Lecture Outline

- What is an Aglet?
- Aglet Under the hood – look at the object model
- Aglet Package
- Aglet Event Model

Introducing the ASDK

- Aglet Software Development Kit.
- Facilitates flexible code distribution, that is factoring functionality and communications.
- Enables programs to be developed that are code in demand.
- Provides an event model that supports mobility-oriented and mobility-triggered operations.
- A set of API that enables all of the above.
- Extensive.

ASDK…cont..

- Aglet API is an agent development kit.
- A set of Java classes and interfaces that allows create mobile Java agents.
- “write once, go anywhere”.
- Once you have written an aglet, it will run on every machine that supports the Aglet API.
- Aglet API mirrors the applet model in Java.

What is an Aglet?

The Oxford dictionary definition:

“A metal tag attached to each end of a shoelace.”

Metal tubes at the end of your shoe lace!!!

What is an Aglet…cont…?

- Light weight agent

Aglet = Agent + Applet
Aglet Defined

- Java based autonomous software agent.
- Mobile agent - supports the ideas of autonomous execution and dynamic routing on its itinerary.
- A Java object that can be transported along with state information.
- Can receive request from external sources, but each individual aglet decides whether or not to comply with external requests.
- Executable in any aglet environment, operating within any Java environment.

Aglet Defined...cont...

- Java programs that can halt execution, travel across the network (with both code and state in tact) and continue execution at another host.
- Hosted by an Aglet Server i.e. Tahiti Server.

Aglet vs Applets?

- Like an applet, the class files for an aglet can migrate across a network.
- Unlike applets, when an aglet migrates it also carries with it its state.
- An aglet is code that can move across a network from a server to a client.
- Aglet is a running Java program (code and state) that can move from one host to another on a network.
- Aglet carries its state wherever it goes, it can travel sequentially to many destinations on the network, including eventually returning back to its original host.

(Venners 1997)

Aglet Object Model

Key abstractions

- Aglet
- Proxy
- Context
- Identifier

Aglet Object Model Explained

- Aglet
- Proxy
  - Shields the aglet from direct access to its public methods.
  - Provides location transparency for the aglet, that is, it can hide the aglet’s real location. This means that an aglet and its proxies can be separated so that a local proxy hides the address of the aglet.

- Context
  - aglet’s workplace.
  - Stationary object provides a means for maintaining and managing running aglets in a uniform execution environment where the host system is secured against malicious aglets. I.e. the Tahiti Server provides context to aglets created in it.

- Identifier
  - a globally unique and immutable throughout the lifetime of the aglet. An id.

Aglet Operations

Fundamental life cycle-related operations:
- Create -
  - within an aglet context; each aglet is assigned a unique identifier.
- Clone -
  - a distinct aglet with a unique identifier is produced.
  - The clone gets a new thread of execution.
- Dispose -
  - Terminates execution and removes the aglet from the context. When no references to the aglet remain, it will be subject to garbage collection.
Aglet Operations…cont.

- **Dispatch** -
  - An aglet is removed from its current context and forwarded to another, typically remote context.
  - At the target site, its execution (its run() method) begins a new.

- **Retract** -
  - An AgletID must be known to be able to perform a retraction.
  - An aglet is removed from its “dispatched-to”, typically remote, context and reintroduced to its “home” context. At home, its execution begins a new.

- **Deactivate**
  - Halts execution. Its state is transferred to secondary storage.
  - The aglet is not removed from the aglet context. The aglet sleeps for at least the specified dormancy period.

- **Activate** - When a deactivated aglet’s sleep cycle expires. It is activated and its execution begins.

Aglet Event Model

- **Aglet programming – event-based**
  - Programmers write listeners into an Aglet
  - Listeners catch particular events in the life-cycle of an Aglet
  - This allows a particular action (which has been coded) to take place when a specified event occurs

- **Three types of Events**
  - **Clone Event** – clone operation
  - **Mobility Event** – dispatch, arrival, retraction operation
  - **Persistency Event** – activate & deactivate operation

Aglet Communication Model

- **Message Passing Model**
- **Message**
  - Object that is exchanged between Aglets
  - Synchronous & Asynchronous Messaging

- **Future Reply**
  - For Asynchronous messaging
  - **Reply Set**
    - Contain multiple future reply objects
    - Get results as and when they become available

Aglet Package

- Aglet
- AgletProxy
- AgletContext
- Message – Next Week
- FutureReply – Next Week
- AgletID

Aglet Class

- **Abstract base class**
  - Used to customise your own Aglets
  - Define methods for controlling its own life cycle
    - E.g. onDisposal, onCreation
  - Respond to events
    - E.g. onCloning
  - Access attributes associated with the Aglet
    - AgletInfo object
      - getAgletInfo()
      - Creation time, codebase, arrival time, address of its current context etc.
Aglet Class

```java
import com.ibm.aglet.*;
public class MyFirstAglet extends Aglet {
    public void onCreation()
    {
        //First method that gets called on creation
        //Typically do the initialization here
    }
    public void run()
    {
        /* Gives the Aglet its thread of execution on creation
           or on arrival in a new context.
           This is the main entry point for the Aglet's thread of
           execution. */
    }
}
```

AgletProxy Interface

- Acts as the handle of an aglet
- Provides a common way of accessing the aglet
- Aglet class
  - several public methods
    - should not be accessed directly from other aglets
    - security reasons
  - For Aglet A to communicate with Aglet B-
    - The proxy of Aglet B must be obtained
    - interact with Aglet B through the interface
- Proxy is a shield that protects Aglets from malicious Aglets
- The AgletProxy object consults the Security Manager
  - Check whether current execution context is allowed to
    perform the method
- AgletProxy Interface also provides Location Transparency
  - If the Aglet is on a remote host, then the proxy forwards a
    request to the Aglet and returns the results

Aglet Proxy…cont...

```
Proxy Proxy

Aglet Aglet

Aglet Context

Aglet Server/Viewer
```

AgletProxy Interface

- Creating an Aglet is a way of getting a Proxy
  - AgletContext.createAglet() returns the proxy of the newly created Aglet
- Other methods
  - AgletContext.retractAglet()
  - AgletProxy.clone()
  - AgletProxy.dispatch()

```
Proxy 1

handleMessage()
{
    ... sendMessage()
    sendReply()
    ...
}
```

```
Proxy 2

Local Context
```

```
Proxy 3

Remote Context
```

AgletProxy Interface

- Proxies of existing Aglets can be obtained as follows:
  - Aglet can get its own proxy:
    - Aglet.getProxy()
  - Retrieve an Enumeration of proxies in a context
    - AgletContext.getAgletProxies()
  - If you know the Aglet ID, you can get the proxy
    - AgletContext.getAgletProxy() via Message Passing
  - An AgletProxy can be sent in a Message object as an argument to aglet locally or remotely
  - Put an AgletProxy into a context by setting the AgletContext.setProperty() method and share the proxy object
- The Aglets framework provides an implementation of the AgletProxy interface, so you don't need to implement this...
  - Unless you want write your own Aglets server etc.
Aglet Context
- Equivalent of a Place
- Aglet’s “home”
- It spends most of its life in a context
- Except when it is traveling or in “transit”, it is always in a context
- When mobile, it moves from one context to another
- An Aglet belongs to only one context at any time
- A uniform execution environment for Aglets – in an otherwise heterogeneous world
- Typically created by a server that has a network daemon which monitors the network for Aglets
- Incoming Aglets are received and inserted into the context by the daemon
- Often a user interface – graphical or command line – is provided to the context

Context – Tahiti
- ASDK comes with a GUI to an implemented Context
  - Tahiti Server
- Tahiti provides a graphical user interface that allows monitoring and controlling of Aglets
- It contains a network Daemon that listens for incoming Aglets
- It has a Security Manager that protects the underlying host
- It instantiates one Context

Tahiti Server
- Creates and loads aglets.
- Dispatches and retracts.
- Process of creating an aglet.
  - Loading the class file.
  - Instantiating the aglet.
  - Establishing the aglet in the context.
  - Invoking onCreation() and run().
AgletContext Interface

- An Aglet uses the AgletContext Interface to:
  - Get information about its environment
  - To send messages to the environment
  - To send messages to other Aglets that are active in the current environment
- Aglet class has a method for getting access to its current Context
  \[
  \text{context} = \text{getAgletContext();}
  \]

AgletContext Interface

- With access to a Context, an Aglet can create new Aglets:
  \[
  \text{context.createAglet(...);}
  \]
- It can retract or pull remote Aglets to its current context:
  \[
  \text{context.retractAglet(remoteContextURL, agletID);}
  \]
- It can also retrieve a list (enumeration) of the proxies of Aglets in the same context
  \[
  \text{proxies} = \text{context.getAgletProxies();}
  \]

AgletID Class

- Every Aglet is assigned a Globally Unique Identity
- Kept throughout its lifetime
- AgletID class – convenient abstraction for this ID
- Hides the implementation specific representation of the aglet identity
- It cannot be changed
- Retrieved directly from the Aglet and its proxy
  \[
  \text{AgletID aid} = \text{proxy.getAgletID();}
  \]
- If you know the ID and the context, you can query the context to retrieve the aglet proxy
  \[
  \text{proxy} = \text{context.getAgletProxy(aid);}
  \]

Aglet Creation

- Gets created in a place (i.e. Context)
- Creation can be initiated either by another agent residing in the same place or by another agent or non agent system outside the place.
- Instantiation and Identifier assignment
- Initialization
- Autonomous execution
- Creation & Initialization Methods
  \[
  \text{public void onCreation(Object init)} \\
  \text{public void run()}
  \]
  - called every time the aglet arrives at or is activated in a new context.

Aglet Creation

- For an Aglet to Create other Aglets, it needs to get access to the current Context
  \[
  \text{public final AgletContext Aglet.getAgletContext();}
  \]
- For an Aglet to load another Aglet (SomeAglet) into the current Context,
  \[
  \text{getAgletContext().createAglet(getCodeBase(), "SomeAglet", null);}
  \]
Creating an Aglet

Public abstract AgletProxy
AgletContext.createAglet(URL codeBase, String code, Object init)

- Creates an instance of the specified aglet class. The aglet's class file can be located on the local file system as well as on a remote server.
- If codebase is null, the context will search for the code in the local system, aglet search path (AGLET_PATH).

Parameters Explained

Codebase
- Specifies base URL of the aglet class file.
- The aglet search path works in a similar way to Java's class path.
- It is typically an environment variable that specifies a list of directories to be searched for aglet class files.

Code
- Gives the name of the class of the aglet being instantiated.

Init
- is an object passed on the aglet's onCreation method.

Creating an Aglet…cont

When an aglet has been successfully created, it is inserted into the current context and started:
- invocation of onCreation()
- followed by run().
- The method returns a handle(Aglet Proxy) for the new aglet as soon as the aglet's constructor method finishes.

Creating an Aglet…cont

Aglet Constructor

protected Aglet.Aglet()

- Creates an Uninitialised Aglet

Method is called only once in the life cycle of an Aglet
- As a rule, avoid overriding the constructor
- Preferred usage is the onCreation() method for Initialisation

Creating an Aglet…cont

Public void Aglet.onCreation(Object init)
- Initialises the new Aglet
- Method is called only once in the life-cycle of an Aglet
- Override this method for customising the initialisation of the Aglet

Creating an Aglet…cont

public void Aglet.run()
- Entry point for the Aglet's own execution thread
- Method is invoked upon successful creation, dispatch, cloning, retraction or activation of the Aglet
Order of Execution for Creation

1. `createAglet()`
2. `Aglet()` – preferable not to override
3. `onCreation()`
4. `run()`

Understanding Codebase

- Important role as a reference point for the aglet class loader.
- The aglet uses the code base to search for the class file of the indicated class.
- Eg. Codebase - http://somehost/path, the class loader will look for the compiled code of some class on the Web Server on somehost with the path path/SomeClass.class.
- If the code of SomeClass is located on the local machine, codebase will be something like. File://c:/somepath
- Codebase is null -> local host aglet search path (AGLET_PATH) is used-works very similar to Java’s class path.

Codebase and Class Transfer

- Rely on classes from codebase as well as the CLASSPATH.
- Only classes from codebase are transferred along with the aglet when it moves in the network.
- Not all classes from the code base is transferred at the same time.
- When an aglet is dispatched to a remote location, only classes in use are transferred.
- An aglet arrives at the remote location, it may require additional classes to continue execution. A network connection is needed for the remote aglet to fetch needed classes from the codebase.
- JAR (Java archive) is a file that contains the class and resources, such as image and sound file for a Java program, all gather into a single file and possibly compressed for faster downloading from a remote host. All classes in the jar file are transferred at one time along with the aglet.
- If the aglet requires additional classes that are not included in the JAR file, the codebase specification is used to fetch the required classes.

Creation Example

Aglet Disposal

- When an Aglet is in a Context, it takes up various resources.
- Therefore, you need to properly dispose of it, once its task has been completed.
- `public final void Aglet.dispose();`
- `onDisposing()` – called when an attempt is made to dispose of an Aglet.
- You can subclass and customise this method to do any cleaning up.

Order of Execution for Disposal

1. `dispose()`
2. `onDisposing()`
Example Code – Proxy Retrieval

- Focus in methods that retrieve aglet proxies.
- play an important role as convenient handles for aglets.
- use to gain access to and communicate with its corresponding (possible remotely located) aglet.
- there are number of ways in which you can retrieve proxies from local as well as remotely located aglets.

Example Code – Proxy Retrieval

Public abstract Enumeration
AgletContext.getAgletProxies()

- Get the proxies of all aglets in the current context.
- Returns an Enumeration of proxies. Which may include currently deactivated aglets.
- Enumeration has two methods that support iteration: hasMoreElements() and nextElement().

Example Code – Proxy Retrieval

Method 1:
public abstract AgletProxy
AgletContext.getAgletProxy(AgletID identity)

- Retrieves the proxy of an aglet in the current context. The identity of the aglet must be known.

Example 1 – Getting a Local Proxy

Public class RetrievalChild1 extends Aglet
{
    public boolean handleMessage(Message msg)
    {
        return true; // message handles
    }
}

Example 1 – Getting a Local Proxy

Public class RetrievalChild2 extends Aglet
{
    private AgletID aid;
    public void onCreation(Object init)
    {
        aid = (AgletID)init;
    }
    public void run()
    {
        // Retrieves the proxy of my brother
        AgletProxy proxy =
        getAgletContext().getAgletProxy(aid);
        proxy.sendMessage(new Message("Hello brother"));
    }
}

Example 2– Getting a Remote Proxy

public abstract AgletProxy
AgletContext.getAgletProxy(URL contextAddress, AgletID identity)

- Retrieve the proxy of an aglet hosted in a remote context. The remote context is identified by its URL and the identity indicates the aglet. Both must be known.

(the following example creates two aglets. One of these aglets is dispatched to a remote context.)
Example 2 – Getting a Remote Proxy

AgletProxy proxy = getAgletContext().createAglet(getCodeBase(), "RetrievalChild1", null);
AgletID aid = proxy.getAgletID();
Proxy.dispatch(destination);
Objects args[] = new Object[]{destination, aid};
getAgletContext().createAglet(getCodeBase(), "RemRetrievalChild2", args);

Public class RemRetrievalChild2 extends Aglet
{
    private URL url;
    private AgletID aid;
    public void onCreation(Object init)
    {
        Object args[] = (Object[])init;
        url = (URL)args[0];
        aid = (AgletID)args[1];
    }
}

Public void run()
{
    AgletProxy proxy = getAgletContext().getAgletProxy(url, aid);
    //Sends him a message
    try
    {
        proxy.sendMessage(new Message("Hello brother");
    }catch(Exception e)
    {
    }
}

Aglet Event Model

- Aglet programming – event-based
  - Programmers write listeners into an Aglet
  - Listeners catch particular events in the life-cycle of an Aglet
  - This allows a particular action (which has been coded) to take place when a specified event occurs
- Three types of Events
  - Clone Event – clone operation
  - Mobility Event – dispatch, arrival, retraction operation
  - Persistency Event – activate & deactivate operation

Aglet Event Model

- Three Listeners
  - Clone Listener
  - Mobility Listener
  - Persistency Listener
- When an Aglet is Cloned, Moved, Saved etc. a number of events are sent to the Aglet.
- The listener is an object that implements a specific listener Interface.
- These Interfaces define one or more methods that must be called in response to the given event

Aglet Event Model

- Clone Listener
  - Listens for Cloning Events
  -Customise the listener to take specific action:
    - When an Aglet is about to be cloned
      - onCloning
    - When the clone is actually created
      - onClone
    - Just after the cloning is done
      - onCloned
Mobility

- Dispatching
  - Push it to a remote location
- Retraction
  - Pull it from a remote location

Aglet Mobility

- State
- Byte Code
- Host
- Sending
- Receiving
- State
- Byte Code
- Host

Mobility - Dispatching

- public final void Aglet.dispatch(URL destination)
  e.g.: dispatch (new URL("http://some.host.com:port"));
- Add Listeners
  - public final void Aglet.addMobilityListener(MobilityListener listener)
- Remove Listeners
  - public final void Aglet.removeMobilityListener(MobilityListener listener)

Order of Execution for Dispatch

- Origin
  1. run()
  2. dispatch()
  3. onDispatching()
- Destination
  4. onArrival()
  5. run()

Mobility Example

```java
package examples.cpe5010;
import java.io.*;
import java.lang.*;
import java.net.*;
import java.util.*;
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

public class HelloWorldAglet extends Aglet {
    private Vector curItinVect;
    private URL destination;
    String strDest;
```
Mobility Example

public void onCreation(Object init)
{
    System.out.println("Hi! I have been created...");
    /* INSERT YOUR SERVER & PORT NO: e.g:
    atp://electra.csse.monash.edu.au:9000 */
    strDest = new String("atp://-------------");
    try {
        destination = new URL(strDest);
    }
    catch (MalformedURLException malURL1){
        System.out.println("mal URL...");
    }
}
public void run()
{
    try {
        dispatch(destination);
    }
    catch (Exception e){
        System.out.println(e.getMessage());
    }
    System.out.println("I am finished....!");
}

Mobility Example

addMobilityListener(
    new MobilityAdapter() {
    public void onArrival(MobilityEvent e) {
        System.out.println("I have arrived at " + strDest);
        dispose();
    }
    });
public void run()
{
    try {
        dispatch(destination);
    }
    catch (Exception e){
        System.out.println(e.getMessage());
    }
    System.out.println("I am finished....!");
}

Mobility - Retracting

- AgletContext.retractAglet(URL contextAddress, AgletID aid)
- public void MobilityAdapter.onArrival(MobilityEvent m)
  - Called when an Aglet arrives in a new location
- public void MobilityAdapter.onReverting(MobilityEvent e)
  - Called when someone from a remote location attempts to retract
    the Aglet. Subclasses can override this method to implement
    actions that should be taken when this is requested.

Order of Execution for Retraction

- Local
  1. retractAglet()
  2. onArrival()
  3. run()
- Remote
  4. run()
  5. onReverting()

Retraction Example

- RetractionExample Aglet is retracting the RetractionChild Aglet
  that it had previously dispatched to some remote location
public class RetractionExample extends Aglet {
    public void run()
    {
        try {
            AgletProxy proxy = getAgletContext().createAglet(
                null, "RetractionChild", null);
            String strDest = new String("atp://some.host.com");
            AgletID aid = proxy.getAgletID();
            destination = new URL(strDest);
            proxy.dispatch(destination);
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}

Retraction Example

public class RetractionChild extends Aglet {
    public void onCreation(Object init)
    {
        addMobilityListener(
            new MobilityAdapter() {
            public void onArrival(MobilityEvent m) {
                System.out.println("I am being called back!");
            }
            });
        public void onReverting (MobilityEvent e) {
            System.out.println("I have come back!");
        }
        public void run()
        {
            System.out.println("Hello!");
        }
    }
}
Agent Transfer Protocol (ATP)

- Simple application-level protocol modelled on HTTP.
- ATP is designed to transmit an agent independently on the agent system. ATP defines the following.
- Standard request methods:
  - Dispatch - requests a destination agent system to reconstruct an agent from the content of a request and to start executing the agent. If the request is successful, the sender must terminate the agent and release any resources consumed by it.
  - Retract - requests that a destination agent system send the specified agent back to the sender. The receiver is responsible for reconstructing and resuming the agent. If the agent is successfully transferred, the receiver must terminate the agent and release any resources consumed by it.
  - Fetch is similar to the GET method in HTTP, it requests that a receiver retrieve and send any identified information.

Cloning

- public final object Aglet.clone()
  - Create an identical copy of it in the current context
  - The method returns the proxy of the clone Aglet
- Add Listeners
  - public final void
  Aglet.addCloneListener(CloneListener listener)
- Remove Listeners
  - public final void
  Aglet.removeCloneListener(CloneListener listener)

Order of Execution for Cloning

**Original**
1. run()
2. onCloning()
3. onCloned()

**Clone**
4. onClone()
5. run()
Cloning Example

```java
public void onCloned(CloneEvent e) {
    System.out.println("OnCloned Event!");
}
```

```java
def run() {
    try {
        clone();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Persistency

- `public final void deactivate(long duration)`
  Deactivates the aglet. It will resume execution in the current context after the duration.

- Add Listeners
  - `public final void Aglet.addPersistencyListener(PersistenceListener listener)`

- Remove Listeners
  - `public final void Aglet.removePersistencyListener(PersistenceListener listener)`

Order of Execution for Persistence

**Before**
1. `run()`
2. `deactivate()`
3. `onDeactivating()`

**After**
4. `onActivation()`
5. `run()`

Event

- `public PersistencyAdapter PersistencyAdapter()`
  - Creates an instance of the clone adapter

- `public void PersistencyAdapter.onDeactivating(PersistencyEvent e)`
  - Called when an attempt is made to Deactivate an Aglet

- `public void PersistencyAdapter.onActivation(PersistencyEvent e)`
  - Called when an attempt is made to activate an Aglet

Mobility Event

- `public MobilityEvent MobilityEvent()`
  - Has a method `getAgletProxy()` that gives the proxy of the Aglet that is being cloned

- `public URL MobilityEvent.getLocation()`

Example

```java
public void onDispatching(MobilityEvent e) {
    System.out.println(e.getLocation().toString());
}
```
Event

- MobilityEvent
  - public AgletProxy MobilityEvent.getAgletProxy()  
  - public URL MobilityEvent.getLocation()

Example

```java
public void onDispatching(MobilityEvent e) {
    System.out.println(e.getLocation().toString());
}
```

Event

- PersistencyEvent
  - public AgletProxy PersistencyEvent.getAgletProxy()  
  - public long PersistencyEvent.getDuration()

Example

```java
public void onDeactivating(PersistencyEvent e) {
    System.out.println(e.getDuration());
}
```

References


http://www.vistabonita.com/papers/DCAglets/  
http://www.geocities.com/SiliconValley/Lakes/8849/simple.htm  
http://www.geocities.com/SiliconValley/Lakes/8849/roaming.htm