Università degli studi della Calabria **anno accademico 2003/2004**

LAUREA SPECIALISTICA
IN
INGEGNERIA INFORMATICA

AGENTI MOBILI IN JAVA
GLI AGLETS

PROFESSORE
Domenico TALIA

STUDENTE
Donato SAVINO

GESTIONE DELLE RISORSE

Architettura centralizzata → Amministrazione gestita da PC o Workstation

- Aumento degli utenti
- Espansione della rete
- Richiesta di segnale di banda

→ Architettura distribuita

Per risparmiare banda sarebbe necessario utilizzare un'entità che effettui le normali operazioni di ricerca, visitando i vari siti in modo autonomo. → AGENTE MOBILE

DEFINIZIONE DI AGENTE MOBILE

Che cosa è un agente mobile?

- Un agente è un programma che simula le relazioni umane, facendo qualche cosa che altrimenti un'altra persona dovrebbe fare per te.

- Gli agenti sono programmi interoperabili, altamente compatibili che vengono eseguiti in ambienti poco sensibili ai guasti, orientati agli oggetti e con memoria sicura, in grado di assicurare un elevata velocità nell'ambito di un'infrastruttura di rete.

DEFINIZIONE FORMALE

Agente mobile: entità software autocontenuta di stato e di comportamento, in grado di migrare in rete ed interagire con le risorse nei nodi visitati, scoprendo i servizi offerti.

CAMPI APPLICATIVI 1

- Motori di Ricerca distribuiti

  Utilizzare agenti mobili per accogliere informazioni che verranno scaricate in un database centralizzato.
- Amministrazione remota di sistemi distribuiti

  Agenti mobili potrebbero circolare in una intranet e segnare eventuali anomalie delle risorse (stampanti senza carta, interconnessioni non funzionanti).
- Messaggi Interattivi

  Spedire messaggi di posta elettronica che interagiscono direttamente con l'utente, implementare ***wizard*** per guidare l'utente nello svolgimento di un'operazione.
- Applicazioni client/server

  Utilizzare client autonomi.

CAMPI APPLICATIVI 2

- Piazzista Virtuale

  I rappresentanti di prodotti potrebbero utilizzare un agente per contattare i loro fornitori, estendendo il campo d'azione a tutto il mondo oppure fornendo un servizio disponibile 24 ore su 24 per l'acquisto di articoli.
- Debugging di applicazioni

  Applicazioni in versione beta potrebbero essere distribuite sotto forma di agenti mobili, in modo che quando si verifica un errore l'applicazione venga clonata e inviata al programmatore che potrà provvedere al debugging.
- Ubiquitous computing

  Le applicazioni installate da un utente potranno seguirlo nel caso la sua ***login*** venga trasferita su un'altra macchina.

SICUREZZA

- SecurityManager
  Controllare e limitare gli accessi al file system locale.

- Digital signatures
  Autenticare la provenienza di un oggetto.

AGLETS

E’ una tecnologia ad agenti ideata e creata dalla IBM of Japan, completamente scritta in Java.

Essendo nata prima del rilascio del JDK 1.1, non ha a disposizione le strutture delle recenti versioni di JDK.

Però già dalla versione 1.1 possiamo trovare i meccanismi di:

- REFLECTION
- SERIALIZATION

REFLECTION E INTROSPECTION

Quando inviamo sulla rete una agente, questo deve avere a disposizione le classi che utilizza nel suo thread d'esecuzione. Per evitare errori di questo tipo di utilizza il meccanismo di Introspection per analizzare la struttura del codice a run time.

In Java questo meccanismo viene implementato dalle Reflection API che basandosi sul ***Dynamic Linking*** permette di estendere le classi:

- senza modifiche di codice
- anche durante l'esecuzione delle applicazioni

SERIALIZZAZIONE

- Quando un aglet viene inviato, clonato o disattivato, il suo stato viene salvato con il meccanismo di serializzazione di Java. Quindi se nel codice vengono inseriti tipi che non supportano questo meccanismo, questi devono essere dichiarati transient, altrimenti verrà lanciata l'eccezione java.io.NotSerializableException.

- Il meccanismo di serializzazione non è attualmente capace di salvare lo stack d'esecuzione dell'oggetto, infatti prima di eseguire funzioni che implichino tale operazione, nel pacchetto di bytes verrà indicato il punto di entrata del nuovo thread.

ARCHITETTURA

- Aglet API
  Insieme di comandi che permettono di utilizzare l'aglet

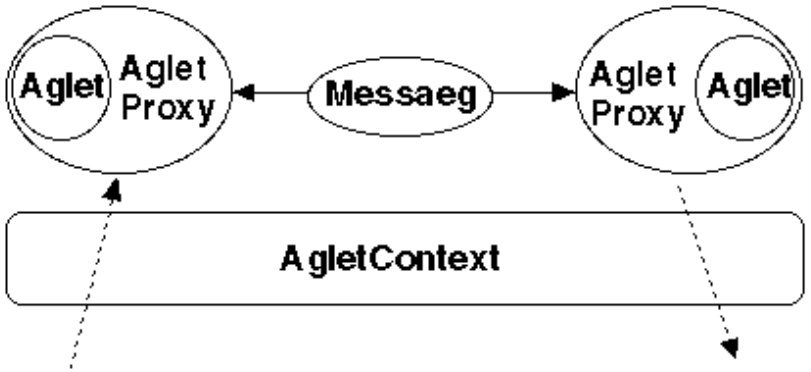
- Aglets Runtime Layer
  Implementazione delle API

- Agent Transport and Communication Interface
  Effettua la traduzione dei comandi con il livello di trasporto

- Transport Layer
  Provvede alla migrazione degli oggetti nella rete

COMPONENTI DELL'AMBIENTE AGLET

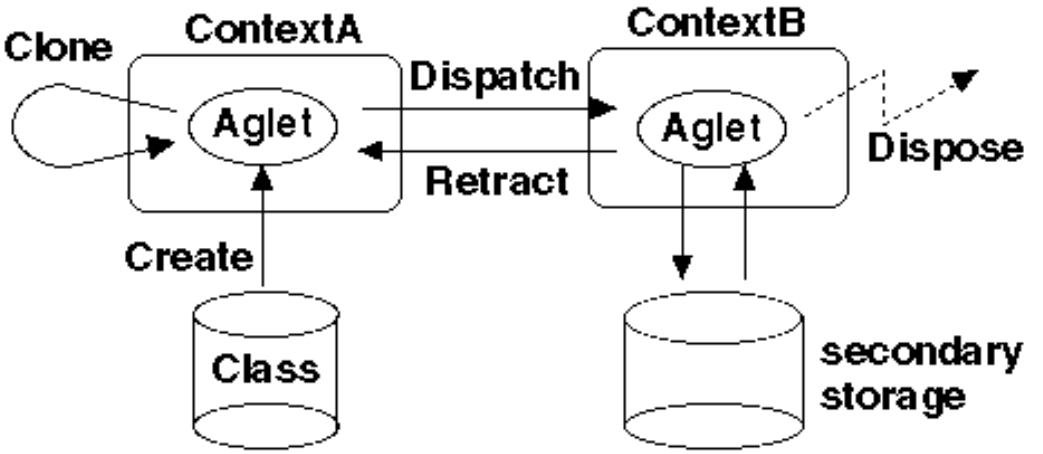
AGENTE
CONTESTO
PROXY
MESSAGE



Ogni Aglet viene associato ad un oggetto AgletContext(sito locale). Gli agenti vengono manovrati da oggetti locali di tipo AgletProxy che provvedono a far recapitare le informazioni raccolte al contesto da cui è partita la ricerca. La comunicazione avviene secondo il meccanismo dell message-passing.

CICLO DI VITA DI UN AGLET

- CREATE
- CLONE
- DISPATCH
- RETRACT
- DEACTIVATE
- ATTIVATE
- DISPOSE



FASI 1

- CREATE

  L'aglet viene creato all'interno del suo contesto e gli viene associato un identificativo unico.

- CLONE

  Viene creata una copia dell'aglet con un differente identificativo. Lo stato di esecuzione non può essere clonato, quindi il suo thread d'esecuzione parte dall'inizio.

- DISPOSE

  Questo processo termina l'esecuzione e rimuove l'aglet dal contesto. Se non rimangono riferimenti, l'oggetto viene passato al garbage collection.

- DISPATCH

  L' aglet viene rimosso dal suo contesto corrente e spedito ad un altro, tipicamente remoto. A destinazione, la sua esecuzione comincia di nuovo.

FASI 2

- RETRACT

  L’ aglet viene riconsegnato al suo contesto di “casa“, dove la sua esecuzione comincia di nuovo.

- DEACTIVATE

  L'esecuzione viene arrestata; l‘oggetto viene disattivato dopo aver suo salvato stato. L'aglet dorme per un periodo di sleep specificato, senza essere rimosso dal suo contesto .

- ACTIVATE

  Quando scade il tempo di sleep indicato al momento della disattivazione, l’aglet riprende la sua esecuzione.

MIGRAZIONE DEGLI AGENTI

1) Sospensione dei threads creati dall'aglet.

2) Serializzazione dello stato dell'aglet usando il meccanismo ***Java Serialization.***

3) Serializzazione del bytecode della classe.

4) Trasferimento dello stream di dati a destinazione e creazione di una copia dell'aglet.

5) Inizializzazione dello stato dell'aglet.

6) Assegnazione di un nuovo thread d'esecuzione.

AGLET DI BASE

```
import com.ibm.aglet.*;

public class Greeting extends Aglet {
    public void onCreation(Object init) {
        System.out.println("Greetings.");
    }
}
```

Class com.ibm.aglet.Aglet 1

- onCreation(Object) // crea l'oggetto
- run() // dà inizio al thread d'esecuzione dell'aglet
- clone() // restituisce una copia del proxy associato all'aglet
- deactivate(long) // disattiva l'aglet per il tempo indicato
- dispatch(URL) // invia l'aglet verso la destinazione indicata
- dispose() // rimuove l'aglet dal suo contesto
- getAgletContext() // restituisce il corrente contesto d'esecuzione
- getProxy() // restituisce l'oggetto proxy che controlla l'aglet
- getText() // restituisce il testo dell'aglet
- setText() // setta il testo dell'aglet
- getAgletInfo() // restituisce un oggetto AgletInfo contenente dati come la // data di creazione, l'indirizzo del // suo contesto corrente…

Interface com.ibm.aglet.AgletContext

- createAglet(URL, String, Object)
  crea un nuovo aglet nel contesto corrente
- getHostingURL()
  restiutisce l'URL del processo che gira sul contesto corrente
- getAgletProxies()
  restituisce un enumeration di AgletProxy
- getName()
  restituisce il nome del contesto
- getProperty(String)
  restituisce la proprietà corrispondente al nome indicato
- retractAglet(URL, AgletID)
  richiama un aglet specificando la sua URL ed il suo identificativo
- setProperty(String, Object)
  memorizza una variabile nel contesto, specificando un nome simbolico
- shutdown(Message)
  termina il contesto
- start()
  avvia il contesto

Coordinator

```
import com.ibm.aglet.*;
import java.net.*;

public class Coordinator extends Aglet {
    String remoteContext = "atp://localhost:5000/";
    public void onCreation(Object init) {
            try {
              URL destination = new URL(remoteContext);
              AgletContext ac = getAgletContext();
              AgletProxy proxy = ac.createAglet(null, "NamedIntroduction",
                              (Object) "Daffy Duck");
              proxy = proxy.dispatch(destination);
              Thread.sleep(5000);
              proxy = ac.retractAglet(destination, proxy.getAgletID());
              dispose();
            } catch (Exception e) { System.out.println("Exception handling
                                              NamedIntroduction: " + e);    }
    }
}
```

NamedIntroduction

```
public class NamedIntroduction extends Aglet {
    private String name = "no-name";
    public void onCreation(Object arg) {
        if (arg != null) {
            name = (String) arg;
        }
    public void run() {
        System.out.println("Hello, my name is " + name + ".");
    }
}
```

SetRemoteProperty 1

```
import com.ibm.aglet.*;
import java.net.*;

public class SetRemoteProperty extends Aglet {
    boolean atHome = true;  URL home = null;
    String remotePropertyKey = "MyProperty";
    String remoteProperty = "SetRemoteProperty was here!";
    public void onCreation(Object init) {
        if (atHome) {
            atHome = false;
            getAgletContext().
            setProperty(remotePropertyKey, remoteProperty);
            try { dispatch(home); } catch (Exception x){ dispose(); }
        }
```

SetRemoteProperty 2

```
                else { atHome = true; }
            home = getAgletContext().getHostingURL();
        }
    }
```

GetRemoteProperty 1

```
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;
import java.net.*;
public class GetRemoteProperty extends Aglet {
    boolean atHome = true;  URL home = null;
    String remotePropertyKey = "MyProperty";
    String remoteProperty = "no property found";
    public void onCreation(Object init) {
        if (atHome) {
            atHome = false;
            remoteProperty = (String) getAgletContext().
            getProperty(remotePropertyKey);
            try { dispatch(home); } catch (Exception x) { dispose(); }
        }
```

GetRemoteProperty 2

```
                    else{
                    atHome = true;
                    System.out.println("Remote property value: " +
                                                    remoteProperty);
                    }
            }
    home = getAgletContext().getHostingURL();
    }
}
```

Class com.ibm.aglet.Aglet 2

- handleMessage(Message)
  gestisce la comunicazione con gli altri aglets
- notifyAllMessages()
  effettua la notifica a tutti i threads in wait
- notifyMessage()
  effettua la notifica ad un singolo thread
- waitMessage()
  mette in stato di wait il thread finchè non viene effettuata una notify
- waitMessage(long)
  mette in stato di wait il thread finchè non viene effettuata una notify o scade il timeout indicato come parametro

Class com.ibm.aglet.Message

- equals(Object)
  valuta se il messaggio che arriva all'aglet è uguale a quello passato come parametro
- getKind()
  restituisce il tipo di messaggio
- getMessageType()
  restituisce la modalità di invio del messaggio
- getTimeStamp()
  restituisce il time stamp relativo all'invio del messaggio
- sameKind(Message)
  controlla se il messaggio ricevuto è uguale a quello passato come parametro
- sendReply()
  invia una reply senza indicare il valore
- sendReply("tipo")
  restituisce come reply un tipo indicato come parametro
- toString()
  restituisce una rappresentazione in stringa del messaggio

StackAglet 1

```
public StackAglet extends Aglets {
    static int capacity = 10; Object stack[] = new Object[capacity]; int num = 0;
    public boolean handleMessage(Message msg) {
        if (msg.sameKind("push")) {
                push(msg);
        } else if (msg.sameKind("pop")) {
                pop(msg);
        } else if (msg.sameKind("isFull")) {
                msg.sendReply( num == capacity);
        } else if (msg.sameKind("isEmpty")) {
                msg.sendReply( num == 0 );
        } else return false;
        return true;
    }
```

StackAglet 2

```
    private void push(Message msg) {
            while (num == capacity) { waitMessage(); }
            stack[num++] = msg.getArg();
            if (num==1) { notifyMessage();}
    }
    private void pop(Message msg) {
            while(num==0) { waitMessage(); }
            msg.sendReply(stack[--num]);
            if (num == (capacity -1)) { notifyMessage(); }
    }
}
```

Interface com.ibm.aglet.AgletProxy

- sendAsyncMessage(Message)
  modalità asincrona
- sendFutureMessage(Message)
  Invio di un mess posticipato
- sendMessage(Message)
  modalità sincrona
- sendOnewayMessage(Message)
  messaggio monodirezionale

MESSAGGIO SINCRONO

```
import …

public class MessageSincro extends Aglet{
    private String question=“”;
    public onCreation(Object arg){
        if (arg != null) { question = (String) arg; }
        else {question=“What’s your name? ”}
    }
    public void run(){
        URL destination = new URL("atp://saturn:8000/");
        AgletProxy proxy = cxt.createAglet(null, "BoredAglet", null);
        proxy = proxy.dispatch(destination);
        String reply = (String) proxy.sendMessage(new Message(question));
    }
}
```

MESSAGGIO ASINCRONO

```
import …

public class MessageAsyncro extends Aglet{
    private String question=“”; int count = 0;
    public onCreation(Object arg){
        if (arg != null) { question = (String) arg; }
        else {question=“What’s your name? ”}
    }
    public void run(){
        …
        FutureReply future = proxy.sendAsyncMessage(new Message(question));
        while(future.isAvailable() == false) {Thread.sleep(1000); count++;}
        System.out.println( (String)future.getReply()+”Time: ”+count );
    }
}
```

Package com.ibm.aglet.event

La tecnologia aglet mette a disposizione un modello basato sugli eventi per gestire la migrazione degli oggetti. Questo modello è rappresentato dalle seguenti interfacce:

- CloneListener
  - onClone()
  - onCloned()
  - onCloning()
- MobilityListener
  - onArrival()
  - onDispatching()
  - onReverting()
- PersistencyListener
  - onActivation()
  - onDeactivating()

Class com.ibm.aglet.Aglet 3

- addCloneListener(CloneListener)

  aggiunge uno specifico listener per gestire gli eventi di copia
- addMobilityListener(MobilityListener)

  aggiunge uno specifico listener per gestire gli eventi di migrazione dell'oggetto
- addPersistencyListener(PersistencyListener)

  aggiunge uno specifico listener per gestire gli eventi di persistenza

MobilityEvents 1

```
import com.ibm.aglet.*;
import com.ibm.aglet.event.*;

public class MobilityEvents extends Aglet implements MobilityListener {
    String history = null;
    int count = 0;

    public boolean handleMessage(Message msg) {
        if (msg.sameKind("dialog")) {       System.out.println(history);
                                            return true;
        }
        else if (msg.sameKind("clear")) {   history = "";
                                            count = 0;
                                            return true;
        }
        return false;
    }
```

MobilityEvents 2

```
public void onCreation(Object arg) {
        history = "[History of MobilityEvents]\n";
        count = 0;
        addMobilityListener(this);
}
public void onArrival(MobilityEvent ev) {
        count += 1;
        history += "onArrival: "
                + getAgletContext().getHostingURL().toString() + "\n";
}

public void onDispatching(MobilityEvent ev) {
        history += "onDispatching: "
                + getAgletContext().getHostingURL().toString() + "\n";
}
```

MobilityEvents 3

```
    public void onReverting(MobilityEvent ev) {
        history += "onReverting: "
                + getAgletContext().getHostingURL().toString() + "\n";
    }
    public void run() {
        setText("hopping count: " + count);
    }
}
```

TAHITI

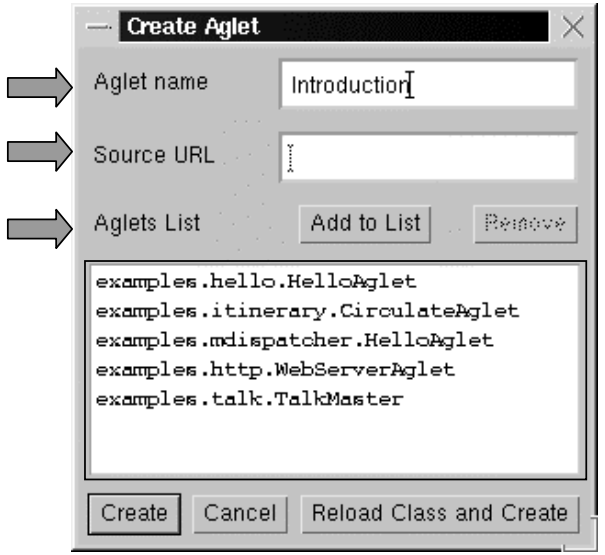
Tahiti è un'applicazione che funziona come un server di aglets, provvedendo allo svolgimento del loro ciclo di vita.

Il processo di creazione comprende le seguenti fasi:

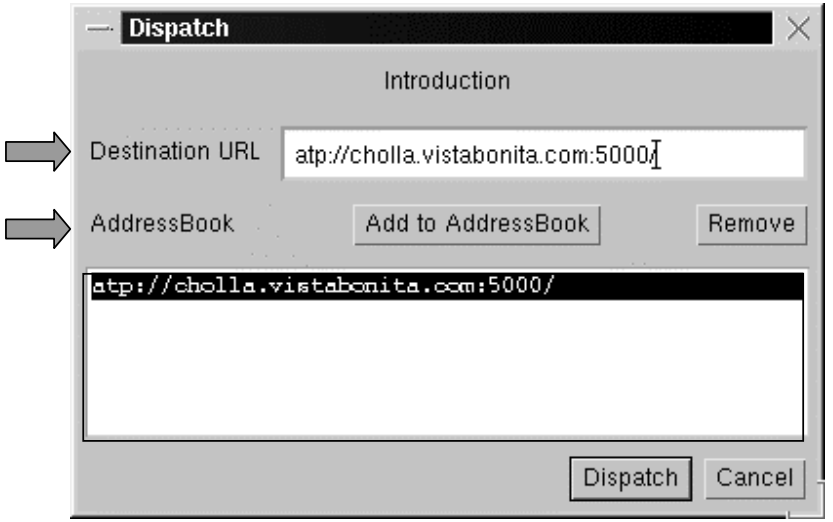
- Caricamento del class file
- Creazione dell'istanza di classe
- Inserimento dell' aglet in un contesto
- Invocazione del metodo onCreation()
- Invocazione del metodo run()

USO DI TAHITI 1

USO DI TAHITI 2



USO DI TAHITI 3



USO DI TAHITI 4

CONCLUSIONI

PRO

- Facilità d'utilizzo.
- Flessibilità.
- Autonomia.
- Linguaggio Java

CONTRO

- Non utilizzano strutture delle versioni di JDK successive alla 1.1.

RIFERIMENTI

1) http://www.infomedia.it/artic/Chiarelli/colpodocchio/codicemobile.htm

2) http://www.trl.ibm.com/aglets/

3) http://www.mokabyte.it

4) http://www.research.ibm.com/massdist

5) http://www.javaworld.com/