

# Compte-rendu de TP

---

ARAR

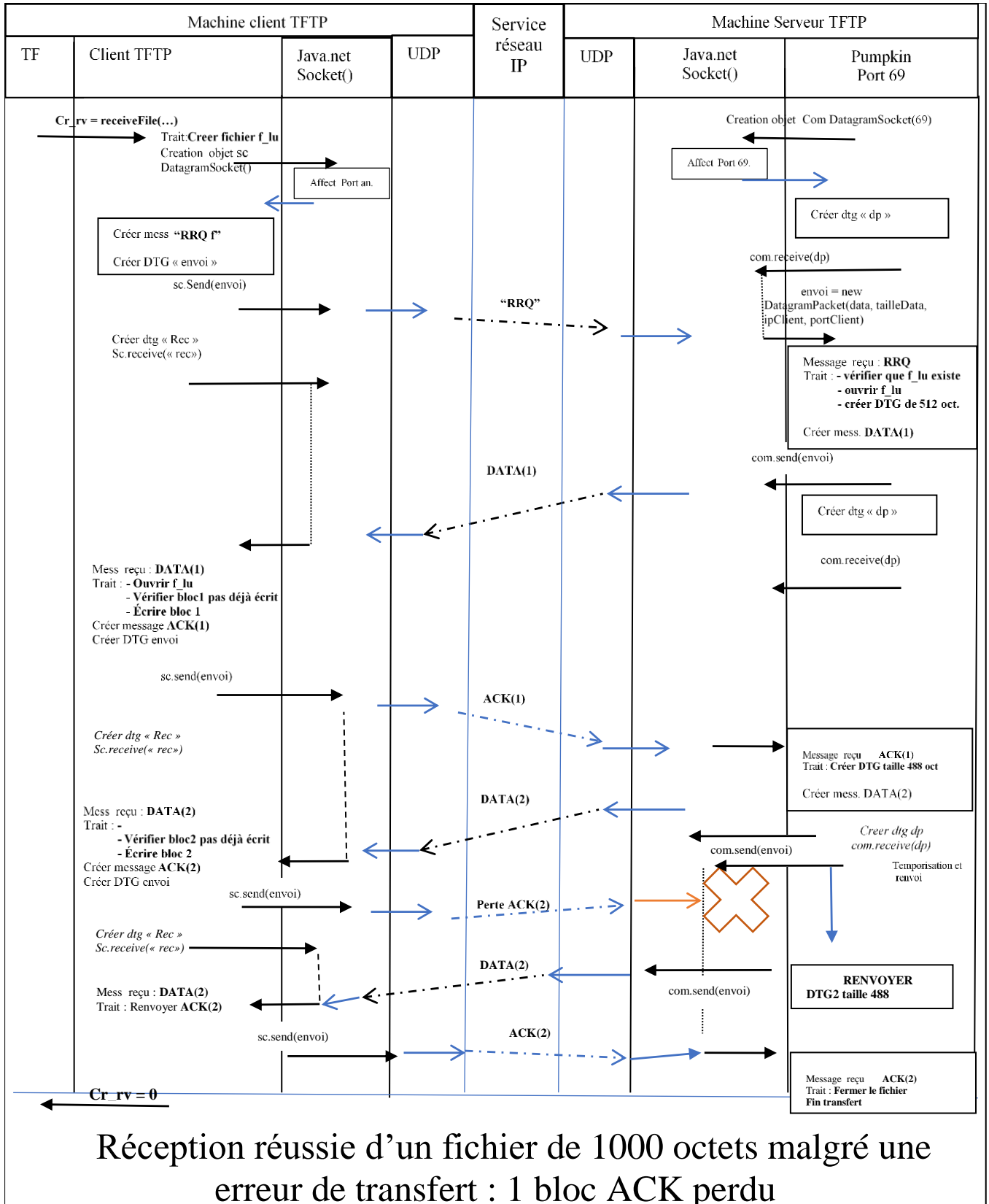
BENALI Myriam, BESSON Cécile, DELPLANQUE Thibaut,  
NAAJI Dorian INFO 3A Groupe 2  
POLYTECH LYON

# Table des matières

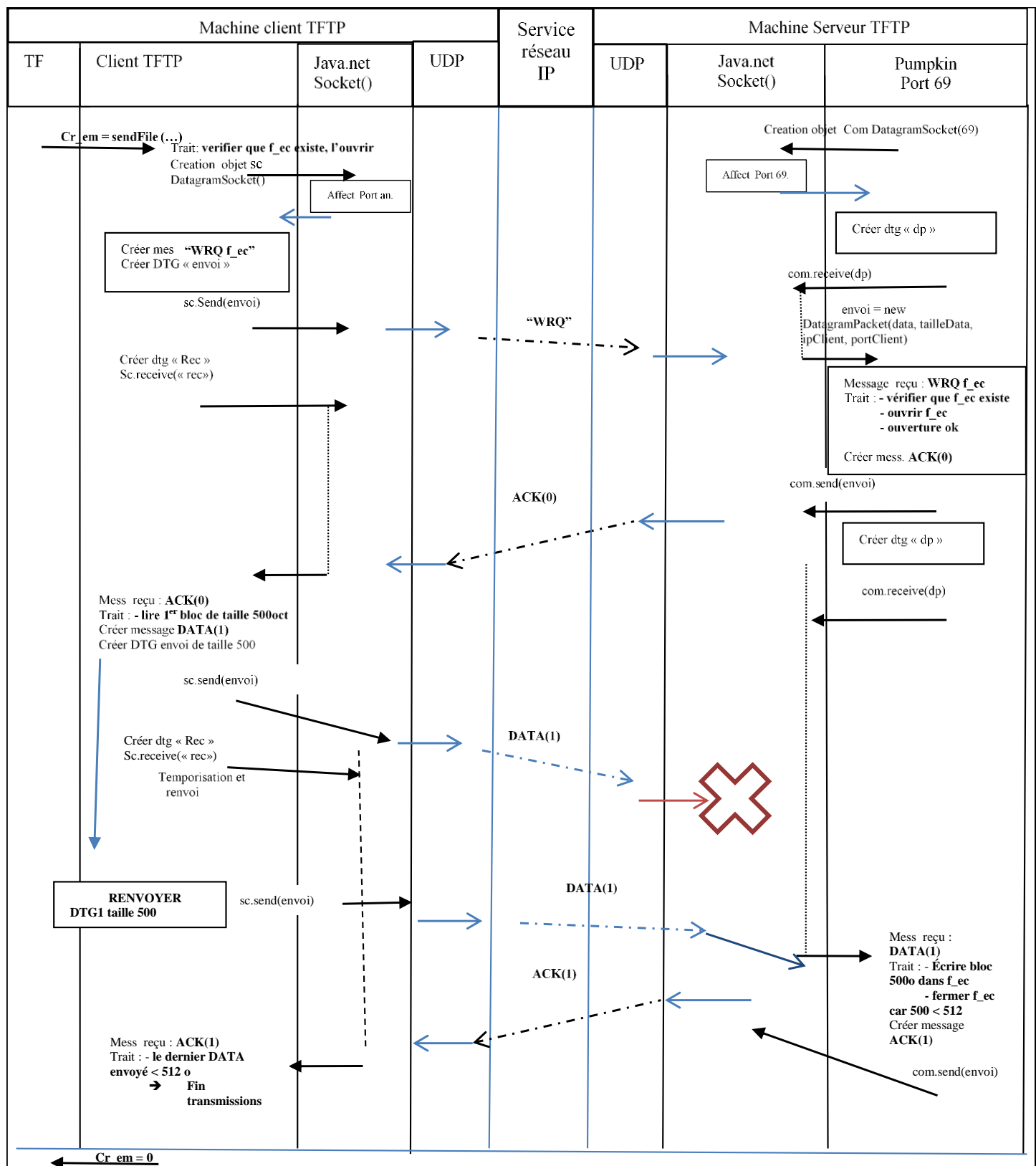
1.	Exemple d'échange de fichiers .....	3
1.1.	Réception_fichier() : Réception réussie d'un fichier de 1000 octets .....	3
1.2.	Envoi_fichier() : Envoi réussi d'un fichier de 500 octets.....	4
2.	Algorithmes des primitives en pseudocode .	5
2.1.	SendFile() .....	5
2.2.	ReceiveFile().....	8
3.	Manuel utilisateur .....	9

## 1.Exemple d'échange de fichiers

### 1.1. Réception\_fichier() : Réception réussie d'un fichier de 1000 octets



### 1.2. Envoi\_fichier() : Envoi réussi d'un fichier de 500 octets



Envoi réussi d'un fichier de 500 octets malgré une erreur de transfert : 1 bloc DATA perdu

## 2.Algorithmes des primitives en pseudocode

### 2.1. SendFile()

```
FUNCTION SendFile(adr_IP_serv, port_serv, nom_fichier_local) :
INTEGER
BEGIN
    CALL FileInputStream(nom_fichier_local) RETURNING file_stream
EXCEPTION
    WHEN IOException
        CALL println("Erreur de lecture de fichier")
        RETURN -1
END
ByteArrayOutputStream imageStream := CALL ByteArrayOutputStream()
BufferedImage image := null
BYTE[] imageBytes;

BEGIN
    BOOLEAN isFileAnImg := CALL isImg(nom_fichier_local)
EXCEPTION
    WHEN UnknowFileFormatException
        CALL println("Le format de fichier est inconnu")
        RETURN -3
END

IF (isFileAnImg) THEN
    BEGIN
        File f := CALL File(nom_fichier_local) e
        image := CALL ImageIO.read(f)
        imageExtension := extension OF file f
        CALL ImageIO.write(image, imageExtension, imageStream)
    EXCEPTION
        WHEN IOException
            CALL println("Erreur de lecture de fichier")
            RETURN -1
    END
ENDIF

BYTE WRQ[] := Nouveau paquet WRQ(OP_CODE_WRITE, nom_fichier_local,
"octet")
BEGIN
    DatagramSocket ds := CALL DatagramSocket()
    CALL ds.setSoTimeout(30000)
    DatagramPacket dpWRQ := CALL DatagramPacket(WRQ, WRQ.length,
adr_IP_serv, port_serv)
    CALL ds.send(dpWRQ)

    BYTE[100] firstServerResponse
    DatagramPacket rep := CALL DatagramPacket(firstServerResponse,
firstServerResponse.length)
    ds.receive(rep)
    IF( (OPCODE OF firstServerResponse = 4) AND (PACKET_NO OF
firstServerResponse == 0) ) THEN
        INTEGER eof := 0
        INTEGER idblock := 1
        BYTE[100] serverResponse
```

```

BYTE[] packet
imageBytes := CALL imageStream.toByteArray()
LONGINTEGER nbBlocsTailleMax := imageBytes.length / 512
INTEGER tailleDernierBloc := imageBytes.length mod 512
INTEGER nbBlocsTailleMaxFaits := 0

REPEAT
  IF (isFileAnImg) THEN
    BYTE[] rawData
    IF(nbBlocsTailleMaxFaits < nbBlocsTailleMax) THEN
      rawData := CALL BYTE[512]
      INTEGER j := 0
      FOR INTEGER i = nbBlocsTailleMaxFaits * 512 TO
((nbBlocsTailleMaxFaits + 1) * 512) - 1
        rawData[j] := imageBytes[i]
        j := j + 1
      ENDFOR
      nbBlocsTailleMaxFaits := nbBlocsTailleMaxFaits + 1
    ELSE
      rawData := CALL BYTE[tailleDernierBloc]
      FOR INTEGER i = nbBlocsTailleMaxFaits * 512 TO
(imageBytes.length) - 1
        rawData[j] := imageBytes[i]
        j := j + 1
      ENDFOR
      eof := -1
    ENDIF
    packet := Nouveau paquet DATA(idblock, rawData)
  ELSE
    ArrayList<Byte> dataList
    INTEGER byteCourant
    FOR INTEGER i = 0 to 511
      byteCourant := CALL file_stream.read()
      IF (byteCourant = -1)
        eof := -1
        break
      ELSE IF ( (byteCourant > -128) AND (byteCourant < 128))
        CALL dataList.add((BYTE) byteCourant)
      ELSE
        RETURN -1
      ENDIF
    ENDFOR
    BYTE[] rawData := CALL ByteArrayList_To_ByteArray(dataList)
    packet := Nouveau paquet DATA(idblock, rawData)
  ENDIF

  DatagramPacket dpData := CALL DatagramPacket(packet,
packet.length, adr_IP_serv, CALL rep.getPort())
  ds.send(dpData)

  DatagramPacket serverResponseData := CALL
DatagramPacket(serverResponse, serverResponse.length)
  ds.receive(serverResponseData)

  IF( (OPCODE OF serverResponse = 5)) THEN
    CALL println("Operations resulted in a server-side error")
    CALL file_stream.close()
    CALL imageStream.close()
    CALL ds.close()
    RETURN 1
  
```

```

        ENDIF
        idBlock := idBlock + 1
    UNTIL (eof = -1)
    CALL ds.close()
ELSE
    CALL println("Operations resulted in a server-side error")
    CALL file_stream.close()
    CALL imageStream.close()
    CALL ds.close()
    RETURN 1
ENDIF
EXCEPTION
    WHEN SocketException
        CALL file_stream.close()
        CALL imageStream.close()
        RETURN -2
    WHEN UnknowFileFormatException
END
CALL file_stream.close()
CALL imageStream.close()
RETURN 0
ENDFUNCTION

```

## 2.2. ReceiveFile()

```
*CC = Chaîne de Caractères

FUNCTION receiveFile(CC adrIp, INTEGER port, CC fileName) : INTEGER

    BEGIN
        Création d'un fichier avec comme nom, fileName
        Ouverture d'un socket avec le serveur
        Envoi d'un datagramme avec l'ordre RRQ
    EXCEPTION
        WHEN Erreur création/ouverture du fichier
            Libère les ressources réservées au fichier
            RETURN -1
        WHEN Erreur avec le socket
            Libère les ressources réservées au fichier
            Ferme le socket
            RETURN -2
    END

    INTEGER N := 1;

    DO
        Attente de réception d'un paquet
        IF Réception d'un paquet ERROR
            Libère les ressources réservées au fichier
            Ferme le socket
            RETURN 1
        ELSE
            IF numéro paquet = N
                Ecriture du paquet (donnée) dans le fichier
                Envoie de ACK n°N
                Incréments N
            ELSE
                Envoie de ACK n°N - 1
            END IF
        END
    WHILE La taille de la donnée est égal à 512 octets

    DO
        Attendre de réception d'un paquet pendant un temps
        IF réception d'un paquet
            renvoi de ACK n° N - 1
        ELSE
            vérification terminée
        END IF
    WHILE vérification non terminée

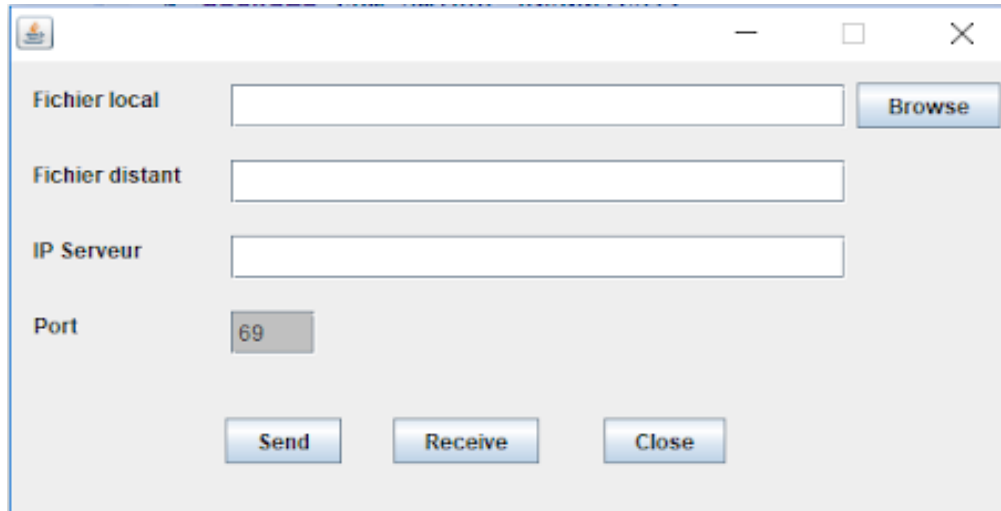
    RETURN 0

ENDFUNCTION
```



### 3.Manuel utilisateur

Notre interface ressemble à ceci :

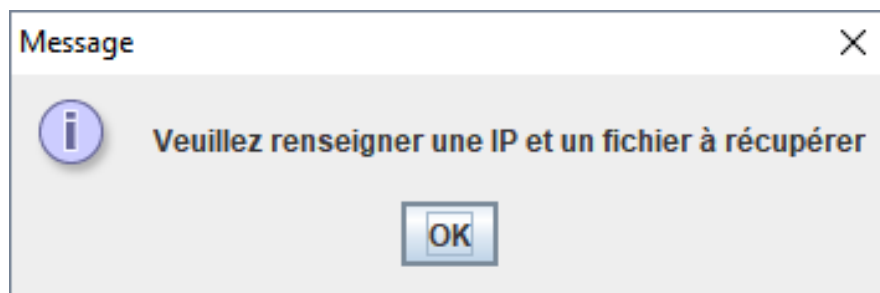


The screenshot shows a window with a title bar containing a small icon and standard minimize, maximize, and close buttons. The main area contains four labeled input fields: 'Fichier local' with a 'Browse' button to its right, 'Fichier distant', 'IP Serveur', and 'Port' (which has the value '69' entered). At the bottom of the window are three buttons: 'Send', 'Receive', and 'Close'.

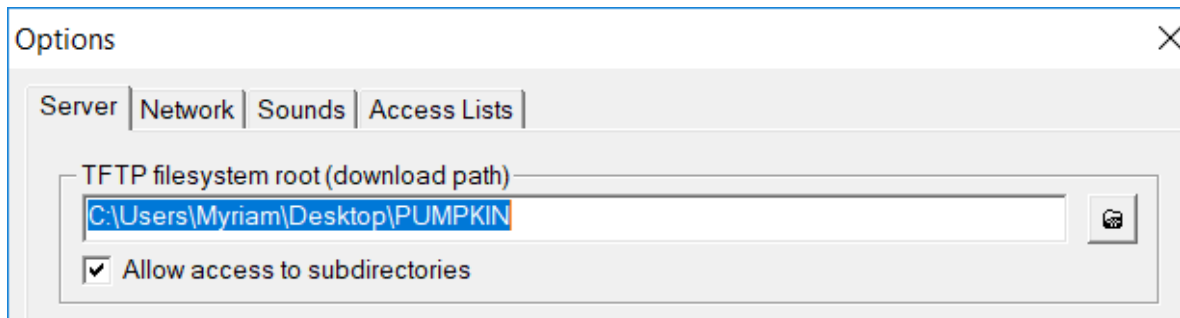
#### **Concernant l'envoi d'un fichier au serveur :**

Pour l'envoi d'un fichier au serveur, nous devons, tout d'abord, renseigner les champs "**Fichier local**" (grâce au bouton Browse, nous pouvons directement accéder aux documents contenus dans notre PC et donc choisir un fichier) et "**IP Serveur**" (qui correspond à l'adresse IP du serveur Pumpkin). Une fois les deux champs remplis, nous pouvons cliquer sur "**Send**" pour envoyer notre fichier.

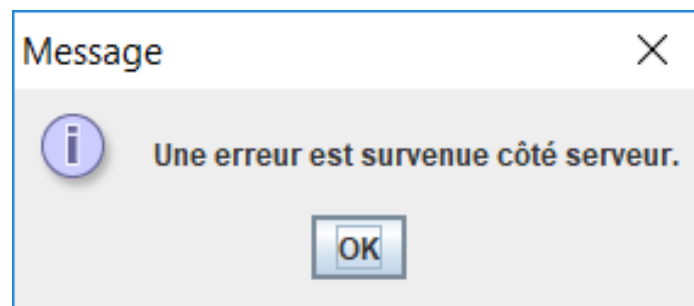
Si nous cliquons sur "Send" sans remplir les champs "Fichier local" et "IP Serveur", nous obtenons le message d'erreur suivant :



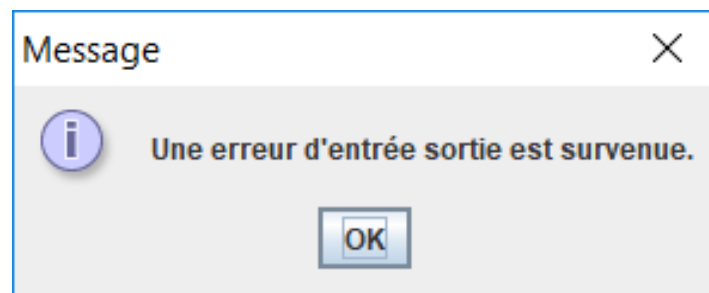
La réception des fichiers par le serveur se fera au niveau du chemin mentionné dans les options de Pumpkin :



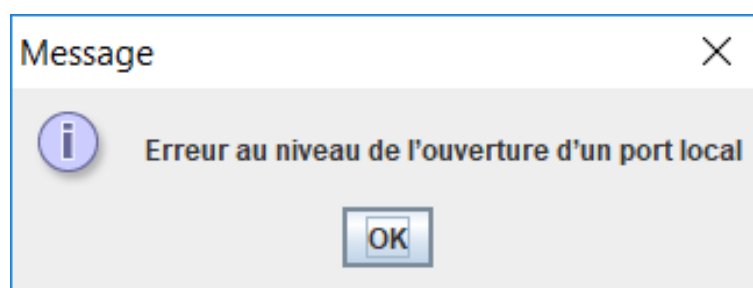
- Il y a différents cas de figure selon la valeur de CrEm :
- Si CrEm = 0 : l'envoi du fichier s'est bien déroulé
- Si CrEm = 1 : il y a une erreur au niveau du serveur Pumpkin



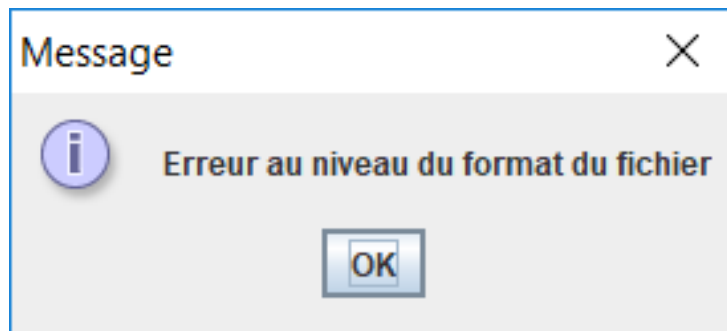
- Si CrEm = -1 : il y a une erreur au niveau de l'ouverture du fichier local (IOException)



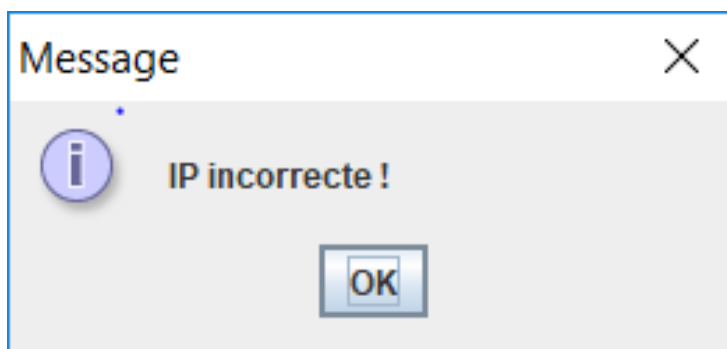
- Si CrEm = -2 : il y a une erreur au niveau de l'ouverture d'un port local (SocketException)



- Si CrEm = -3 : il y a une erreur au niveau du format du fichier (format inconnu) (UnknownFileFormatException)

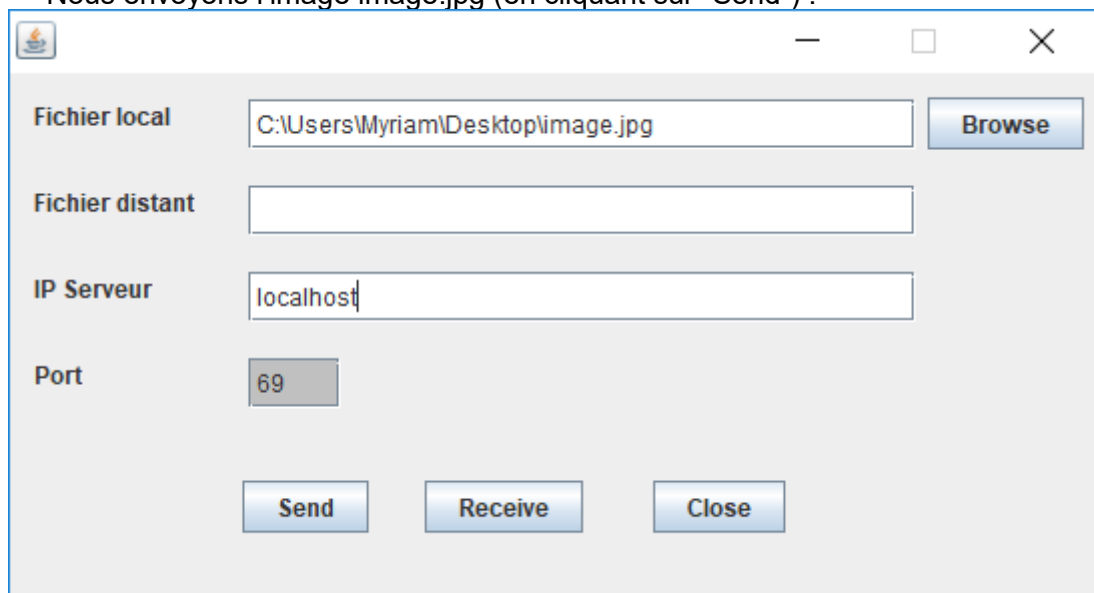


- Si CrRV = -4 : il y a une erreur au niveau de l'adresse IP (UnknownHostException)

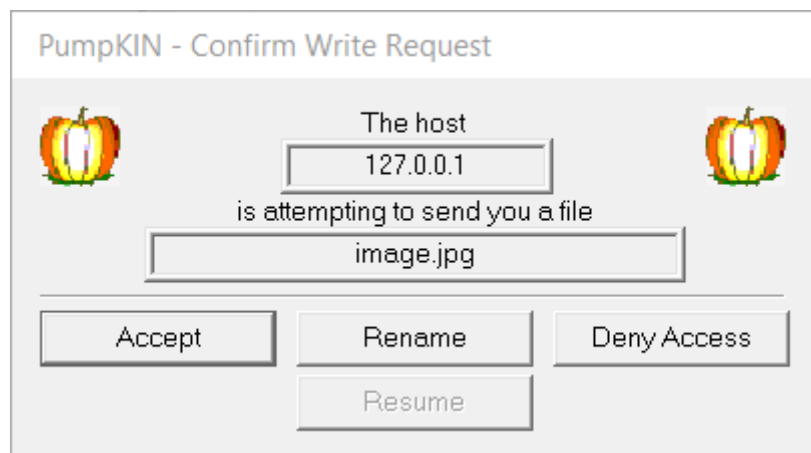


Exemple d'un envoi d'une image au serveur :

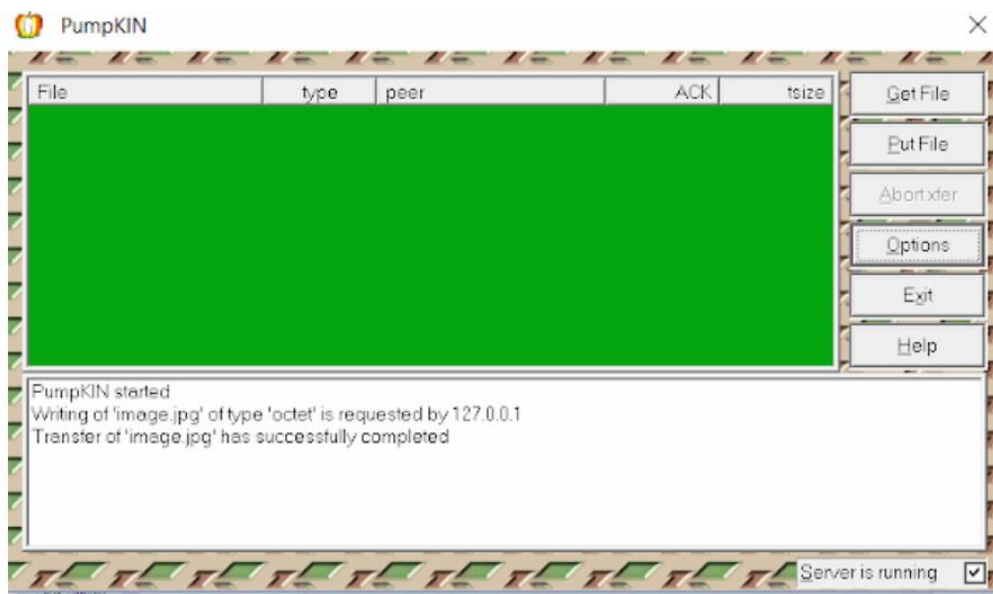
Nous envoyons l'image image.jpg (en cliquant sur "Send") :



Nous acceptons la réception de l'image au niveau du serveur PumpKIN :



Le transfert de l'image s'est bien déroulé :



Pour finir, nous trouvons bien l'image image.jpg dans le dossier PUMPKIN dans le bureau :



image

I

### ***Concernant la réception d'un fichier du serveur :***

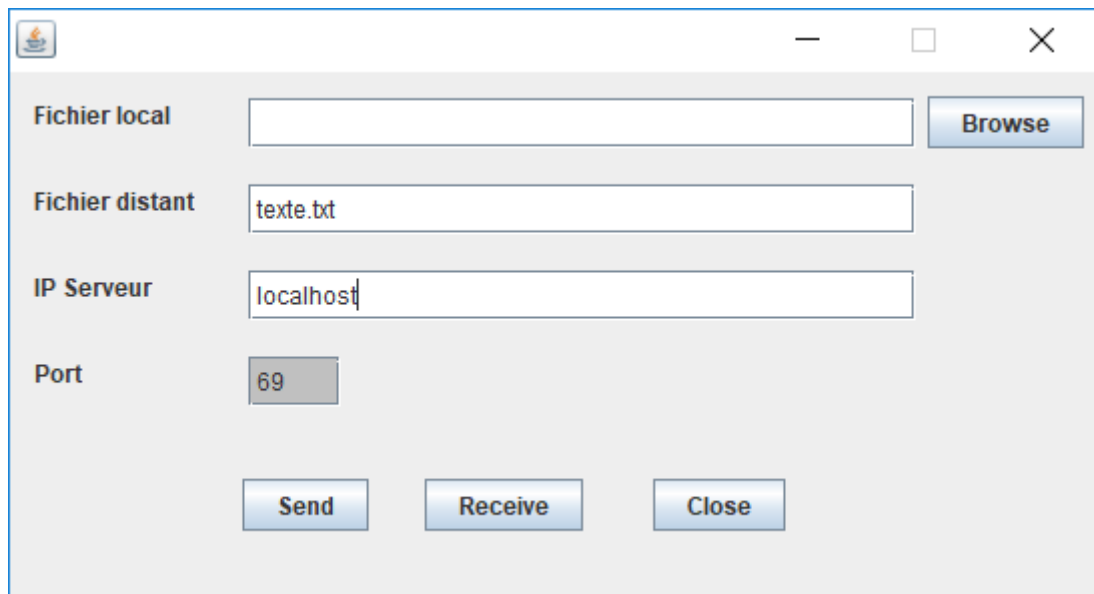
Pour la réception d'un fichier du serveur, nous devons, tout d'abord, renseigner les champs "**Fichier distant**" (correspondant au nom du fichier que l'on veut récupérer du serveur) et "**IP Serveur**". Une fois les deux champs remplis, nous pouvons cliquer sur "**Receive**" pour recevoir le fichier.

Il y a différents cas de figure selon la valeur de CrRv :

- Si CrRv = 0 : l'envoi du fichier s'est bien déroulé
- Si CrRv = 1 : il y a une erreur au niveau du serveur Pumpkin (ServerSideException)
- Si CrRv = -1 : il y a une erreur au niveau de l'ouverture du fichier local (IOException)
- Si CrRv = -2 : il y a une erreur au niveau de l'ouverture d'un port local (SocketException)
- Si CrRv = -3 : il y a une erreur au niveau du format du fichier (format inconnu) (UnknownFileFormatException)
- Si CrRV = -4 : il y a une erreur au niveau de l'adresse IP (UnknownHostException)

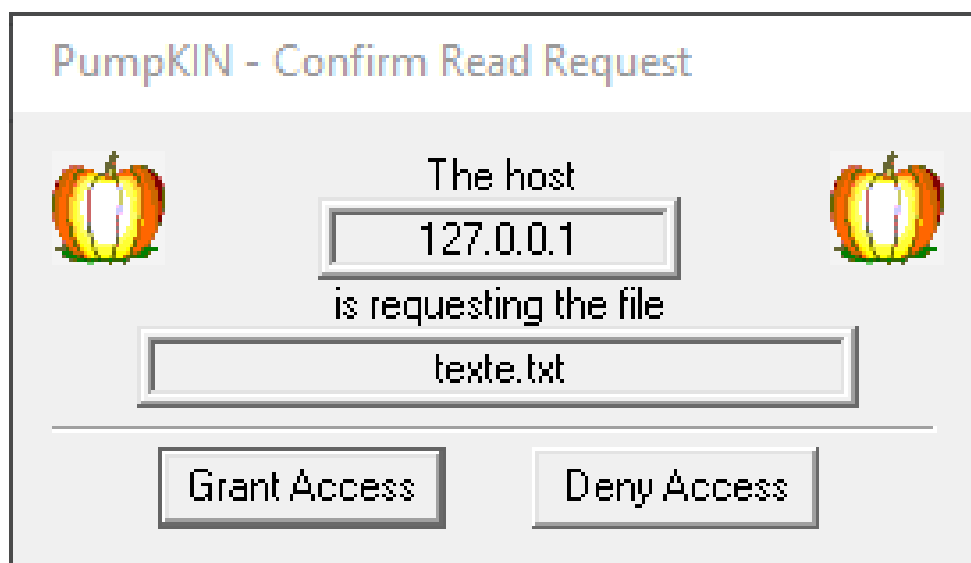
### Exemple de la réception d'un texte texte.txt du serveur :

Nous notons le nom du fichier distant (texte.txt) :



The screenshot shows a window titled 'PumpKIN' with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are four labeled input fields: 'Fichier local' (empty), 'Fichier distant' (containing 'texte.txt'), 'IP Serveur' (containing 'localhost'), and 'Port' (containing '69'). To the right of the 'Fichier local' field is a 'Browse' button. At the bottom of the window are three buttons: 'Send', 'Receive', and 'Close'.

PumpKIN demande alors une confirmation concernant le RRQ s'il a été configuré pour afficher les requêtes entrantes. Il faut alors que le serveur accepte l'accès au fichier pour démarrer la transmission.



Une fois que l'accès est accordé par PumpKIN, notre programme JAVA est autorisé à récupérer le fichier sur le serveur, s'en suit alors une série d'échanges de paquets. Le fichier est finalement placé dans le dossier depuis lequel est exécuté l'application JAVA.