Activation

- **In generale i Sistemi ad oggetti distribuiti sono progettati per lavorare con oggetti persistenti. Dato che questi sistemi saranno composti da migliaia (forse milioni) di tali oggetti, sarebbe irragionevole che un oggetto rimanesse attivo, occupando risorse di sistema, per periodi indefiniti di tempo senza essere utilizzato.**

- **Inoltre, i client hanno bisogno di memorizzare riferimenti persistenti ad un oggetto, così che la comunicazione fra oggetti può essere riattivata dopo un crash di sistema, mentre tipicamente un riferimento ad un oggetto distribuito è valido solamente finché l'oggetto è attivo.**

1

Activation

- In sintesi:
  - è inutile avere attivi degli oggetti se non vengono utilizzati
  - è possibile che ci siano dei problemi sulla macchina server che obblighi a far "ripartire" il sistema
    - ripristinando i dati relativi agli oggetti server
- A questo scopo sono stati introdotti in RMI gli "*Oggetti Attivabili*"

Activation

- **L'attivazione di oggetti è un meccanismo per fornire riferimenti persistenti ad oggetti distribuiti, gestendo la loro esecuzione.**
- **In RMI, l'attivazione permette agli oggetti remoti di iniziare la loro esecuzione "on-demand".**
- **Quando su un oggetto remoto attivabile viene chiamato un metodo, se l' oggetto remoto non è in esecuzione in quel momento, il sistema di attivazione di RMI inizia l'esecuzione dell'oggetto in un JVM appropriata.**

Alcune definizioni

- **Un "*oggetto attivo*" è un oggetto remoto che è instanziato ed è in esecuzione su una JVM del sistema.**
- **Un "*oggetto passivo*" è un oggetto che non è stato ancora istanziato (o eseguito) su una JVM, ma che può essere portato in uno stato attivo.**
- **Trasformare un oggetto passivo in un oggetto attivo è un processo noto come "*attivazione*".**
- **Nel sistema RMI, si usa la cosiddetta "*lazy activation*". La lazy activation posticipa l'attivazione di un oggetto fino alla prima chiamata di un metodo dell'oggetto da parte di un client.**

Lazy Activation

- **E' implementata usando il meccanismo di *"riferimento remoto faulting":***
  - **In azione alla prima invocazione su un metodo di un oggetto**
- **Ogni riferimento remoto *faulting* contiene**
  - **un identificatore di attivazione (*activation identifier*)**
    - **serve per comunicare i dati per far attivare l'oggetto**
  - **un riferimento remoto (*live reference*) all'oggetto remoto**
    - **all'inizio vale *null,* poi contiene il riferimento dell'oggetto attivato**

Importante: anche in presenza di fallimenti (indicati attraverso RemoteException) RMI garantisce la semantica sui metodi chiamata "*at most once*": il metodo non è mai eseguito più volte

5

Protocollo di attivazione

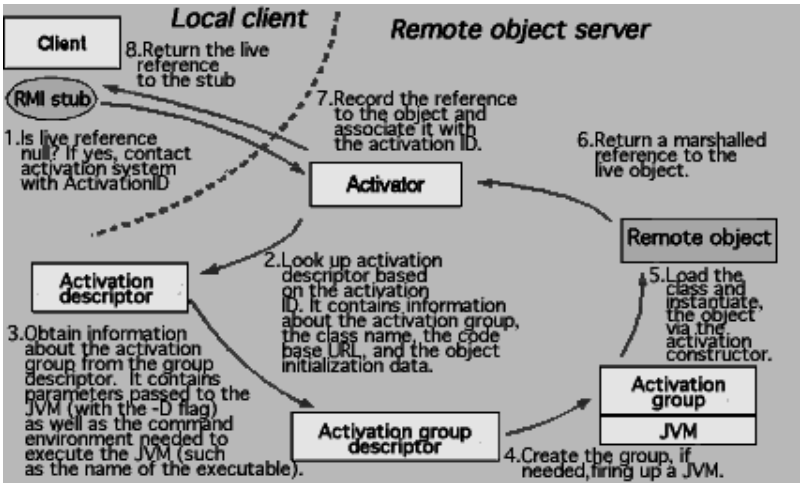
- In caso il riferimento remoto sia *null* il riferimento remoto *faulting* inizia un protocollo di attivazione
- Il protocollo di attivazione coinvolge diversi componenti:
  - Attivatore
  - Gruppo di attivazione
  - L'oggetto da attivare

Attivatore e gruppo di attivazione

- **L'attivatore (uno per host)**
  - **contiene un database di informazioni che associano identificatori di attivazione alle informazioni necessarie per la attivazione (classe, URL dal quale caricare le classi, dati per il bootstrap, etc)**
  - **manager di JVM (una per gruppo). Avvia le JVM (quando necessario) inoltrando le richieste di attivazione di oggetti al corretto gruppo di attivazione all'interno della JVM**

- **Nota che l 'attivatore tiene gli identificatori di attivazione per gli oggetti attivi come una cache, così che il gruppo non necessita di essere consultato per ogni richiesta di attivazione.**
- **Un *gruppo di attivazione* (uno per JVM) è un' entità che riceve una richiesta per attivare un oggetto nella JVM e ritorna l'oggetto attivato all' attivatore.**

Fasi del protocollo di attivazione



Protocollo di attivazione

Passi del protocollo:

- **l'attivatore usa le informazioni relative all' identificatore di attivazione**
- **se esiste il gruppo di attivazione, invia la richiesta al gruppo altrimenti, instanzia una JVM per il gruppo e poi invia la richiesta**
- **il gruppo carica la classe per l'oggetto, instanzia l'oggetto (con un costruttore particolare)**
- **ne restituisce il riferimento remoto all'attivatore**
- **che lo memorizza e lo restituisce al riferimento remoto *faulting (interno allo stub)***
- **che invia la richiesta di esecuzione di metodo direttamente all' oggetto remoto**

Come creare un oggetto attivabile

1. **Registrare il descrittore di attivazione per l'oggetto remoto**
   - **Si ottiene così l' Id dell' oggetto attivabile**
2. **Includere nell' oggetto remoto uno speciale costruttore usato per l'attivazione**

Il descrittore di attivazione può essere registrato:

- **Con il metodo statico *register()* della classe Activatable**
- **Con il metodo exportObject() di Activatable**
  - **Se l'oggetto non deriva da Activatable**
- **All'atto della creazione (se si vuole istanziare comunque l'oggetto senza una chiamata)**
  - **Usando i costruttori di Activatable**

Descrittore di attivazione

- **Un ActivationDesc contiene le informazioni necessarie ad attivare un oggetto. Contiene l' identificatore di gruppo di attivazione dell'oggetto, il nome della classe per l'oggetto, un codebase path (o URL) dal quale può essere caricata la classe dell'oggetto, ed un MarshalledObject che può contenere dati di inizializzazione da usare durante ogni attivazione.**
- **Un descrittore registrato col sistema di attivazione è consultato (durante il processo di attivazione) per ottenere informazioni da utilizzare per creare o attivare un oggetto. Il MarshalledObject nel descrittore dell'oggetto è passato come secondo argomento al costruttore dell'oggetto remoto in modo che venga usato durante l'attivazione.**

La classe *Activatable*

- **La classe Activatable è la classe principale per gestire ed usare oggetti attivabili. Da notare che prima che un oggetto possa essere registrato e/o attivato, è necessario che l' attivatore di sistema rmid sia in esecuzione su quell' host**

```
rmid –J-Djava.security.policy = rmid.policy
```

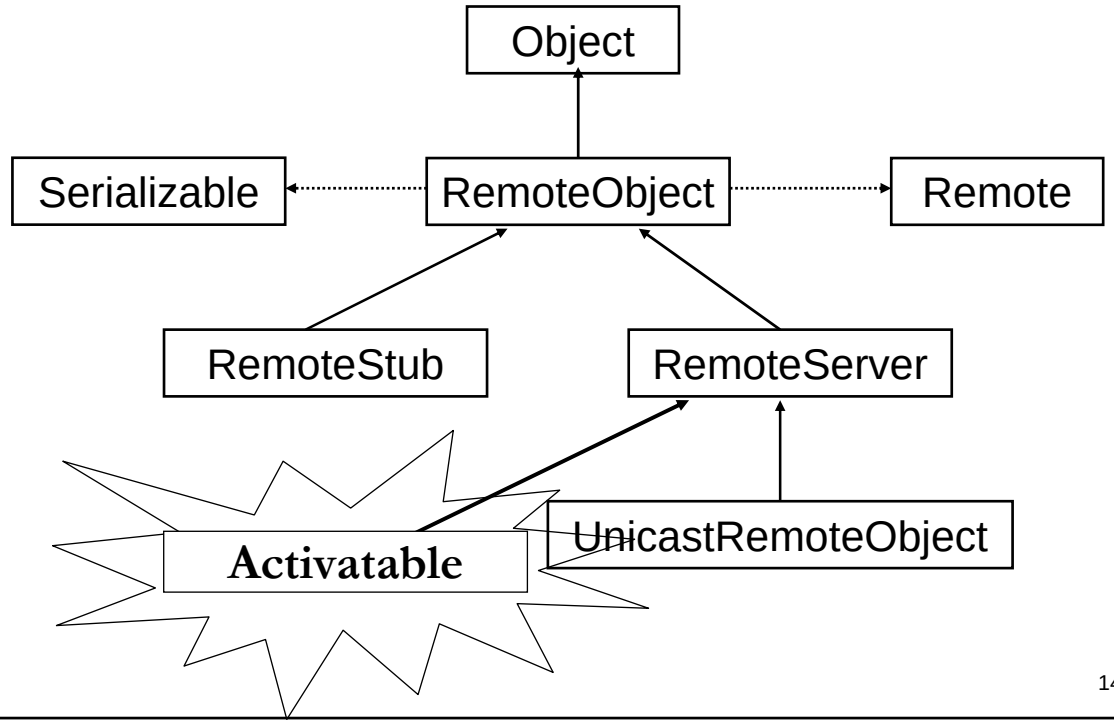
Fasi di creazione di un oggetto attivabile

- **Nel ciclo di vita di un oggetto attivabile esistono due fasi distinte, con due distinte JVM coinvolte**
  1. ***Fase di Setup*: l' oggetto viene registrato con il sistema di attivazione**
  2. ***Fase di creazione*: l' oggetto viene creato dal sistema di attivazione (attraverso l' attivatore)**

- **Nella fase di Setup si specifica la JVM su cui l'oggetto verrà eseguito, si specifica l'oggetto al sistema di attivazione, si registra l'oggetto con il sistema di attivazione e si esporta l'oggetto sul runtime di RMI**

13

Gerarchia delle classi RMI



14

I costruttori di Activatable

Esistono due costruttori per questa classe:

- **per instanziare e registrare**
  - **se si vuole istanziare un oggetto e contemporaneamente renderlo attivabile**
  - **può servire per permettere l'automatico restart dell'oggetto**
- **per attivare**
  - **chiamato dall' attivatore all' atto della prima invocazione**

Ognuno di questi costruttori ha una variante per l'utilizzo di socket personalizzati con una SocketFactory

- **crittografia (SSL, etc.), protocolli particolari**

I costruttori di Activatable - 1

- **Costruttori per istanziazione e attivazione**
  - **protected Activatable (*String codebase, java.rmi.MarshalledObject data, boolean restart, int port*) throws ActivationException, java.rmi.RemoteException;**
  - **protected Activatable (*String codebase, java.rmi.MarshalledObject data, boolean restart, int port, RMIClientSocketFactory csf,RMIServerSocketFactory ssf*) throws ActivationException, java.rmi.RemoteException;**

16

I costruttori di Activatable - 2

- **Costruttori per l' attivazione**
  - **se la porta è zero viene usato una porta anonima**

- **protected Activatable (*ActivationID id, java.rmi.MarshalledObject data, int port*)**
  throws java.rmi.RemoteException;

- **protected Activatable (*ActivationID id,java.rmi.MarshalledObject data, int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf*)**
  throws java.rmi.RemoteException;

Il daemon di attivazione: *rmid*

- **Permette di registrare e attivare gli oggetti in una Java Virtual Machine.**
- ***rmid* svolge il ruolo di attivatore**
- **eseguito con il comando**

  >rmid -J-Djava.security.policy=rmid.policy

  che permette il passaggio della security policy al demon
- **Usa una directory log (modificabile da parametri) per mantenere il database degli identificatori di attivazione**

Un esempio

Composto dalle seguenti classi:

- ***MyRemoteInterface*: interfaccia remota**
- ***Client*: client**
- ***MyRemoteInterfaceImplementation*: oggetto attivabile**
  - **deriva da Activatable**
- ***Setup*:**
  - **registra l'oggetto MyRemoteInterfaceImplementation (o meglio il suo stub) all' activator (rmid) con le informazioni necessarie per permetterne la attivazione**

19

MyRemoteInterface

```
import java.rmi.*;
public interface MyRemoteInterface extends Remote {
   public Object callMeRemotely() throws RemoteException;
}
```

MyRemoteInterfaceImplementation

```
import java.rmi.*;
import java.rmi.activation.*;

public class MyRemoteInterfaceImplementation extends
   Activatable implements MyRemoteInterface {

   public MyRemoteInterfaceImplementation(ActivationID id,
       MarshalledObject data)throws RemoteException {
       // Register the object with the activation system
       // then export it on an anonymous port
       super(id, 0);
    }
   public Object callMeRemotely() throws RemoteException {
       return "Success";
    }
}
```

class Setup

```
import java.rmi.*;
import java.rmi.activation.*;
import java.util.Properties;

public class Setup {

    /* This class registers information about the
       ActivatableImplementation
       class with rmid and the rmiregistry
     */

    public static void main(String[] args) throws Exception
       {
       System.setSecurityManager(new RMISecurityManager());
       Properties props = new Properties();
       props.put("java.security.policy","policy");
```

[to be continued…]

class Setup

```
ActivationGroupDesc.CommandEnvironment ace = null;
ActivationGroupDesc exampleGroup =
                    newActivationGroupDesc(props, ace);

// Once the ActivationGroupDesc has been created, register it
// with the activation system to obtain its ID

ActivationGroupID agi =
          ActivationGroup.getSystem().registerGroup(exampleGroup);

// The "location" String specifies a URL from where the class
// definition will come when this object is requested (activated).

String location = "file:/";

MarshalledObject data = null;
```

[to be continued…]

class Setup

```
    ActivationDesc desc = new ActivationDesc(agi,
         "MyRemoteInterfaceImplementation", location, data);

    // Register with rmid, obtaining the object's activation ID
    MyRemoteInterface mri =
         (MyRemoteInterface)Activatable.register(desc);

    System.out.println("Got the stub for ActivatableImplementation");

    // Bind the stub to a name in the registry running on 1099
    Naming.rebind("MyRemoteInterfaceImplementation", mri);

    System.out.println("Exported ActivatableImplementation");

    System.exit(0);
   }

}
```

Client

```
import java.rmi.*;

public class Client {
   public static void main(String args[]) {

   String server = "localhost";
   if (args.length < 1) {
       System.out.println ("Usage: java Client <rmihost>");
       System.exit(1);
   } else {
       server = args[0];
   }
   // Set a security manager so that the client can
   // download the activatable object's stub
   System.setSecurityManager(new RMISecurityManager());
```

[to be continued…]

25

Client

```
try {

   String location = "rmi://" + server +
                     "/MyRemoteInterfaceImplementation";
   MyRemoteInterface mri =
              (MyRemoteInterface)Naming.lookup(location);

   System.out.println("Got a remote reference to the object
                           that" +" extends Activatable.");

   // The String "result" will be changed to "Success" by
   // the remote method call

   String result = "failure";
```

[to be continued…]

Client

```
        System.out.println("Making remote call to the
                                                server");
        result = (String)mri.callMeRemotely();
        System.out.println("Returned from remote call");
        System.out.println("Result: " + result);

    } catch (Exception e) {e.printStackTrace();}

  }
}
```