Java Aglets

Mobile Agent Implementation
CS525M Multiagent Systems
Alex Weiner

What are Mobile Agents?

- A mobile agent is a software object that has *behavior, state, and location.* The agent architecture will usually implement interagent *messaging* and an *event model* as well.
- Agents exist inside some type of agent server.
- Agents can migrate from server to server carrying their code and state with them.
- Agents can load their code from various locations.

2

Why Mobile Agents?

- **Reduce Network Load** - Allow conversations to happen locally rather than across network.
- **Overcome Network Latency** - Real time systems could have a control agent dispatched to them.
- **Eliminate Protocol Hassles** - Agent can take the conversation to the server, eliminating some of the headache of communications over heterogeneous systems.
- **Reduce Reliance on Connections** - An agent can be deployed, the connection broken and then reestablished later to retrieve agent.
- **Dynamic, Robust and Fault Tolerant** - Mobile agents are able to relocate to improve performance or fault tolerance.

3

Potential Mobile Agent Applications

- **Secure Brokering** - Allow parties to meet on a trusted host where collaboration can take place without worry that the host will aid one party or the other.
- **Distributed Information Retrieval** - Agents roam, collect desired information and eventually return with information of interest.
- **Personal Assistant** - Allows agent to act on the behalf of its creator at remote hosts without fear of connectivity problems.
- **Server Farm Maintenance** - Agent can roam connected computers performing a plethora of tasks - i.e. installs, upgrades, backups, monitoring logs.
- **Monitoring and Notification** - Agents can be dispatched to wait for certain types of information to become available then notify the user or act upon the information.

4

Mobile Agent Systems

- **Aglets by IBM** - Implemented with Java class libraries and mirrors the Java applet model.
- **Odyssey by General Magic** - invented the mobile agent and originally implemented the first commercial mobile agent system in Telescript (their proprietary language). Replaced by Odyssey implemented in Java class libraries.
- **Voyager by ObjectSpace** - Java based.
- **Concordia by Mitsubishi** - Java based.

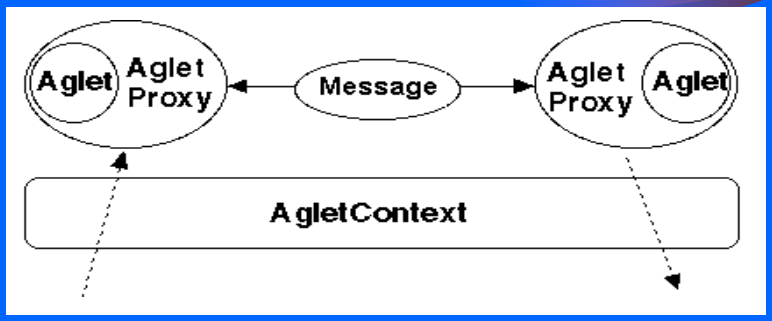
5

The Aglet Model

- **Aglet** - Mobile Java object that runs in its own thread, acts autonomously, visits local and remote hosts, and reacts to events and messages.
- **Proxy** - Provides Aglet with location transparency and a shield from direct access.
- **Context** - Stationary workplace that hosts Aglets. Provides platform resources
- **Identifier** - Globally unique, immutable Aglet identifier. AgletID maintained in an AgletInfo object associated with Aglet.

6

The Aglet Model



7

Aglet Proxy

AgletProxy - Each Aglet has an associated AgletProxy which serves to:

- Shield uncontrolled access to Aglet's public methods. Application or other Aglets will normally make calls to an Aglet's associated AgletProxy object and cannot get direct access to an Aglet unless that Aglet gives out a direct reference to itself.
- Act as handle for local, remote, and deactivated Aglet.
- Aglet can have multiple remote proxies which provide location transparency to those calling the Aglet.

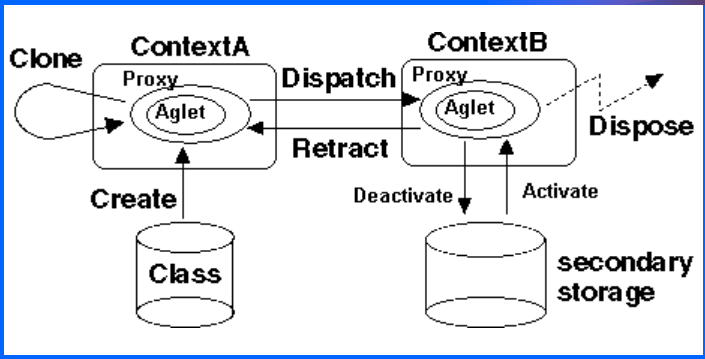
8

Aglet Operations Overview

- **Creation** - created in a context and starts running.
- **Cloning** - Aglet duplicated, but with new ID and execution restarted.
- **Dispatching** - 'Pushed' out from current context and restarted in another with state maintained.
- **Retraction** - 'Pulled' from current context and restarted with state maintained.
- **Deactivation** - Aglet is temporary halted and stored to secondary storage. Activation restores Aglet to same context.
- **Disposal** - Halts execution and removes Aglet from context.

9

Aglet Operations Overview



10

Aglet Operations Creation & Disposal

- **createAglet** - Method of Aglet instantiation. Aglet created by request to Context.

  Methods called: Aglet(), onCreation(param),run()
- **clone** - Method of creating a new Aglet which is identical to another (same state) but restarts execution. CloneListener provides events.
- **Dispose** - Aglet ending its own (or other's) life, allowing resources to be garbage collected. Methods called: onDisposing()

11

Aglet Operations Mobility

- **dispatch** - Push agent to new host (URL). Agent is serialized, sent over network, received, reinstantiated with state (heap not stack), and execution continued in **run** method.
- **retractAglet** - Pull agent back to previous host - requested from Context. Same effect as dispatch, but in 'reverse' direction.
- **MobilityListener** provides events for both methods.

12

Aglet Operations Persistence

- **Deactivate / activate** - All Aglet threads are killed and the Aglet sleeps for a specified time period or until activated by another Aglet or system entity. The Aglet is serialized and moved to secondary storage during this time. When the Aglet is activated, execution starts in **run** method.
- **PersistencyListener** provides events for both deactivation and activation.

13

Aglet Context

Provides Aglet with a uniform execution environment and access to resources on the underlying server while providing the server security against malicious Aglets. Aglets must be instantiated within a context, and at any time belong to exactly one context.

The Aglet uses its context for the following:

- **createAglet** - Create a new Aglet in the context and receive a reference to the new Aglet's proxy.
- **retractAglet -** Retract Aglet back from a remote context.
- **getAgletProxies / getAgletProxy** - Used to retrieve Aglet proxies in local and remote contexts.
- **setProperty/getProperty –** Allows attribute-value pairs to be saved and retrieved in the context.

14

Aglet Code Example Directory Listing

```
package agletbook;
import com.ibm.aglet.*; import
    com.ibm.aglet.event.*;
import java.io.*;
public class ListingAglet extends Aglet {
  boolean retracted = false;
  File dir = null;
  String[] list;
  public void onCreation(Object init) {
    dir = (File)init;
    addMobilityListener(
      new MobilityAdapter() {
        public void
    onArrival(MobilityEvent me) {
        try {
            if (retracted) {
            for (int i = 0; i < list.length; )
                System.out.println(i + ": " +
                list[i++]);   }
            else {
                list = dir.list();
              }
            } catch (Exception e) {
              dispose();
            }
          }
          public void
      onReverting(MobilityEvent me) {
            retracted = true;
          }
        }
      );
    }
}
```

15

Aglet Code Example Directory Listing

```
public class ListingAgletMaster extends Aglet {
public void run() {
    try {
      URL destination = new URL("atp://killi.genmagic.com");
       File directory = new File("C:");
      AgletProxy proxy = getAgletContext().createAglet(getCodeBase(),
            "agletbook.ListingAglet", directory);
      proxy = proxy.dispatch(destination);
      pause();            //Give Aglet time to do its job
      proxy = getAgletContext().retractAglet(destination, proxy.getAgletID());
     proxy.dispose(); }
   catch (Exception e) {
      print("Failed to create the child.");
      print(e.getMessage());
    }  }
```

16

Aglet Communication Overview

Aglets communicate using a location transparent, extensible, blocking or non-blocking, queued and prioritized messaging infrastructure.

- **Message** object is sent to an Aglet's proxy which passes it on to the Aglet.
- **handleMessage** method is used by an Aglet to retrieve and selectively handle messages sent to it.
- The receiving Aglet can use the incoming Message object to send a reply (**sendReply** method).
- **MessageManager** provides serialized and prioritized message queuing for the Aglet.

17

Aglet Communication Message Object

- Message will always have a *kind* parameter (string) to identify it, and optionally an argument (atomic or tabular) for associated data.
- The receiving agent can choose to handle the Message based on its *kind*. If the agent does not handle the message, it will raise a NotHandledException back at the sending agent.
- The receiving agent can respond to a message using the received message's sendReply method, optionally sending back associated data.

18

Aglet Communication Peer-to- Peer Messaging

- **Synchronous** - Sending Aglet will block until it receives a reply or an exception. Result of using sendMessage method.
- **Asynchronous** - Sending Aglet thread will continue execution after sending message using sendFutureMessage method. This method returns a FutureReply object that the Aglet can query to determine if reply has been received.
- **Remote Messaging** - Messages can be sent to remote Aglets in the same way as local Aglets (location transparent). However, objects sent/received must be of a serializable Java type and the class code must exist in at both locations.

19

Aglet Communication Multicasting

- Aglets can send messages to multiple agents in the *local context only* with the multicastMessage context method.
- Aglets in the local context can receive the message by subscribing to the message (Aglet.subscribeMessage method).
- An Aglet sending a multicast message can receive multiple replies using a ReplySet object. The ReplySet object is a container for FutureReply objects. It can be used to determine how many replies have been received at any point and to receive the replies.

20

Aglet Communication Message Management

- Each Aglet has a MessageManager that provides deterministic message handling by placing incoming messages into a queue and forwarding them one at a time to the Aglet's message handler.
- Each Aglet can set a priority for message types that it handles (or allow a default priority be assigned). When a message comes in, it will be placed at the end of the queue based on the priority assigned it.
- By default, message handling is deterministic. A second message will not be delivered from the queue until the handling of the first has been completed.
- Parallel message handling can be employed by allowing a second thread to handle the next message on the queue (using exitMonitor method).
- Messages can be synchronized by allowing the handling of a message to be suspended until a future time (i.e. by receiving another message type).

21

Aglet Code Example Directory Listing with Messages

```
package agletbook;
import com.ibm.aglet.*;import com.ibm.aglet.event.*;import java.io.*;
public class ListingAglet1 extends Aglet {
  AgletProxy _proxy = null;   File _dir = null;
  public void onCreation(Object args) {
    _dir = (File)((Object[])args)[0];
    _proxy = (AgletProxy)((Object[])args)[1];
    addMobilityListener(
      new MobilityAdapter() {
        public void onArrival(MobilityEvent me) {
          try {
            _proxy.sendMessage(new Message("Listing", _dir.list()));
          } catch (Exception e) {
            dispose();
          } } }       );  } }
```

22

Aglet Code Example
Directory Listing with Messages

```
public class ListingAgletMaster1 extends
    Aglet {
  public void run() {
    try {
      URL destination = new
            URL("atp://killi.genmagic.com");
     Object[] args = new Object[] { new
    File("C:"),
    getAgletContext().getAgletProxy(getAglet
    ID()) };
    AgletProxy proxy =
            getAgletContext().createAglet(getC
    odeBase(),
    "agletbook.ListingAglet1", args);
     proxy = proxy.dispatch(destination);
    } catch (Exception e) {
      print("Failed to create the child.");
      print(e.getMessage());
    }  }     // end run()
```

```
  public boolean
     handleMessage(Message msg) {
    print("is handling a message...");
    if (msg.sameKind("Listing")) {
      String[] list =
    (String[])msg.getArg();
      for (int i = 0; i < list.length; )
        System.out.println(i + ": " +
    list[i++]);
      return true;     // Message handled
    } else
      return false; // Not Handled
  } // end handleMessage
}    //end class
```

23

Aglet Security Threats

Security issues specific to mobile agent systems:

- Protection of host against Aglets.
- Protection from other Aglets.
- Protection of the Aglet from the host - no good solution yet.
- Protection of the underlying network - no good solution yet.

Security issues that are not possible to solve in the current Aglet system:

- Hide anything within an agent without encryption.
- Distinguish an agent from a clone.
- Verify whether a host system /Java interpreter have been altered or will run an agent properly.
- Verify that a host will run an agent as requested.
- Any security holes in Java interpreter will affect Aglets.

24

Aglet Security Model

- **Principals** - authenticated identities used to enforce the security policy: Consists of Aglet and Context, and their associated Manufacturers and Owners, and the Network Domain (group of trusted servers).
- **Permissions** - Sets actions available on resources by principals. Based on JDK 1.2 policy file definition. For example, an aglet may be given access to a specific file.
- **Protections** - Principle's means to protect its resources. For example an aglet may request that only itself (or its owner) be able to dispose of it.
- **Security Policy** - Set of rules defining permissions and protections. Set by policy authority (the principal responsible for a particular resource). For example, the Context authority is responsible for keeping the server safe from malicious agents. It might have the following policy:

```
grant codebase "http://thishost" signed by "IBM" owned by
"Alex" {permission java.io.FilePermission "/tmp/file.dat
"read"; java.net.SocketPermission "www.trl.ibm.co.jp:80"
"connect";
```

25

References

Published Sources

- ***Agents: Not just for Bond anymore*, Bret Sommers, Java World, April, 1997.**
- ***The architecture of aglets*, Bill Venners, Java World, April, 1997.**
- ***A Hands-On Look At Java Mobile Agents*, Kiniry and Zimmerman, IEEE Internet Computing, July-August, 1997.**
- ***Mobile Agents: Are they a good idea?*, Harrison, Chess, and Kershenbaum, IBM Research Report, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY.**
- **Programming and Deploying Java Mobile Agents with Aglets, Lange and Oshima, Addison-Wesley, 1998. - *very good reference on mobile agents and Aglets written by the creators of the Aglet model.***
- ***A Security Model for Aglets*, Karjoth, Lange, and Oshima, IEEE Internet Computing, July-August, 1997.**
- ***Solve real problems with Aglets, a type of mobile agent*, Bill Venners, Java World, May, 1997.**

Websites

- **IBM Aglets website: http://www.trl.ibm.com/aglets - *creators of Aglets.***
- **Aglets Portal website: http://aglets.sourceforge.net/links.html - *development of Aglets being continued by open source community.***

26