyager includes seamless support for large-scale multicast messaging that does not require modifying your classes in any way. To perform multicast messaging, add objects to a Space, establish a virtual reference to the Space, and send the Space a message as if the Space were a single object. The message is propagated in a fault-tolerant and parallel fashion to every object in the Space.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia do not support Java message multicasting.

Distributed Events

ObjectSpace Voyager

Voyager includes seamless support for large-scale distributed JavaBeans™ events. To send an event to a group of objects, first process the event listener class using VCC. Then, add the group of objects to a Space and attach the appropriate virtual event listener to the Space. Finally, add the virtual event listener to the event source. When an event is sent to the virtual event listener, the event is sent to every object in the Space that implements the appropriate event listener interface. The Voyager events system allows you to send any JavaBeans event to a network of distributed listeners without modifying the beans in any way.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia do not support distributed JavaBeans events.

Publish/Subscribe

ObjectSpace Voyager

Voyager includes seamless support for large-scale, distributed publish/subscribe of messages and events. To send a message or event to all objects in a Space interested in a particular subject, use a *OneWayMulticast* message with a selector. All objects in the Space that are registered subscribers of the selected subject receive the message or event. The Voyager publish/subscribe feature is 100% nonintrusive and does not require modifying the communicating objects in any way.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia do not support publish/subscribe.

Applet Connectivity

Most browsers allow an applet to establish a network connection only to its server. As a result, most platforms for distributing computing (like RMI and many CORBA implementations) allow an applet to communicate only with objects located on the same server.

ObjectSpace Voyager

Voyager includes a lightweight software router that allows a server to act as a gateway, thereby enabling full applet-to-applet and applet-to-program connectivity. The router also allows Voyager agents to move freely between applets and programs.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia do not allow agents to move between applets or between applets and programs. Thus, an agent created with any of these platforms is significantly restricted in its ability to move.

Security

The Java system supports the concept of security managers—installable "watchdogs" that prevent unauthorized code from executing a preset variety of operations.

ObjectSpace Voyager

Voyager includes a VoyagerSecurityManager class that can be installed to restrict the operations that mobile agents can perform. The source code for this security manager is available at the ObjectSpace Web site and can be tailored to your specific security needs.

Odyssey, Aglets, and Concordia

Odyssey does not include a security manager. Aglets contains an AgletSecurityManager class, but does not include source that you can customize to your specific requirements.

Availability and Pricing

ObjectSpace Voyager

Voyager is free for commercial use in regular PC/UNIX environments. Licenses for source code and deployment in non-PC/UNIX environments are available based on a flexible pricing scheme. Voyager 1.0 was released in mid-September of 1997. Voyager 2.0 Beta 1, which includes CORBA integration, was released in early December 1997. Version 2.0 Beta 2 is scheduled for release in January 1998, and the 2.0 production version is scheduled for release by the end of the first quarter of 1998.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia are free for noncommercial use only. Odyssey is currently in early beta release, and Aglets and Concordia are currently in alpha release. General Magic, IBM, and Mitsubishi have not confirmed their plans to release these packages as commercially supported products.

Product Size

Voyager

The total size of Voyager is about 270KB. Because Voyager is a self-contained ORB, it does not require RMI, CORBA, or any other distributed computing platforms.

Odyssey, Aglets, and Concordia

The Odyssey kernel is 60KB, and the worker package which supports simple itineraries is 7KB. Odyssey also requires RMI, which is an additional 175KB. Thus, the total size is 242KB. Because Odyssey uses RMI, you must also run an `rmiregistry` name server and a Web server on the server side of your application. This increases the size and complexity of installing Odyssey in embedded systems or in an intranet.

The size of the Aglets kernel is 120KB. If you want any fundamental distributed computing functions, you also have to use CORBA or RMI.

We were unable to get an evaluation copy of Concordia to determine its size. It relies on RMI; therefore, its size is estimated to be at least 175KB.

Browser Compatibility

ObjectSpace Voyager

Voyager does not rely on RMI or any other platform for distributed computing. Voyager has been tested for compatibility with HotJava, Netscape Navigator 4.0, and Internet Explorer 4.0.

Odyssey, Aglets, and Concordia

Odyssey, Aglets, and Concordia use RMI, which is not currently compatible with Netscape Navigator 4.0. In addition, Microsoft publicly stated that Internet Explorer will not support RMI. The incompatibility of these agent platforms with Microsoft browsers is a serious disadvantage.

Performance

ObjectSpace Voyager

For common use, Voyager messaging is comparable to RMI and functions at about the same speed. In some cases, when messages are delivered to objects using the Voyager `moveTo()` method, Voyager proves to be significantly faster.

Odyssey, Aglets, and Concordia

Performance figures for messaging in Odyssey, Aglets, and Concordia are meaningless because these platforms do not support sending Java messages to agents.

Future Direction

ObjectSpace Voyager

ObjectSpace strives to make Voyager the preferred platform for building distributed Java applications, regardless of whether the applications make use of object mobility or mobile, autonomous agents.

Listed below are our publicly stated goals for future versions of the ObjectSpace Voyager Core Technology and Voyager add-ons:

- Seamless CORBA and DCOM integration (CORBA integration already implemented in Version 2.0 Beta 1, released December 1997)
- A system similar to an air-traffic controller system that allows real-time visualization and control of distributed objects
- Object auditing
- Trading service
- Agent collaboration service
- Quality of service for messages
- Support for the “Linda” paradigm of distributed computing
- Message store and forward
- Agent store and forward
- Rule-based security system to supplement the standard Java security scheme
- Ability to handle class versioning
- Object replication

Odyssey, Aglets, and Concordia

General Magic, IBM, and Mitsubishi have not publicly stated any future goals for their respective platforms.

For additional technical information on ObjectSpace products and programs or for information on how to order and evaluate ObjectSpace technology, contact us today!

14850 Quorum Drive, Suite 500
Dallas TX 75240

972.726.4100
1.800.OBJECT.1
Fax: 972.715.9099

E-mail: sales@objectspace.com
Web: www.objectspace.com

Dallas • Austin • Chicago •
San Francisco • Washington DC

Java is a trademark of Sun Microsystems.
ObjectSpace Voyager and Space are trademarks of ObjectSpace, Inc.
All other trademarks are the property of their respective companies.