Software Technologies

Mobile Code

Sebastian Fischmeister
fischmeister@softwareresearch.net
University of Salzburg

Aglets

- Originates from a combination of two terms: “Applet” and “Agent”
  - Applet known from Java Applets
  - Agent

- Do not mix up these two:
  - Applets belong to code-on-demand paradigm
    - code is mobile
    - stack is static
    - data is static
  - Aglets belong to the mobile-agent paradigm
Programming Language

- Aglets are implemented in Java
  - easy to implement MA systems
    - dynamic class loading
    - multi-threaded programming
    - serialization
    - reflection
    - platform independence
  - in 1996 Java was „the“ thing

Aglet Environment

- IBM Aglets Workbench
  - Tahiti as management application
    - Tahiti is an application program that runs as an agent server. You can run multiple servers (Tahiti) on a single computer by assigning them different port numbers. Tahiti provides a user interface for monitoring, creating, dispatching, and disposing of agents and for setting the agent access privileges for the agent server. (Tahiti user’s guide)
    - Aglet API
    - example Aglets
- Resources
  - http://aglets.sourceforge.net/
Aglet API

- Four Basic Elements
  - **Aglet**: eq. to mobile agent
  - **Proxy**: eq. to stub in client/server
  - **Context**: eq. to places
  - **Identifier**
- Aglet
  - the mobile agent
  - reactive (responds to messages)
  - proactive (runs within own thread of execution)
  - autonomous (can move on its own volition)
- Proxy
  - protects the agent from direct access
  - forwards messages to a remote Aglet

Aglet API

- Context
  - equivalent to a place in the OMG MASIF
  - one Tahiti basically runs one context
- Identifier
  - unique identifier for each aglet

- Missing (compared to Grasshopper):
  - region
  - region registry
Relationship: Aglet and Proxy

- Proxy represents the agent
- Protects the agent from direct access (attacks)
- Allows location transparent access

![Diagram showing relationship between Aglet and Proxy](image_url)

Operations

- Tahiti and Aglets can perform six basic operations on another Aglet.

- Six basic operations
  - creation
  - cloning
  - dispatching
  - retraction
  - activation and Deactivation
  - disposal
Operations

- Creation
  - initializes a new aglet
  - creates a new thread
  - assigns an identifier
- Cloning
  - produce an identical copy of an Aglet
  - assigns a different identifier (unique)
- Dispatching
  - Aglet is transported between two Tahiti contexts
  - usually between different locations

Basic Operations

- Retraction
  - calls back an Aglet
  - moves the Aglet from foreign context to the local context
- Deactivation and Activation
  - stops and resumes an Aglet
  - deactivate must occur before activate
  - useful before restarting Tahiti
  - differences to Grasshopper
    - no wake-up events
Basic Operations

- Disposal
  - stops execution of an Aglet
  - frees all used resources (e.g., thread group) - garbage collection
  - removes an Aglet from its current context

Agent Life-Cycle Model
Aglet Methods

- Aglet programming model
  - uses Java for implementing the Aglets
  - uses observer pattern
  - callback model
- Available listeners
  - clone listener
  - mobility listener
  - persistence listener

Aglet Methods

- Listener specifics
  - similar to Java events
  - methods are invoked after the event occurs
  - methods and events have similar names
  - allow customized behavior (e.g., vetos)
Event Model Listeners

- Clone listener
  - listens for cloning events
  - actions can be customized to occur before or after cloning
    - onCloning()
    - afterCloning()

- Mobility listener
  - listens for mobility events
  - can customize actions before and after moving actions
    - onDispatch()
    - onReverting()
    - onArrival()
Event Model Listeners

- Persistence listener
  - listens for persistence events
  - actions can be customized to occur when an Aglet is about to be deactivated or has been activated
    - onDeactivating()
    - onActivation()

Aglet API

- Aglet API
  - simple and flexible
  - represents lightweight pragmatic approach to mobile agents (compare it to the Grasshopper API)
  - not as lightweight as other approaches
- Java classes
  - Aglet
  - Message
  - Futurereply
  - Agletid
  - Agletproxy
Aglet API

- Java interfaces
  - AgletProxy
  - AgletContext
- Aglet class
  - contains all methods needed to perform the basic aglet operations
    - moving, messaging, ...
  - contains all the elements of the aglet
    - Aglet id

Aglet API

- Aglet creation
  - create a customized aglet
  - extend the class com.ibm.aglet.Aglet

```java
import com.ibm.aglet.*;
public class MyFirstAglet extends Aglet{
    // Put aglet's methods here
}
```

- Similar to Grasshopper:
  - specific methods for specific actions
  - to not use the standard OO approaches!!
    (e.g., new operator)
Operation Examples

- Dispatch operation
  - dispatches an aglet to a remote context
  - uses URLs (Grasshopper uses separate class)
    
    ```java
    dispatch(new URL("atp://remote.host.com/context"));
    ```

- Dispose operation
  - removes the aglet from the current context
    
    ```java
    dispose();
    ```

- See the API for similar ones
- We will try more in the lab…

Messaging Example

- Message class
  - communication is performed by exchanging dedicated objects (message objects)
  - AgletProxy class is responsible for actually sending and receiving messages

- Message creation
  - define the field ‘Type’
  - second field of is optional (can contain additional information)
Messaging Example

- **Code**
  ```java
  Message myName = new Message("my name", "Lois");
  or
  Message yourName = new Message("Steve");
  ```

- The `handleMessage` method receives all messages (return true ! -&gt; chain of responsibility pattern)
  ```java
  public boolean handleMessage(Message msg){
      if(msg.sameKind("hello")){
          doHello(); //respond to 'hello' message
          return true; //yes I handled message
      }
      else
          return false; //not handled message
  }
  ```

AgletProxy and FutureReply

- Message objects are sent using the AgletProxy class methods
  - `Object sendMessage(Message msg)`
  - `FutureReply sendFutureMessage(Message msg)`
  - `void sendOnewayMessage(Message msg)`

- **Code example**
  ```java
  proxy.sendMessage(myName);
  String name =
      (String)proxy.sendMessage(yourName);
  ```

- **FutureReply Class**
  - used for asynchronous messaging
  - the Aglet can continue execution while waiting for the reply
FutureReply Example

- FutureReply objects are retrieved using the AgletProxy class method
  - `sendFutureMessage(msg)`
- Code example
  - the sender can continue executing periodic tasks while waiting for a reply

```java
FutureReply future = proxy.sendFutureMessage(msg);
while (!future.isAvailable()){
    doPeriodicWork();
}
Object reply = future.getReply();
```

AgletID Class

- AgletID Class
  - represents the identifier of the Aglet
  - the identifier is unique to each Aglet
  - the identifier object hides the implementation specific representation of the Aglet identity
- Code example
  - identifier can be retrieved from the Aglet and its proxy
    - `AgletID aid = proxy.getAgletID();`
  - query the context to retrieve Aglet with identity `aid` (aid and Context must be known)
    - `proxy = context.getAgletProxy(aid);`
### Aglet API: Interfaces

- **AgletProxy interface**
  - the handle of the Aglet
  - abstraction of the real implementation (separation of concerns & frameworks)

- **Benefits**
  - used by other Aglets for communication
  - provides protection mechanisms (e.g., cannot directly invoke dispose)
  - can provide a remote location for the Aglet
  - possibility for standardization

### AgletProxy

- **Retrieval and setting methods of proxies**
  - Aglet can get its own proxy object
    
    ```java
    Aglet.getProxy();
    ```
  
  - retrieve an enumeration of all proxies within the current context
    
    ```java
    AgletContext.getAgletProxies();
    ```
  
  - get an Aglet proxy for a given identifier
    
    ```java
    aid.getAgletProxy();
    ```
  
  - place AgletProxy object into context property (useful for sharing resources, provide services)
    
    ```java
    AgletContext.setProperty();
    ```
AgletContext

- Aglet Context
  - execution environment for Aglets
  - equivalent to the place
  - hosts the Aglets
  - Aglets only exist inside a context
- AgletContext interface
  - abstract interface of the Aglet context
  - get information about environment
  - send messages to environment
- AgletContext interface methods
  - Aglet class can gain access to current context
    ```java
    Context = GetAgletContext();
    ```
  - Aglet can create new Aglets
    ```java
    Context.CreateAglet(...);
    ```

---

<table>
<thead>
<tr>
<th>AgletContext interface methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>retract (pull) remotely located Aglets into current context</td>
</tr>
</tbody>
</table>
| ```java
  Context.RetractAglet(remotecontexturl, Agletid);
  ``` |
| retrieve a list of proxies of its fellow Aglets in the same context |
| ```java
  Proxies = Context.Getagletproxies();
  ``` |
Example: Remote File Update

- **Aglet Example: Remote File Update**
  - premise: large multiple remote files that must be updated by word replacement
  - one solution: move files to central server, perform update, and move files back
  - another solution: an Aglet that updates files by replacing all occurrences of one specified word in the files with another specified word
  - distributes the load of updates to multiple servers
  - we’re moving “Code” rather than files

---

Example

- **Host**
  - **Aglet**

- **Host**
  - **Aglet**

- **Host**
  - **Aglet**

- **Host (Updating)**
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.net.*;
import java.io.*;
public class UpdateFile extends Aglet{
    URL destination = null;
    File dir = null;
    String from = null;
    String to = null;
    public void onCreation(Object args){
        destination = (URL)((Object[])args)[0];
        dir = (File)((Object[])args)[1];
        from = (String)((Object[])args)[2];
        to = (String)((Object[])args)[3];
        addMobilityListener(){
            new MobilityAdapter(){
                public void onArrival(MobilityEvent e){
                    replace(args.file,args.from,args.to);
                    dispose(); }
                }
            }
        }
    } try{
        dispatch(args.destination);
    }catch (Exception e){
        System.out.println("Failed to dispatch.");
    }
    void replace(File file, String from, String to){
        //Open 'file' and replace 'from' with 'to'
    }
References

[3]: http://www.msci.memphis.edu/~franklin/AgentProg.html
[6]: http://luckyspc.lboro.ac.uk/Docs/Papers/Mesela97.html
[8]: http://www.trl.ibm.co.jp/aglets/whitepaper.htm
[10]: Kimble Cheron, Professor Steven A. Demurjian, and Mitch Saba course on Software Agents and Aglets as basis of this slides