le cnam

Java Card RMI

Java Card RMI

Samia Bouzefrane

Maître de Conférences

CEDRIC –CNAM

samia.bouzefrane@cnam.fr
http://cedric.cnam.fr/~bouzefra

le cnam

Rappel de Java RMI

Rappel de Java RMI

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Rappel : Appel local (interface et objet)

```
public interface ReverseInterface {
String reverseString(String chaine);
}


public class Reverse implements ReverseInterface
{
public String reverseString (String ChaineOrigine){

int longueur=ChaineOrigine.length();
StringBuffer temp=new StringBuffer(longueur);
 for (int i=longueur; i>0; i--) {
   temp.append(ChaineOrigine.substring(i-1, i));}
 return temp.toString();
 }
}
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

le cnam

Rappel : Appel local (programme appelant)

```
import ReverseInterface;

public class ReverseClient
{
    public static void main (String [] args)
    {  Reverse rev = new Reverse();
          String result = rev.reverseString (args [0]);
          System.out.println ("L'inverse de "+args[0]+" est
"+result);

    }
}

$javac *.java
$java ReverseClient Alice
L'inverse de Alice est ecilA
$
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Java RMI
Remote Method Invocation

permet la communication entre machines virtuelles Java (JVM) qui peuvent se trouver physiquement sur la même machine ou sur deux machines distinctes.

Présentation

RMI est un système d'objets distribués constitué uniquement d'objets java ;

- RMI est une **A**pplication **P**rogramming **I**nterface (intégrée au JDK 1.1 et plus) ;

- Développé par Oracle ;

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

• Mécanisme qui permet l'appel de méthodes entre objets Java qui s'exécutent éventuellement sur des JVM distinctes ;

• L 'appel peut se faire sur la même machine ou bien sur des machines connectées sur un réseau ;

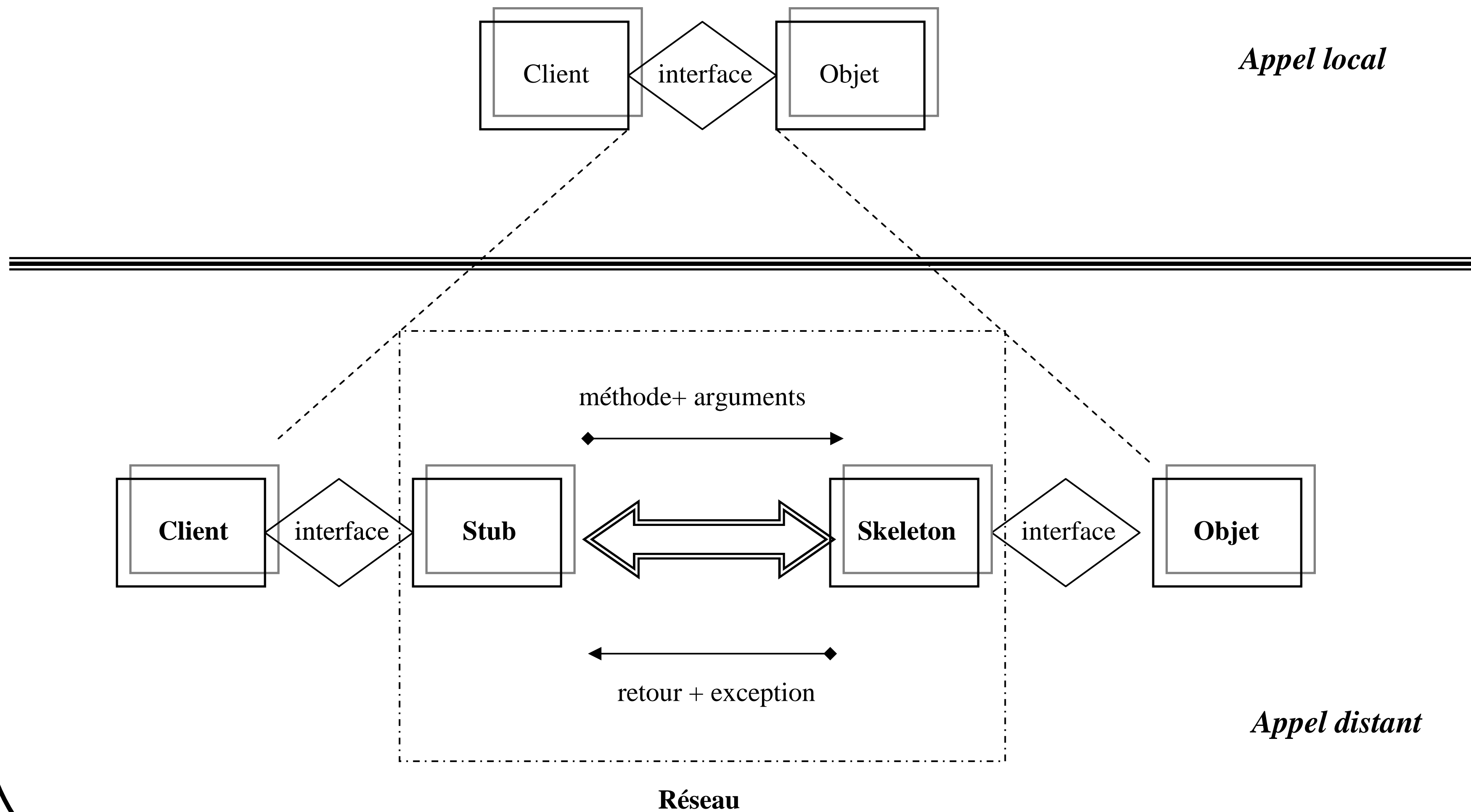
• Utilise les sockets ;

• Les échanges respectent un protocole propriétaire : **R**emote **M**ethod **P**rotocol ;

• RMI repose sur les classes de sérialisation.

le cnam

Appel local versus Appel à distance



Les amorces (Stub/Skeleton)

- Elles assurent le rôle d'adaptateurs pour le transport des appels distants

- Elles réalisent les appels sur la couche réseau

- Elles réalisent l'assemblage et le désassemblage des paramètres (*marshalling*, *unmarshalling*)

- Une référence d'objets distribué correspond à une référence d'amorce

- Les amorces sont créées par le générateur **`rmic`**.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

La couche des références d'objets
Remote Reference Layer

- Permet d'obtenir une référence d'objet distribué à partir de la référence locale au stub ;

- Cette fonction est assurée grâce à un service de noms **`rmiregister`** (qui possède une table de hachage dont les clés sont des noms et les valeurs sont des objets distants) ;

- Un unique **`rmiregister`** par JVM ;

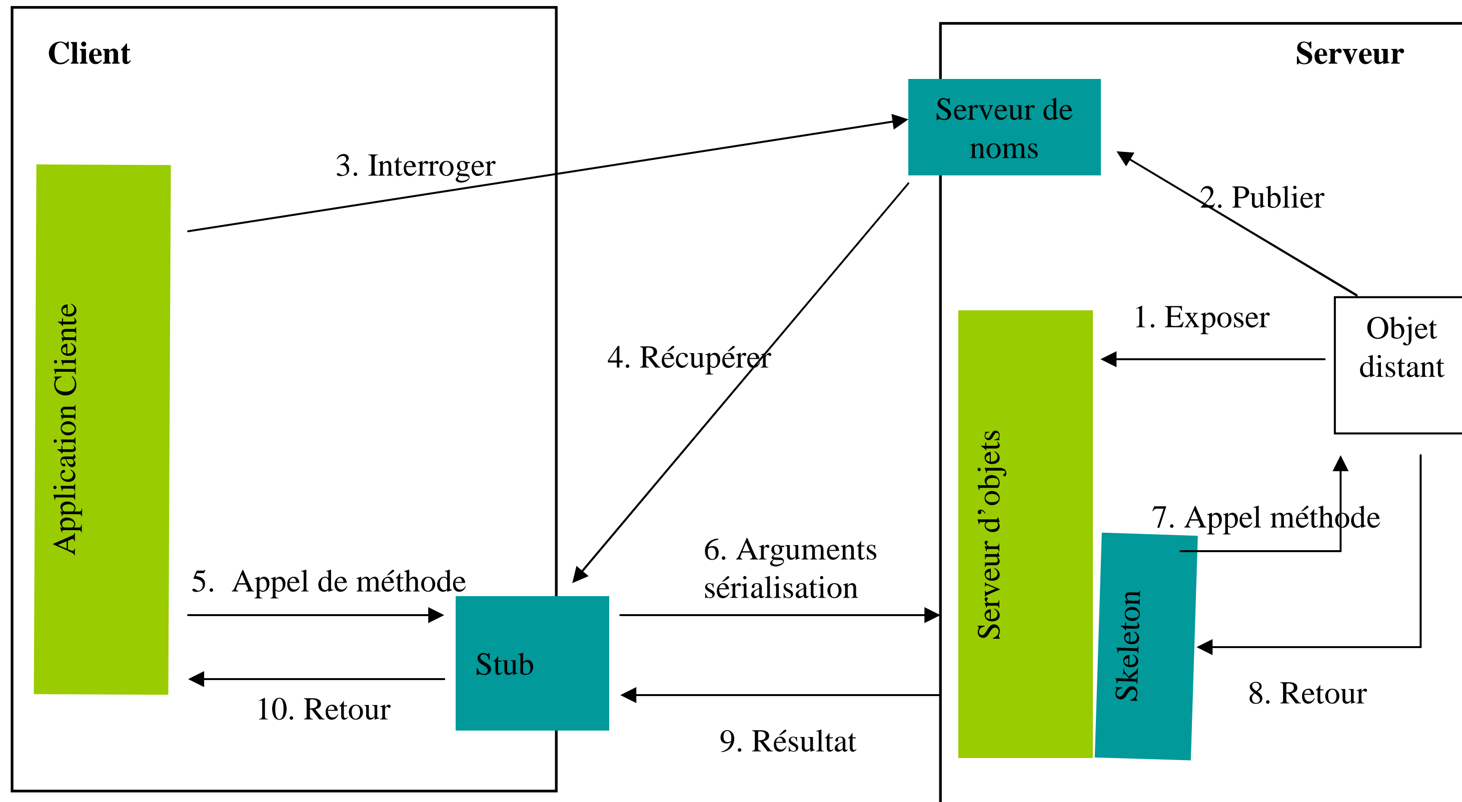
- **`rmiregister`** s'exécute sur chaque machine hébergeant des objets distants ;

- **`rmiregister`** accepte des demandes de service sur le port 1099;

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

le cnam

Etapes d'un appel de méthode distante



Développer une application avec RMI : Mise en œuvre

1. Définir une interface distante (**`Xyy.java`**) ;

2. Créer une classe implémentant cette interface (**`XyyImpl.java`**) ;

3. Compiler cette classe (**`javac XyyImpl.java`**) ;

4. Créer une application serveur (**`XyyServer.java`**) ;

5. Compiler l'application serveur ;

6. Créer les classes stub et skeleton à l'aide de **`rmic XyyImpl_Stub.java`** et **`XyyImpl_Skel.java`** (**`Skel`** n'existe pas pour les versions >1.2) ;

7. Démarrage du registre avec **`rmiregistry`** ;

8. Lancer le serveur pour la création d'objets et leur enregistrement dans **`rmiregistry`**;

9. Créer une classe cliente qui appelle des méthodes distantes de l'objet distribué (**`XyyClient.java`**) ;

10. Compiler cette classe et la lancer.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Inversion d'une chaîne de caractères à l'aide d'un objet distribué

Invocation distante de la méthode **`reverseString`**() d'un objet distribué qui inverse une chaîne de caractères fournie par l'appelant.

On définit :

- **`ReverseInterface.java`** : interface qui décrit l'objet distribué
- **`Reverse.java`** : qui implémente l'objet distribué
- **`ReverseServer.java`** : le serveur RMI
- **`ReverseClient.java`** : le client qui utilise l'objet distribué

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

le cnam

Fichiers nécessaires

Côté Client

- l'interface : `ReverseInterface`
- le client : `ReverseClient`

Côté Serveur

- l'interface : `ReverseInterface`
- l'objet : `Reverse`
- le serveur d'objets : `ReverseServer`

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Interface de la classe distante

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ReverseInterface extends Remote {
String reverseString(String chaine) throws
RemoteException;
}
```

le cnam

Implémentation de l'objet distribué

```
import java.rmi.*;
import java.rmi.server.*;
public class Reverse extends UnicastRemoteObject implements
ReverseInterface
{
public Reverse() throws RemoteException {
     super();
}
public String reverseString (String ChaineOrigine) throws
RemoteException {

     int longueur=ChaineOrigine.length();
     StringBuffer temp=new StringBuffer(longueur);
     for (int i=longueur; i>0; i--)
     {
          temp.append(ChaineOrigine.substring(i-1, i));
     }
     return temp.toString();
} }
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Le serveur

- Programme à l'écoute des clients ;

•On lance rmiregistry

```
LocateRegistry.createRegistry(1099);
```

•Enregistre l'objet distribué dans `rmiregistry`

```
Naming.rebind("rmi://hote.cnam.fr:1099/MyReverse", rev);
```

Le serveur

```
import java.rmi.*;
import java.rmi.server.*;

public class ReverseServer {
public static void main(String[] args)
{
    try {
// lancement de rmiregistry
   LocateRegistry.createRegistry(numPortRMIRegistry);
   System.out.println( "Serveur : Construction de l'implémentation ");
   Reverse rev= new Reverse();
   System.out.println("Objet Reverse lié dans le RMIregistry");
   Naming.rebind("rmi://sinus.cnam.fr:1099/MyReverse", rev);
   System.out.println("Attente des invocations des clients …");
  }
   catch (Exception e) {
       System.out.println("Erreur de liaison de l'objet Reverse");
       System.out.println(e.toString());
        }
} // fin du main
} // fin de la classe
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Le Client

• Obtient une référence d’objet distribué :

```
ReverseInterface ri = (ReverseInterface) Naming.lookup
("rmi://sinus.cnam.fr:1099/MyReverse");
```

• Exécute une méthode de l’objet :

```
String result = ri.reverseString ("Terre");
```

Le Client

```
import java.rmi.*;
import ReverseInterface;

public class ReverseClient
{
    public static void main (String [] args)
    {   try{
          ReverseInterface rev = (ReverseInterface) Naming.lookup
                ("rmi://sinus.cnam.fr:1099/MyReverse");
          String result = rev.reverseString (args [0]);
          System.out.println ("L'inverse de "+args[0]+" est "+result);
        }
        catch (Exception e)
        {
            System.out.println ("Erreur d'accès à l'objet distant.");
            System.out.println (e.toString());
        }
    }
}
```

Compilation et Exécution

- Lancer le serveur :

```
Serveur :Construction de l'implémentation
Objet Reverse lié dans le RMIregistry
Attente des invocations des clients …
```

- Exécuter le client :

```
L'inverse de Alice est ecilA
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

le cnam

Écriture d’une applet pour une plateforme Java Card

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Écriture d’une Applet (Cardlet)

-Définir un AID pour le package qui va contenir l’applet

-Définir un AID pour l’applet elle-même

-Définir les commandes APDU à échanger entre le terminal et l’applet

-Écriture de l’applet

Structure d'une Applet/1

-Inclure **`javacard.framework`**
- l'applet doit hériter de la classe Applet
- déclarer les constantes et variables
- dans le constructeur de l'applet, prévoir l'initialisation des variables déclarées et l'enregistrement auprès du JCRE

```
Monnaie (byte [] bArray, short bOffset, byte bLength) {
        balance = (short) 0x1000;
        register();
   }
```

- Une méthode *install* pour l'installation de l'applet auprès du JCRE

```
public static void install (byte [] bArray, short bOffset, byte bLength) {
       new Monnaie (bArray, bOffset, bLength);
   }
```

L'installation = création de l'objet Applet et son enregistrement.

- Deux méthodes *select* et *deselect* pour sélectionner ou désélectionner l'applet, car avant toute utilisation d'une applet, celle-ci doit être sélectionnée.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Structure d'une Applet/2

-Une méthode *process* qui traite toute commande APDU reçue du terminal:

```
public void process (APDU apdu) throws ISOException {
        byte [] buffer = apdu.getBuffer();
if (selectingApplet()) return;
….
    }
```

- La méthode *process* doit faire appel à *selectingApplet*() car même la commande APDU *SELECT* sera reçue par la méthode *process*.

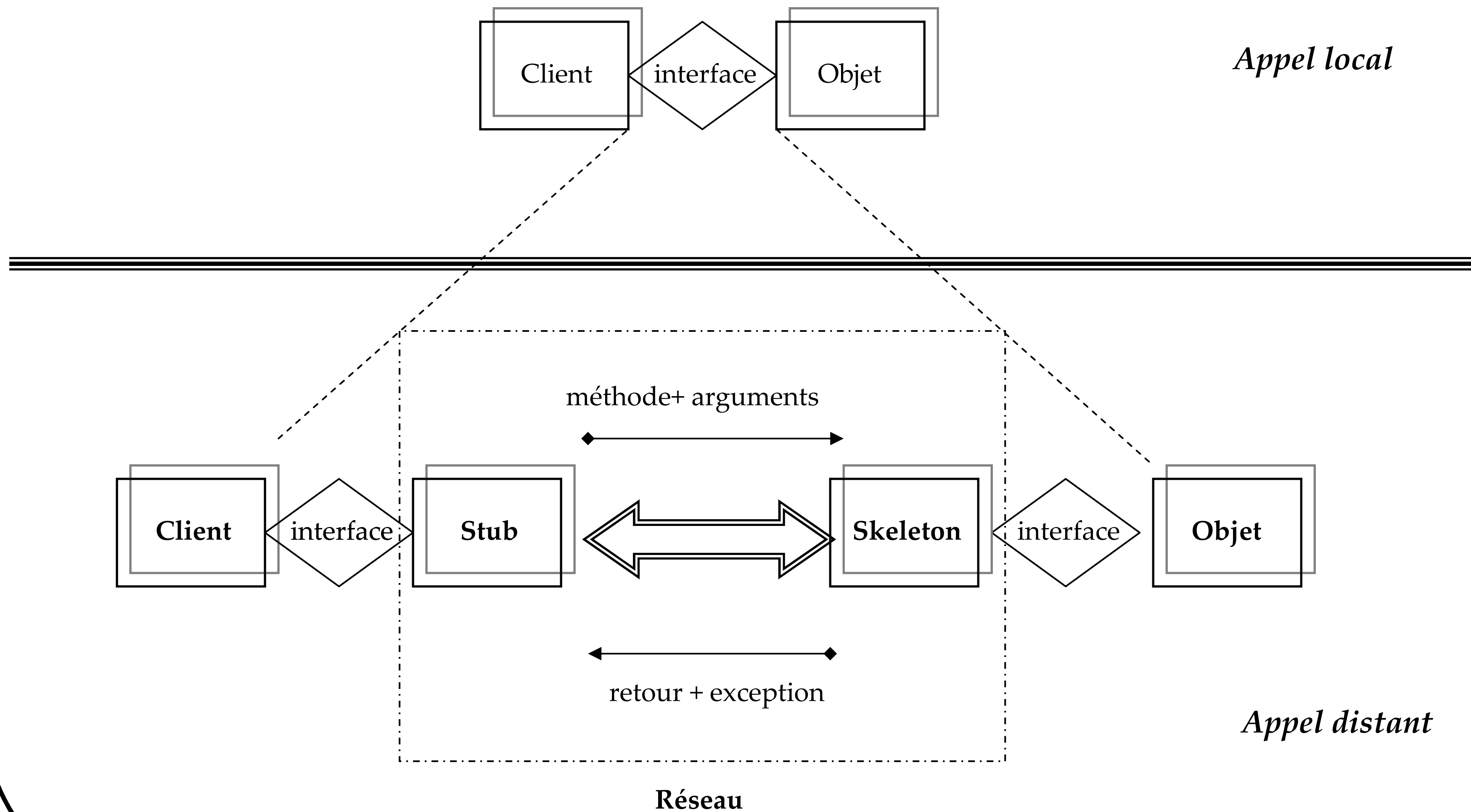
- Toute commande APDU traitée nécessitera une réponse APDU

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Écriture d'une applet pour une plateforme Java Card RMI

le cnam

Appel local versus Appel à distance



samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

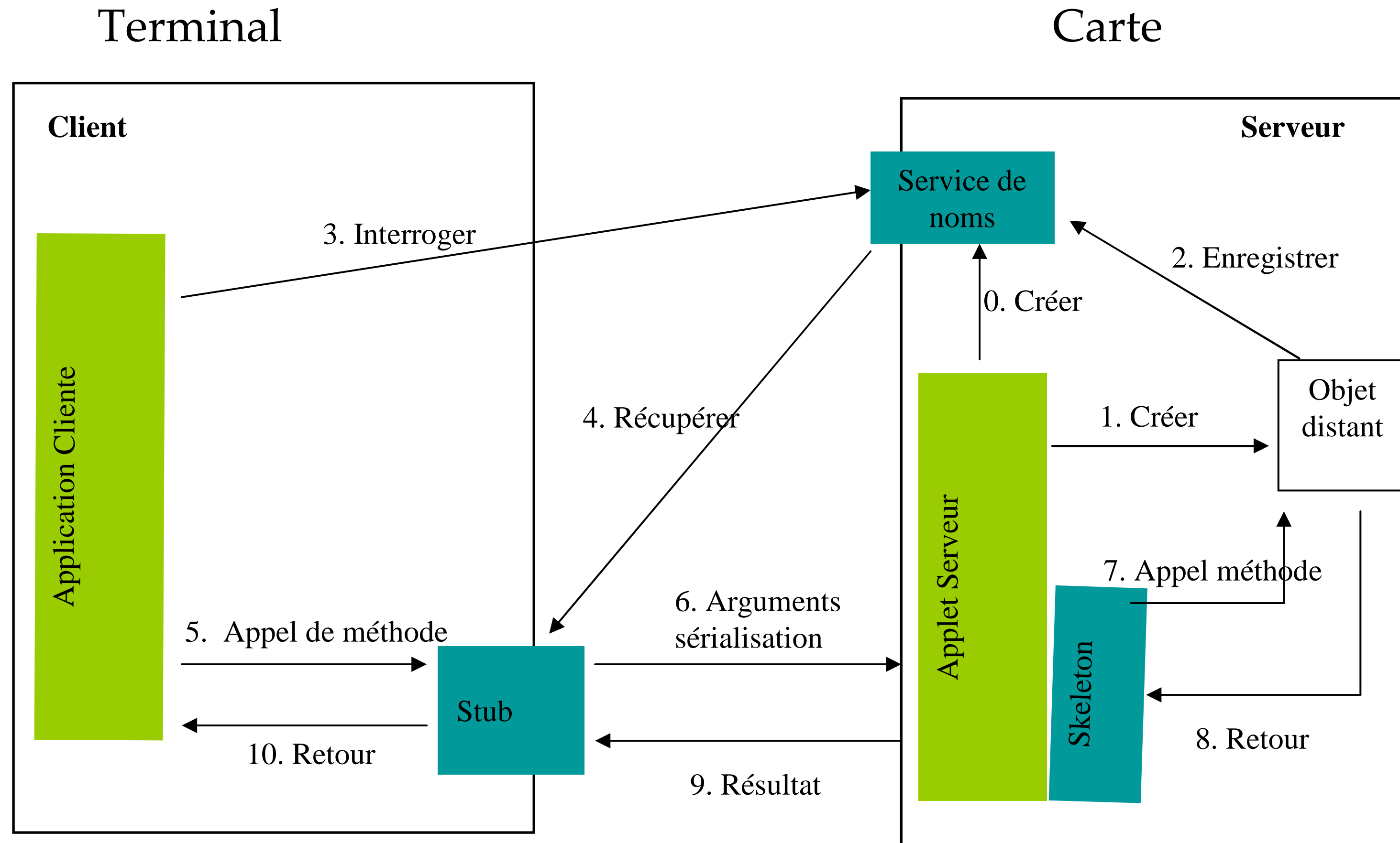
Les amorces (Stub/Skeleton)

- Le stub est le représentant local de l'objet distant
- Le stub assemble les paramètres pour leur transfert à la JVM distante
- Le stub désérialise la valeur ou l'exception renvoyée et la propage au client ;
- Le skeleton désérialise les paramètres pour appeler la méthode distante ;
- Le skeleton assemble le résultat (valeur renvoyée ou exception) à destination de l'appelant.
- Les amorces sont créées par le générateur rmic.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

le cnam

Etapes d'un appel de méthode distante en JCRMI



Le service de noms

- **`rmiService`** est un service de noms qui possède une table de hachage dont les clés sont des noms et les valeurs sont des références aux objets distants ;

- **`rmiService`** est un objet créé par l'applet ;

- Tout objet distant créé par l'applet est enregistré dans **`rmiService`** ;

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Développer une application JCRMI
Côté Serveur (carte)

1. Définir une interface distante (`Xyy.java`) ;

2. Créer une classe implémentant cette interface (`XyyImpl.java`) qui sera l'objet distant (ou le service)

3. Compiler cette classe (`javac XyyImpl.java`) ;

4. Créer une applet qui jouera le rôle du serveur d'objets (`XyyServer.java`) ;

5. Compiler l'applet serveur et l'installer sur la carte ;

6. L'installation de l'applet sur la carte permet la création du service de noms **rmiService** et des objets à enregistrer auprès de ce service;

Développer une application JCRMI
Côté Client (terminal)

7. Créer une classe cliente qui appelle les méthodes distantes de l'objet distant (**XyyClient.java**) ;

8. Compiler cette classe et lancer son exécution.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Objectifs du développement en JCRMI

- Appeler le service (méthodes) offert par l'applet comme si le service se trouve sur le même terminal que le client (faire abstraction de l'endroit physique où se trouve le service à appeler)

- Avoir un niveau d'abstraction plus élevé afin de ne pas manipuler par exemple le protocole APDU qui est d'un niveau plus bas

- Séparer les spécifications fonctionnelles de l'application des aspects non-fonctionnels (relatifs à la communication avec la carte).

L'interface de l'objet distant

- Inclure en plus de `javacard.framework` le package `java.rmi`

- L'interface doit hériter de la classe `java.rmi.Remote`

```
public interface Purse extends Remote {

  // on déclare les constantes à utiliser
         // exemple
    public static final short BAD_ARGUMENT  = (short)0x6002;


// on définit les signatures des méthodes avec une levée
// de l'exception RemoteException et UserException si exception
// définie par l'utilisateur
// exemple :
public void debit(short m) throws RemoteException, UserException;

}
```

Implémenter l'interface de l'objet distant

- La classe d'implémentation doit étendre `CardRemoteObject` pour désigner l'objet distant
- Cette classe doit implémenter l'interface pour associer un code aux méthodes définies dans l'interface

```
public class PurseImpl extends CardRemoteObject implements Purse {

// définir le constructeur qui se contente d'appeler la classe mère
     public PurseImpl() {  super(); }

// associer un code à chaque méthode définie dans l'interface
// en utilisant Java standard
// Exemple de la méthode debit

     public void debit(short m) throws RemoteException, UserException {
     if(m<=0) UserException.throwIt(BAD_ARGUMENT);

             if((short)(balance-m) < 0) UserException.throwIt(UNDERFLOW);

             balance -=m;
     }
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Paramètres des méthodes distantes

➢ Les arguments des méthodes distantes sont de type simple (*boolean*, *byte*, *short* et *int*), ou bien un tableau à une dimension de type simple.
Le type *int* n'est pas supporté par toutes les plateformes.

➢ La valeur de retour doit être aussi de type simple (*boolean*, *byte*, *short* ou *int*) ou bien un type d'interface distante, ou encore de type *void*.

➢ Les objets distants sont passés par référence. Une référence à un objet distant est en fait une référence au stub qui est le représentant local de l'objet côté client.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Construire l'applet Serveur/1

- **L'AID attribué à l'applet doit être connu du client (terminal)**

- **L'objet distant doit être instancié et initialisé, ce qui sera fait dans la méthode *install*().**

- **Les objets distants communiqueront avec l'extérieur via la méthode *process()* de l'applet.**

- **La méthode *install*() permet l'enregistrement de l'applet auprès du JCRE pour qu'elle soit sélectionnée après**

- **Les méthodes *select*() et *deselect*() pour la sélection et la désélection de l'applet**

Construire l'applet Serveur/2

➢ **Le constructeur de l'applet serveur contient la ligne suivante :**

- `RemoteService rmi = new RMIService(new test.PurseImpl() ) ;`

  permet de créer l'objet sur la carte et de lui associer une référence RMI qui sera accessible par le client

- `dispatcher = new Dispatcher((short)1) ;`
  `dispatcher.addService(rmi, Dispatcher.PROCESS_COMMAND) ;`

  définit un seul service à rajouter dans le service de noms (`RMIService`)

➢ **La méthode *process*() propage la commande APDU vers le service RMI:**

- `dispatcher.process(apdu)`

  délègue le traitement de la commande APDU au service RMI qui propage l'appel à l'objet distant.

Comportement de l'appel côté Serveur

- La commande APDU SELECT invite le service RMI à renvoyer la référence de l'objet distant.

- Une invocation de méthode distante est récupérée par l'applet qui délègue l'appel au service RMI qui propage l'appel vers l'objet distant. La valeur retournée emprunte le chemin inverse.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

le cnam

Exemple d'applet Serveur

```
package RMITest;
import javacard.framework.service.*;
import javacard.framework.*;
import java.rmi.*;
public class PurseApplet extends Applet {
  private Dispatcher dispatcher;
  protected PurseApplet() {
        super();
        RemoteService rmi = new RMIService(new RMITest.PurseImpl());
        dispatcher = new Dispatcher((short)1);
        dispatcher.addService(rmi, Dispatcher.PROCESS_COMMAND);
        }
  public static void install(byte[] buffer, short offset, byte length) {
        //instantiation et enregistrement de l'applet
 (new PurseApplet()).register(buffer,(short)(offset+1),(byte)buffer[offset]);
        }

  public void process(APDU apdu) {
        dispatcher.process(apdu);
  }
  public boolean select() {/* rien a faire*/
        return super.select();
  }
  public void deselect() {/* rien a faire*/
        super.deselect();
  }
}
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Écriture du client

➢ **Dans notre environnement, le client est généré automatiquement. Il restera uniquement à faire des invocations aux méthodes distantes.**

➢ **Opérations faites par le client :**

- Initialisation d'un CardAccessor
- Connexion à la plateforme JCRMI
- Sélection de l'applet serveur
- Obtention de la référence de l'objet distant
-Faire les invocations des méthodes de l'objet distant
- Mise hors tension de la carte après « usage ».

Écriture du client: Initialisation d'un CardAccessor

L'interface **`CardAccessor`** est utilisée par la plateforme JCRMI pour communiquer avec la carte. L'initialisation de la carte est effectuée à la création du **`CardAccessor`** :

```
CardAccessor ca = (CardAccessor) new PCSCAccessor();
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Écriture du client: connexion à la plateforme JCRMI

La classe `JCRMIConnect` permet la connexion du client à la plateformes JCRMI.
Elle fournit les fonctions nécessaires pour sélectionner l'applet.
Le constructeur a besoin d'un `CardAccessor`.

```
// Créer une connexion RMI a la plateforme Java Card

JCRMIConnect jcrmi = new JCRMIConnect(ca);
```

Écriture du client: Sélection de l'Applet Serveur

Avant toute invocation de méthodes distantes, le client doit sélectionner l'applet serveur adéquate, identifiée sur la carte par son AID (défini lors de l'installation) :
// sélectionner l'applet Java Card:

```
jcrmi.selectApplet(AID,JCRMIConnect.REF_WITH_CLASS_NAMES);
```


samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Écriture du client: Obtenir la référence de l'objet distant

Le client doit obtenir la référence de l'objet distant pour faire les invocations.

La référence retournée est celle du Stub.

// obtenir la référence initiale de l'interface `PurseInterf`

```
PurseInterf myPurse = (PurseInterf) jcrmi.getInitialReference();
if (myPurse == null) {
        throw new RuntimeException("Cannot get the initial reference");
}
```

Écriture du client: Utiliser l'objet distant dans les invocations

Le client utilise la référence de l'objet distant pour faire l'invocation des méthodes.

// appel de la méthode **debit**()

```
short balance = myPurse.debit ( debitAmount );
```

Écriture du client: Fin de l'utilisation de la carte

Terminer l'utilisation de la carte (libération de ressources) :

```
ca.closeCard();
```

le cnam

Exemple de client

```
Package rmiClient;
import com.sun.javacard.clientlib.*;
import com.sun.javacard.rmiclientlib.*;
import java.rmi.*;
import javacard.framework.UserException;
import rmi.PurseInterf;

public class MonClient {
 private static byte AID []= {
     (byte) 0xA0, (byte) 0x00, (byte) 0x00, (byte) 0x18,
     (byte) 0x50, (byte) 0x00, (byte) 0x00, (byte) 0x00,
     (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x52,
     (byte) 0x41, (byte) 0x44, (byte) 0x41
   };
  static CardAccessor ca = null; PurseInterf wallet;
  public MonClient() {   }

  public static void main(String[] args) {
    Instance init try {
    ca = (CardAccessor)new PCSCAccessor();
    JCRMIConnect jcrmi = new JCRMIConnect(ca);

jcrmi.selectApplet(AID,JCRMIConnect.REF_WITH_INTERFACE_NAMES
);
       PurseInterf myPurse = (PursetInterf)
jcrmi.getInitialReference();

       myPurse.crediter((short)10000);
       myPurse.getBalance());
     }
     catch(Exception ex) {
      System.out.println(ex.getMessage());
    }
```

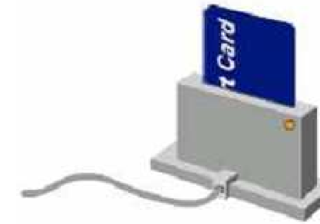

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Exemple de client (suite)

```
finally{
      try {
        ca.closeCard();
       } catch(Exception e) {
        System.out.println(e.getMessage());
      }
     }
   } // end main
 } // end class
```

Résumé/1

ATR



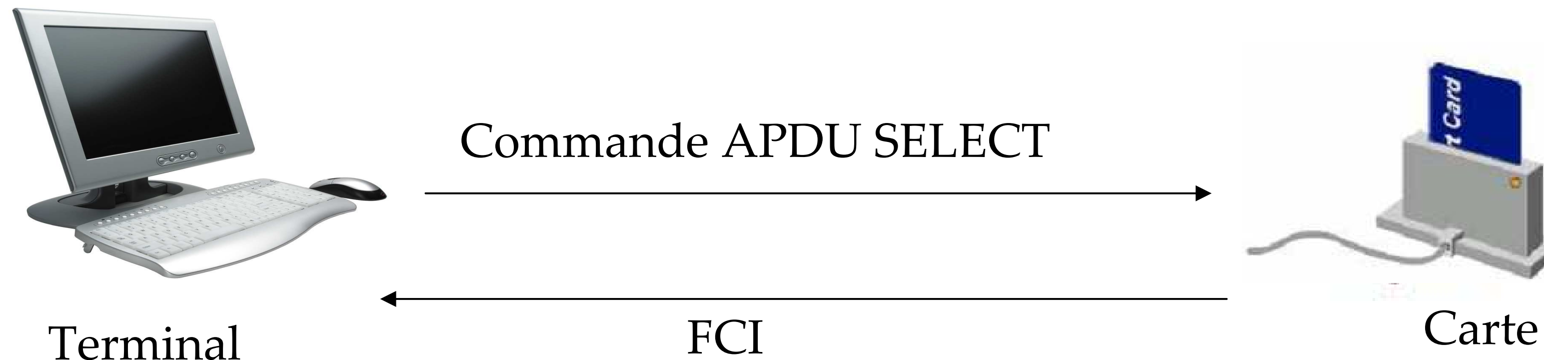
Terminal

Carte

-Dès la mise sous tension de la carte, elle envoie l'ATR au terminal pour s'identifier.

- L'ATR est prise en compte par l'objet **`CardAccessor`**

Résumé/2

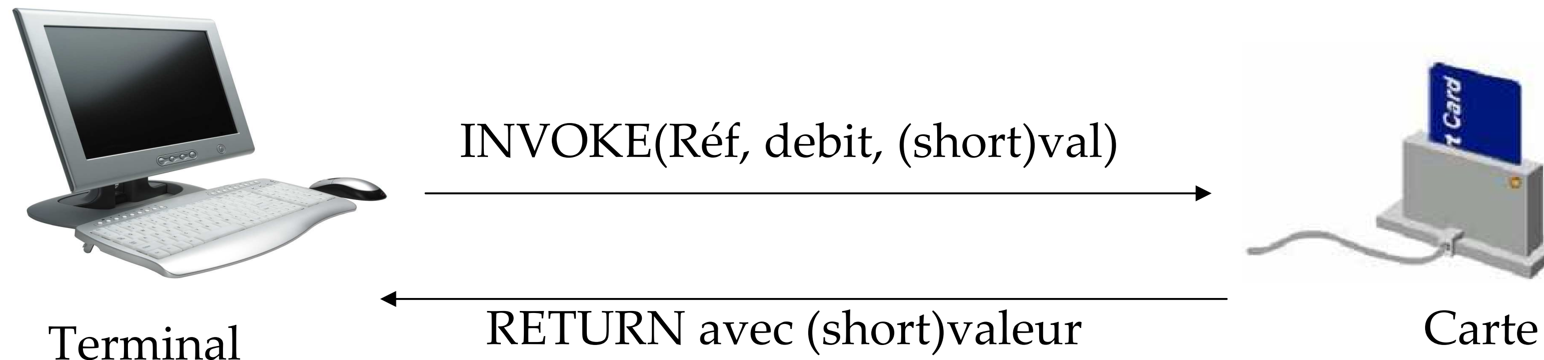


Lorsque le client appelle la méthode **`selectApplet`** de **`JavaCardRMIConnect`**, il envoie une commande APDU SELECT à la carte via l'objet **`CardAccessor`**.

L'instance de **`RMIService`** répond en envoyant une réponse APDU FCI (File Control Information) qui inclut la référence à l'objet distant.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Résumé/3



- Lorsque le client appelle la méthode **`debit`**() de l'interface distante **`Purse`**, l'objet **`PurseImpl_Stub`** envoie une commande d'invocation vers la carte via l'objet **`CardAccessor`**, en identifiant la référence distante, la méthode et paramètres de l'invocation.

- L'instance **`RMIService`** de l'applet désérialise l'information et appelle la méthode **`debit`**() de l'objet distant et retourne la réponse dans une réponse APDU.

samia.bouzefrane@cnam.fr - CEDRIC (CNAM) -

Conclusion: Java Card RMI

➢**Avantages**

▪ offre un niveau d'abstraction plus élevé (pas de manipulation de commandes/réponses APDU qui deviennent d'un niveau plus bas)

▪ se concentre sur l'aspect fonctionnel (l'aspect non-fonctionnel telle que la communication selon le protocole APDU est à la charge de la plateforme)

▪ le code écrit est réutilisable sur d'autres plateformes JCRMI car il utilise les services de RMI sans faire appel à des caractéristiques intrinsèques (clés de sécurité, commandes APDU particulières) à la carte: c'est le principe de PIM.

➢**Inconvénient**

▪ JCRMI cache les spécificités de la carte et les problèmes liés à son interaction (la gestion de la sécurité, le protocole APDU, etc.) qui sont importants à connaître si on s'intéresse à ce domaine.

Bibliographie

-Application Programming Notes : Java Card™ Platform, Version 2.2.1, Sun MicroSystems, 2003.

- Samia Bouzefrane, Java RMI, Support de Cours en SMB111, http://cedric.cnam.fr/~bouzefra