Modélisation et Implémentation des Systèmes Complexes
Agents mobiles

A. Cardon, C. Bertelle et D. Olivier
LIH - Laboratoire d'Informatique du Havre

DEA Informatique Théorique et Applications
Ecole Doctorale SPMI - Rouen/Le Havre

Année 2003/2004

5 Les Agents mobiles

5.1 Agents distribués

5.2 Code mobile

5.3 Exemples

5.1 Agents distribués

Agents coopératifs, non mobiles. Le placement est statique, les agents s'éxécutent sur la même machine jusqu'à leur mort.

- Une plateforme (répartie) avec des agents qui coopérent grace aux communications (distantes) ;
- Plusieurs plateformes homogènes ;
- Plusieurs plateformes hétérogènes $\Rightarrow$ interopérabilité avec communication entre agents.

Il faut faire attention, nous nous plaçons dans un contexte d"'agent intelligent". L'agent est alors considéré comme (au moins) une entité autonome, sociale et réactive et pro-active.

- Autonomie : capacité de l'agent à fonctionner sans intervention directe d'un tiers et il contrôle ses propres actions et son état interne ;
- Socialité : capacité de l'agent d'interagir avec les autres quand la situation l'exige, pour compléter ses tâches ou aider les autres ;
- Réactivité : capacité de l'agent à percevoir son environnement et interagit avec lui ;
- Pro-activité : capacité de l'agent sur sa propre initiative de se fixer des buts pour atteindre ses objectifs. Son comportement est opportuniste en fonction de son état interne sans intervention d'un tiers.

L'interactivité se fait au moins entre deux agents en :

– communiquant directement ;
– par l'intermédiaire d'un autre agent ;
– en agissant sur l'environnement.

La répartition peut avoir plusieurs formes :

– Une plate-forme (répartie) avec des agents qui coopérent grace aux communications (distantes). Dans ce cas nous pouvons avoir à faire à des mécanismes transparents au niveau de la plateforme.
– Plusieurs plate-formes homogènes ;
– Plusieurs plate-formes hétérogènes $\Rightarrow$ interopérabilité avec communication entre agents.

5.2 Code mobile

C'est la capacité de déplacer des composants logiciels d'un système distribué d'un nœud du système à un autre.
L'initiateur envoie s'exécuter ses procédures à l'endroit où se trouve les données.

5.2 Code mobile

Modéles d'exécution les paradigmes :

5.2.1 Code à la demande ;

5.2.2 Echange de message ;

5.2.3 Code référençable ;

5.2.4 Evaluation distante ;

5.2.5 Agents mobiles.

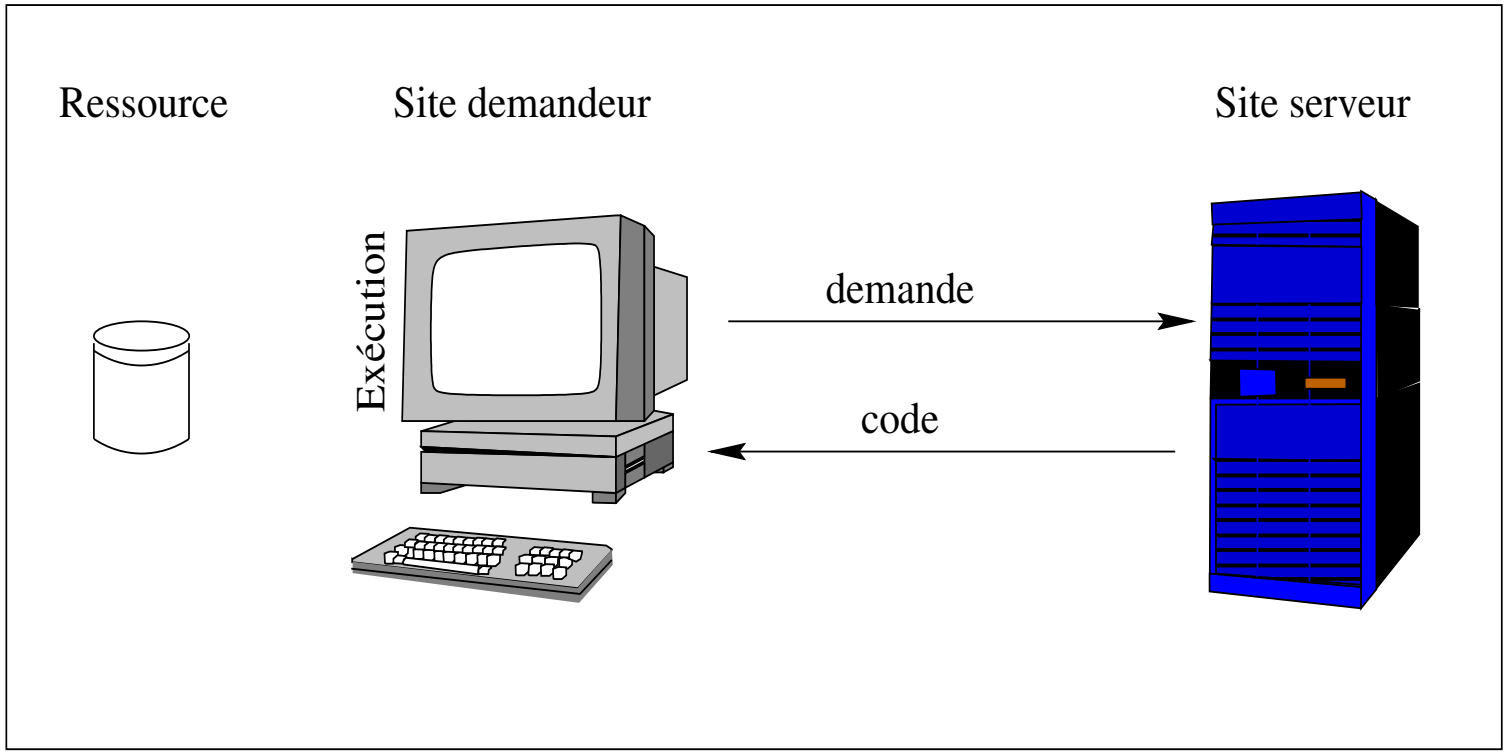
5.2.1 Code à la demande

Code on demand, cela ressemble à du téléchargement. Il y a deux sens possibles à ce téléchargment :

- De façon imposée par un nœud du système distribué à un autre nœud ;
- A la demande.

– De façon imposée par un nœud du système distribué à un autre nœud ; (ex : m.à.j, fichier Postscript !)
– A la demande. (m.à.j, applets Java)

5.2.1 Code à la demande



Le site recepteur reçoit le code du serveur. Il exécute alors le code localement.

5.2.1 Code à la demande

- Mécanisme asynchrone ;
- Gros grain de distribution ;
- Utilisable dans les réseaux de grande échelle.

5.2.1 Code à la demande

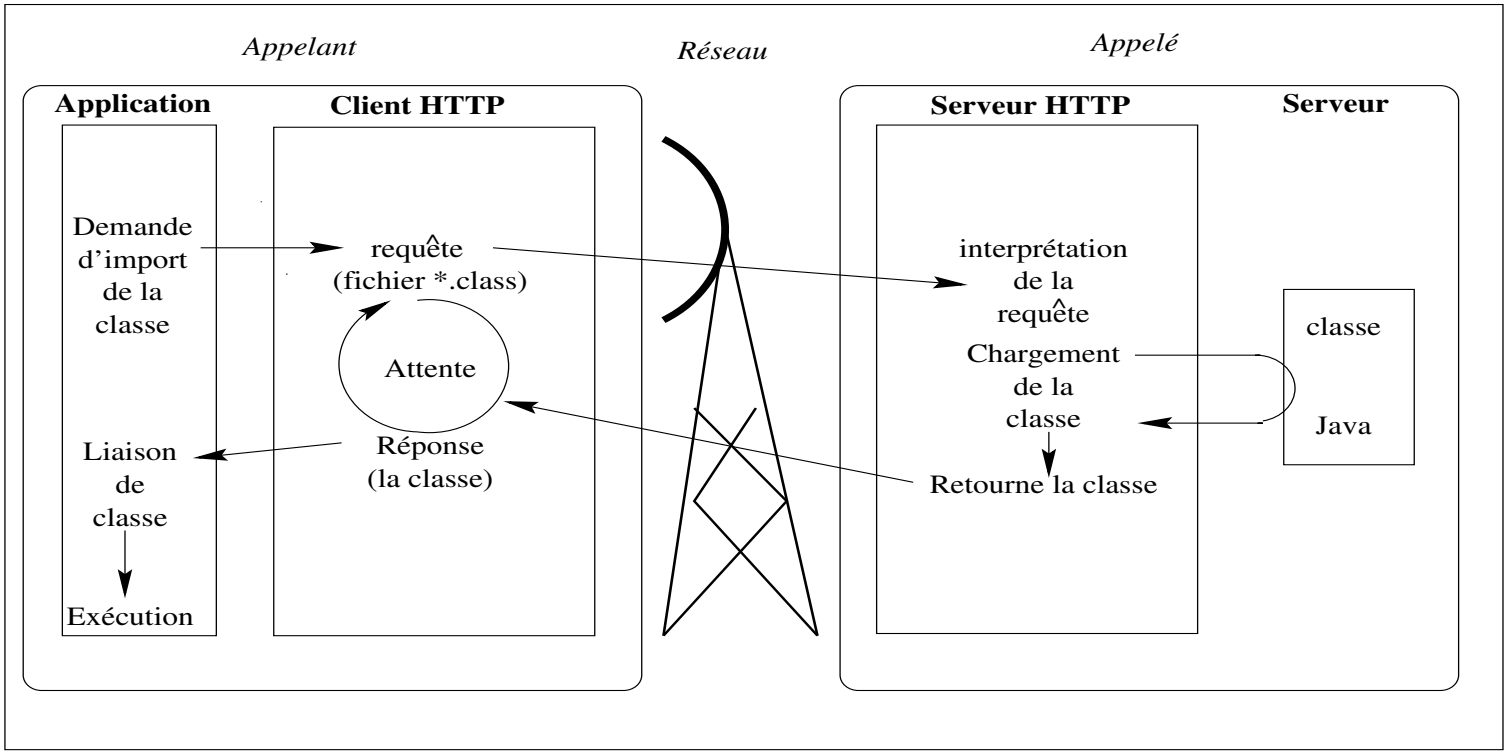


FIG. 1 – Fonctionnement d'une applet java

5.2.2 Passage de message

- Mécanisme souple et puissant ;
- Obligation de prendre en compte au niveau du programme de :
  - Les communications ;
  - La synchronisation.
- Sockets Unix ;
- Mécanisme de base dans Mach et Chorus (OS) ;
- Utilisé pour les applications parallèles ;
- S'appuie dans certain cas sur PVM et MPI (standard).

Attention dans ce cas les messages ne contiennent pas obligatoirement du code. Ils peuvent être constitués de données et eventuellement de code.

5.2.2 Passage de message

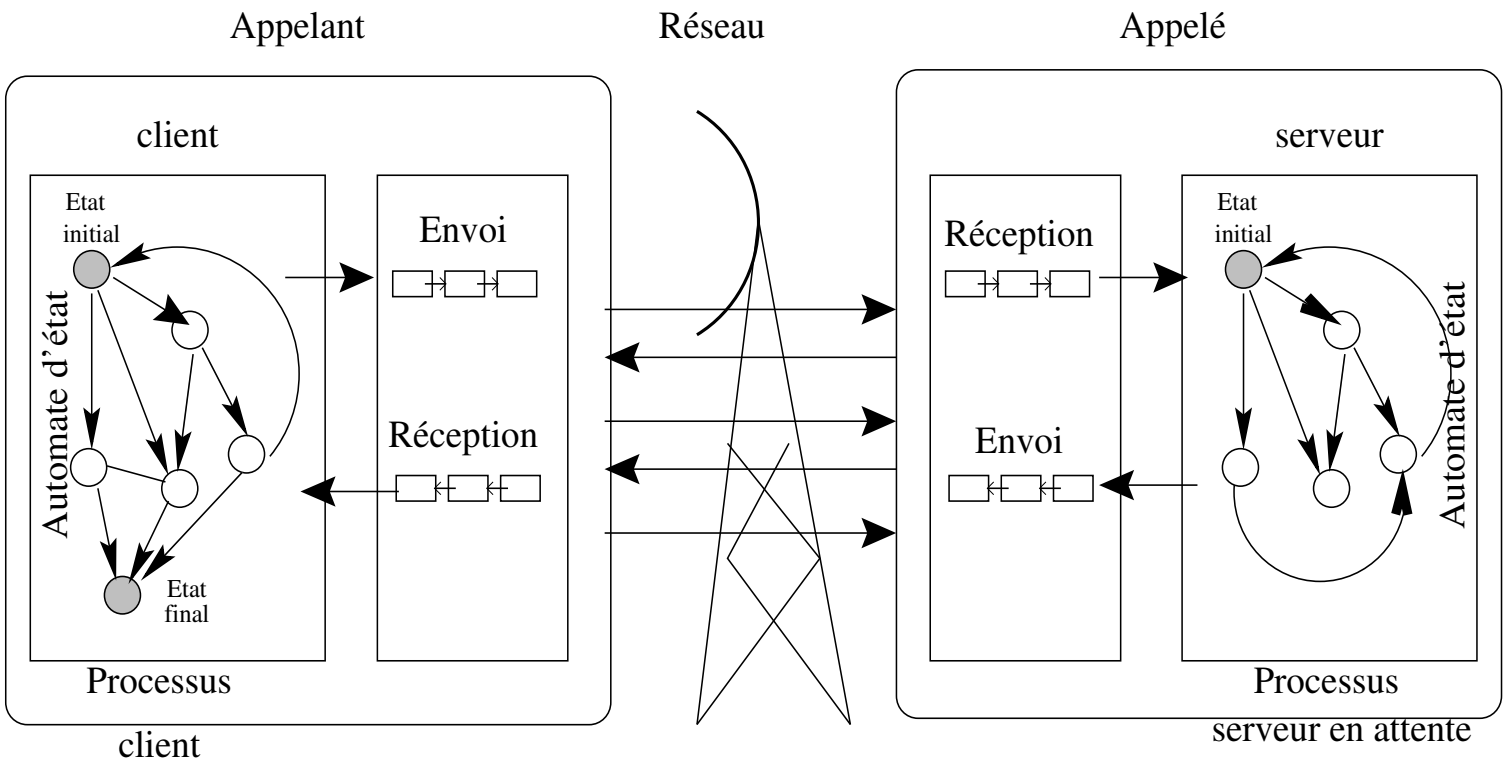


FIG. 2 – Echange de messages

5.2.3 Code référençable

- Appel de procédures à distance (RPC) ;
- Invocation d'objets à distance, objets distribués (RMI, CORBA) ;
- HTTP et CGI ;

5.2.3 Code référençable

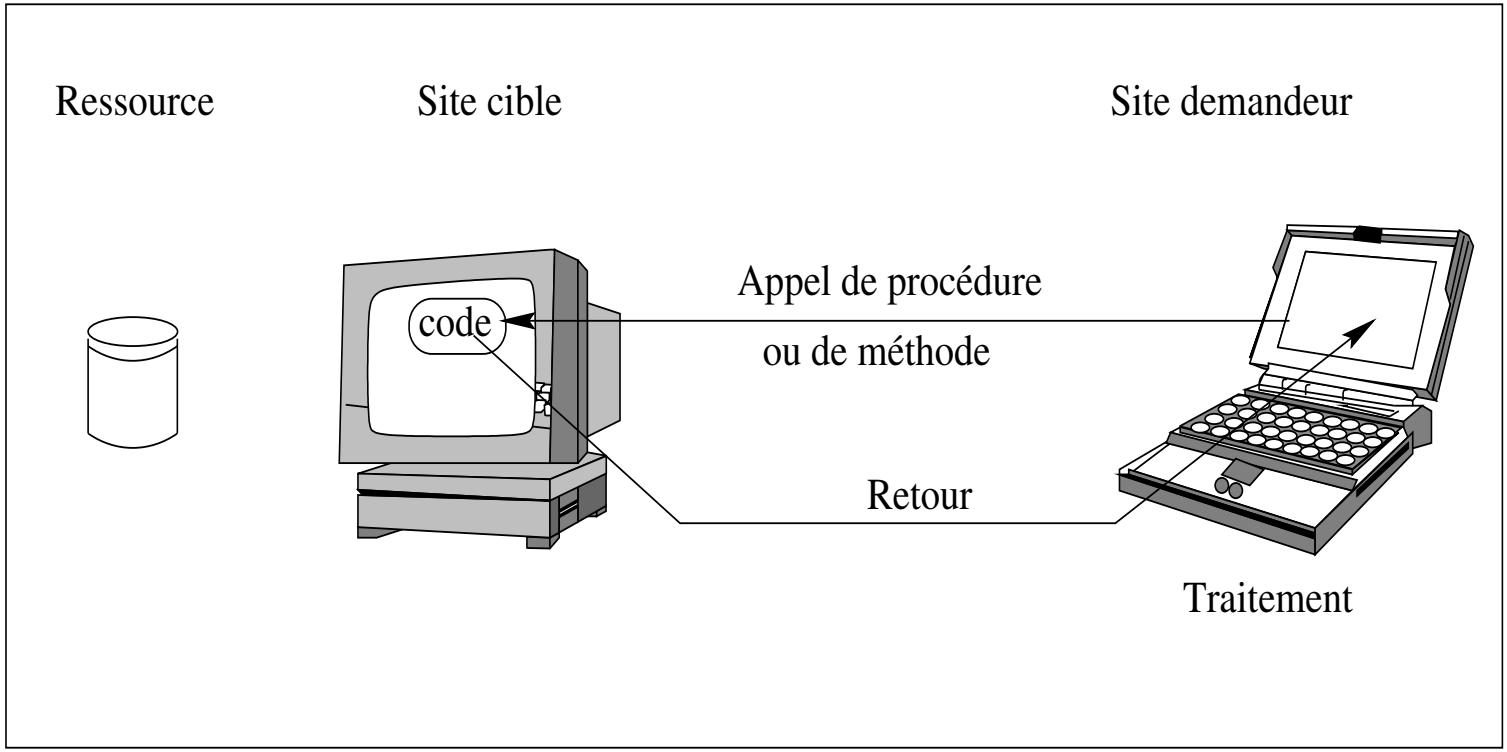


FIG. 3 – Principe du code référençable

5.2.3 Code référençable

Caractéristiques courantes

- Synchrone ;
- Grain fin ;
- Interface statique (le plus souvent) ;
- Réseaux locaux (le plus souvent).

5.2.3 Code référençable

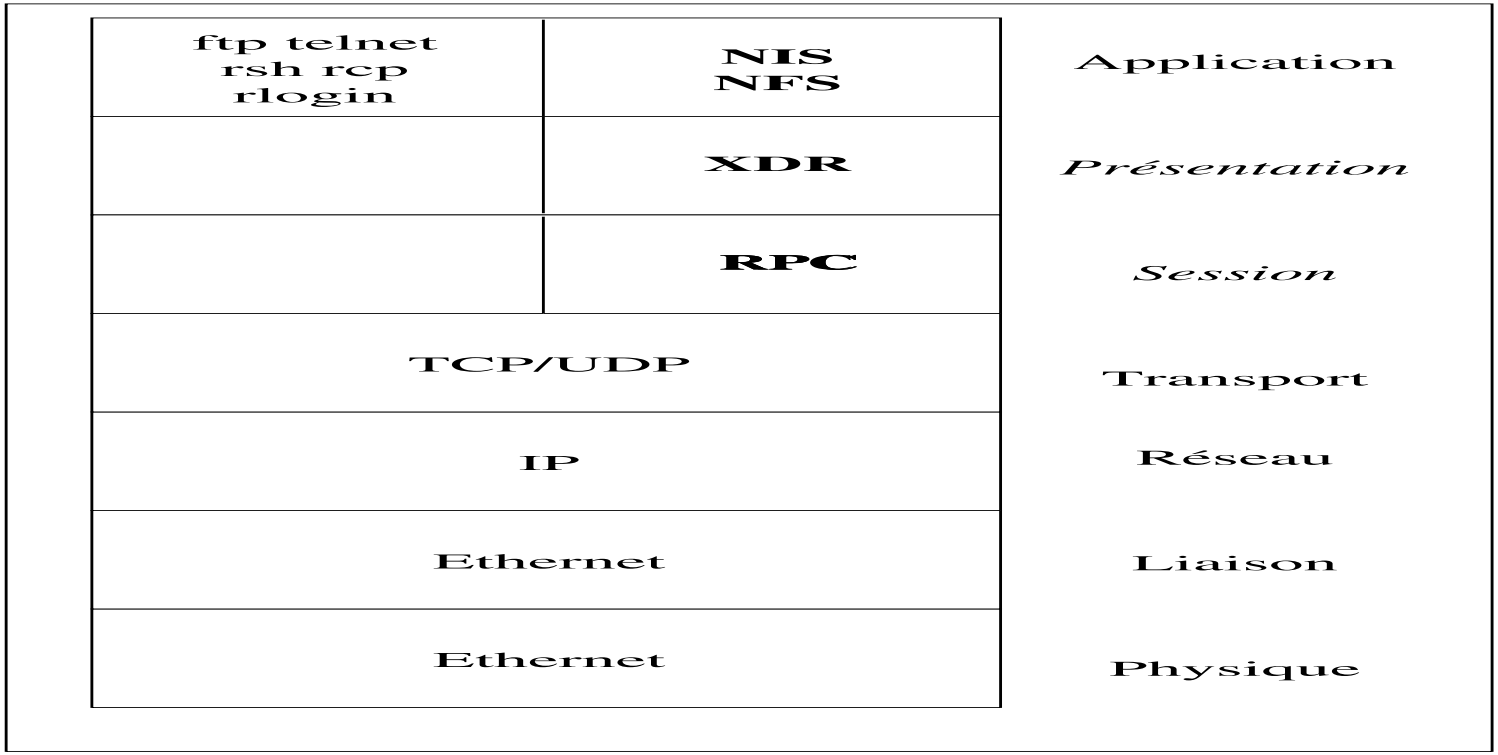


FIG. 4 – RPC dans le modèle OSI

5.2.3 Code référençable

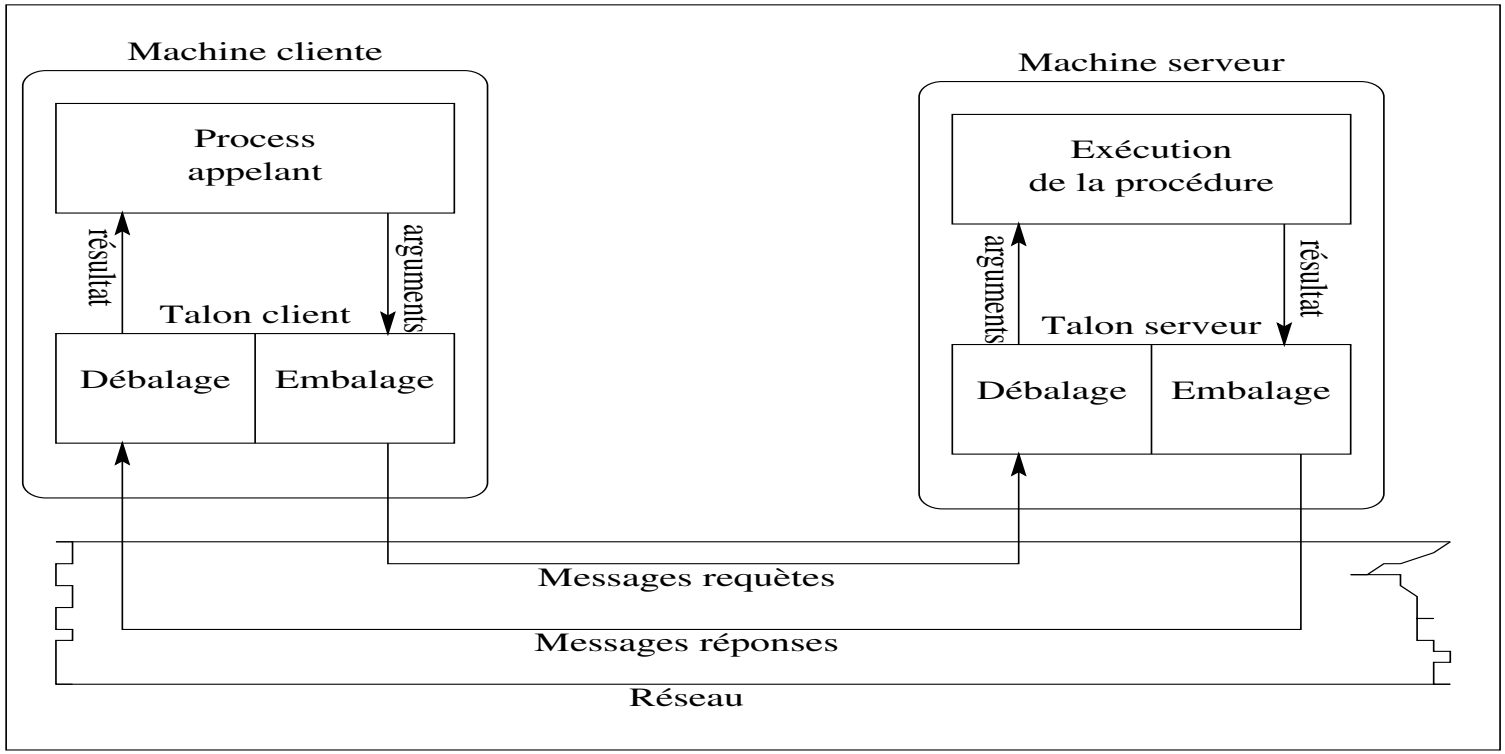


FIG. 5 – Appel de procédure à distance

5.2.3 Code référençable

Les “Remote Procedure Call”

- Permet à un serveur de définir une procédure qu’un programme sur un client peut appeler ;
- Chaque procédure admet des paramètres par valeur et retourne un résultat ;
- Les informations échangées doivent être représentées sous une forme standard (ex : XDR eXternal Data Representation).
- Ex : NFS.

5.2.3 Code référençable

Invocation d'objets à distance

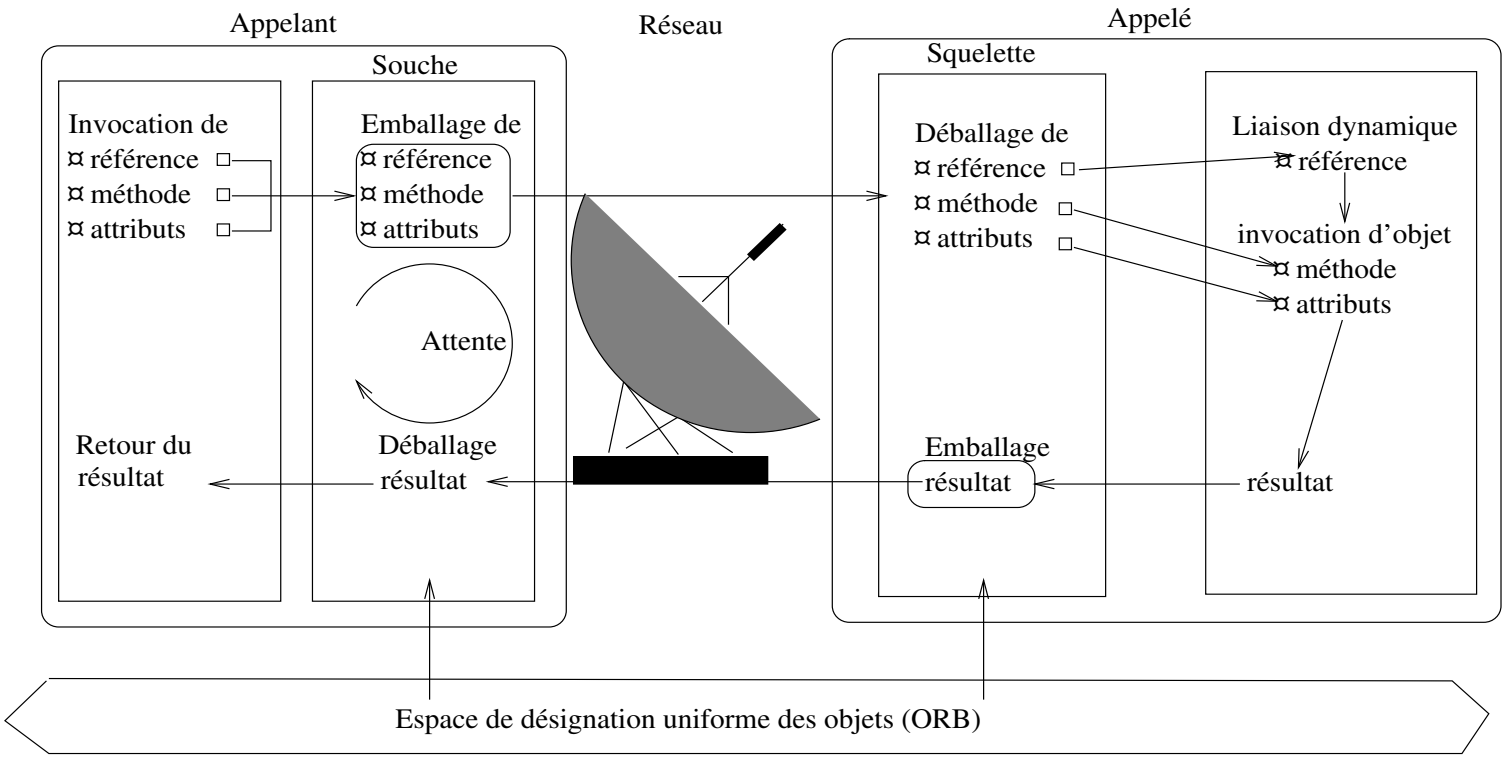


FIG. 6 – Objets distribués

L'ORB permet l'échange de requêtes entre les objets de façon transparente à la localisation et l'implémentation des objets. L'envoi d'une requête est réalisée par le stub, souche d'opération qui crée un message correspondant à cette requète, la reception du résultat est également assurée par le stub. Le squelette d'opération (skeleton) reçoit le message qui encode la requête et la transforme en appel d'implementation et récupère le résultat pour les transmettre à l'appelant par l'ORB.

5.2.3 Code référençable

Invocation d'objets à distance

- Mécanisme de courtage de requètes entre objets ;
- Mécanisme utilisé dans RMI, CORBA et COM/OLE ;
- Extension des RPC ;
- Construit sur un ORB.

5.2.3 Code référençable

HTTP et CGI

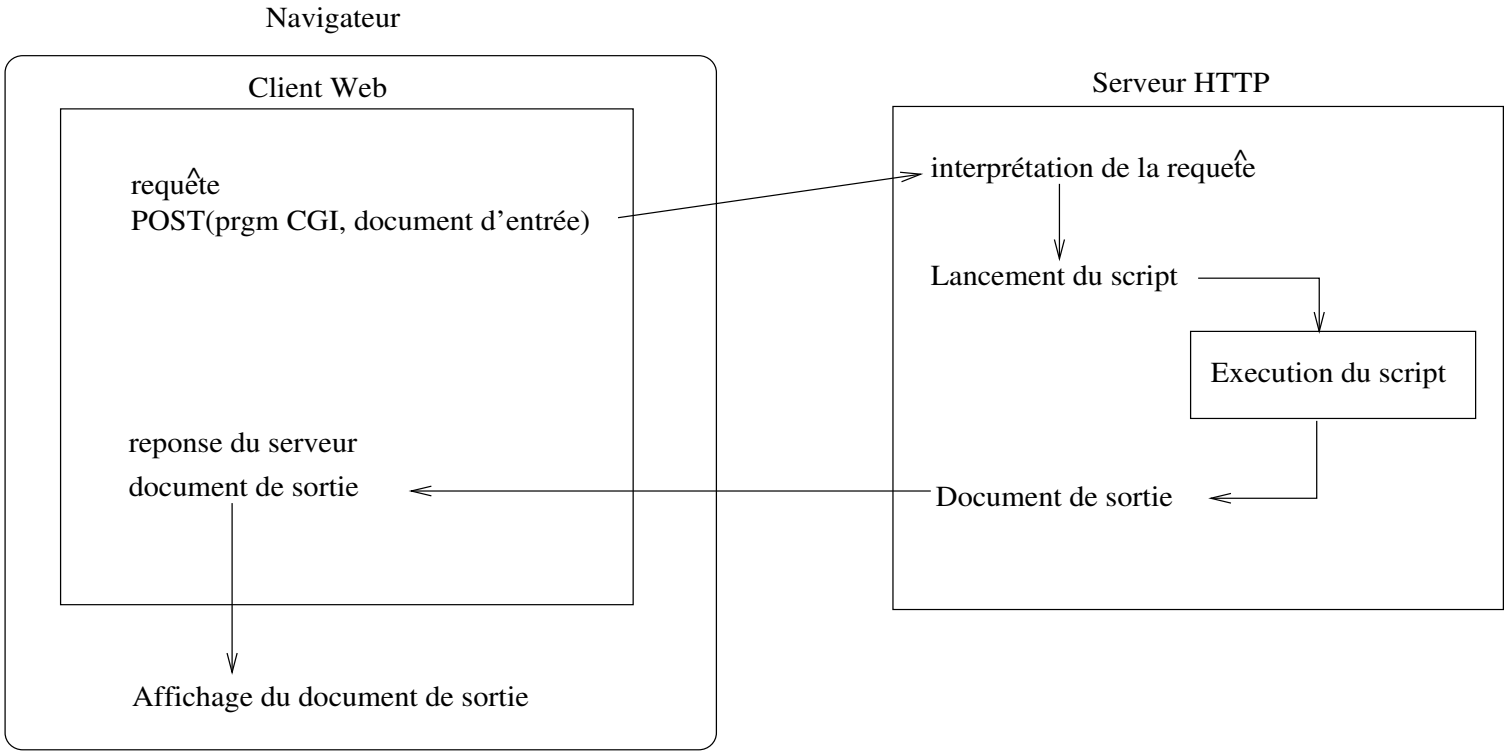


FIG. 7 – Protocole HTTP et script CGI

Le serveur en général écoute les requètes des clients sur le port 80 de TCP et il emet un document vers le navigateur de la façon suivante :

- Le navigateur effectue une connection TCP sur le port HTTP du serveur ;
- Le serveur accepte ;
- Le navigateur emet sa requête ;
- Le serveur interprète la requête ;
- Le serveur lance et execute le script CGI ;
- Le résultat du script est un document HTML qui est expédié au navigateur ;
- Le serveur coupe la connection ce qui signifie la fin du document ;
- Durant la reception le navigateur interprète le fichier HTML et l'affiche.

5.2.4 Evaluation distante

- Servlets
- Synchrone
- Gros grain de distribution, code et interface définis par le client ;
- Réseau de grande échelle (Internet).

5.2.4 Evaluation distante

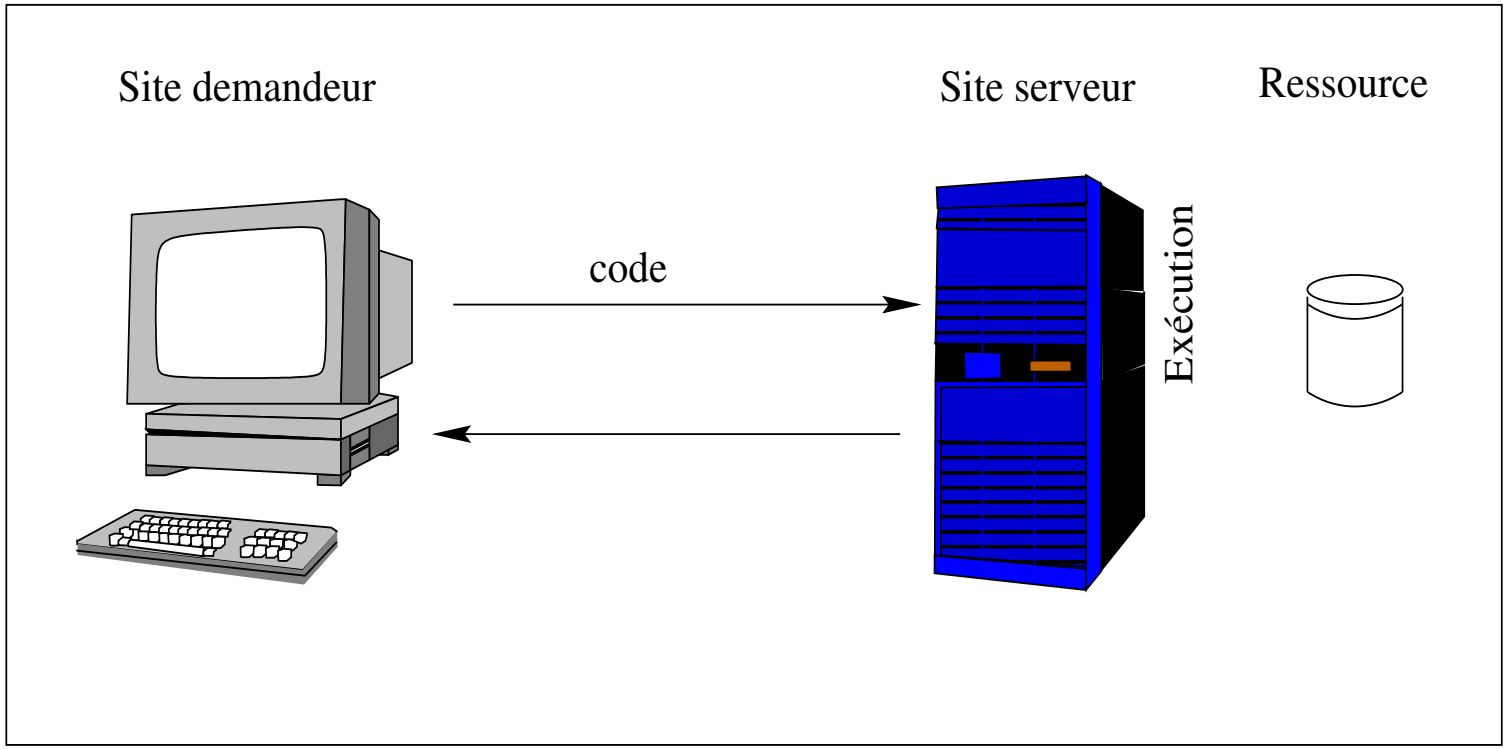


FIG. 8 – Principe de l'évaluation à distance

5.2.4 Evaluation distante

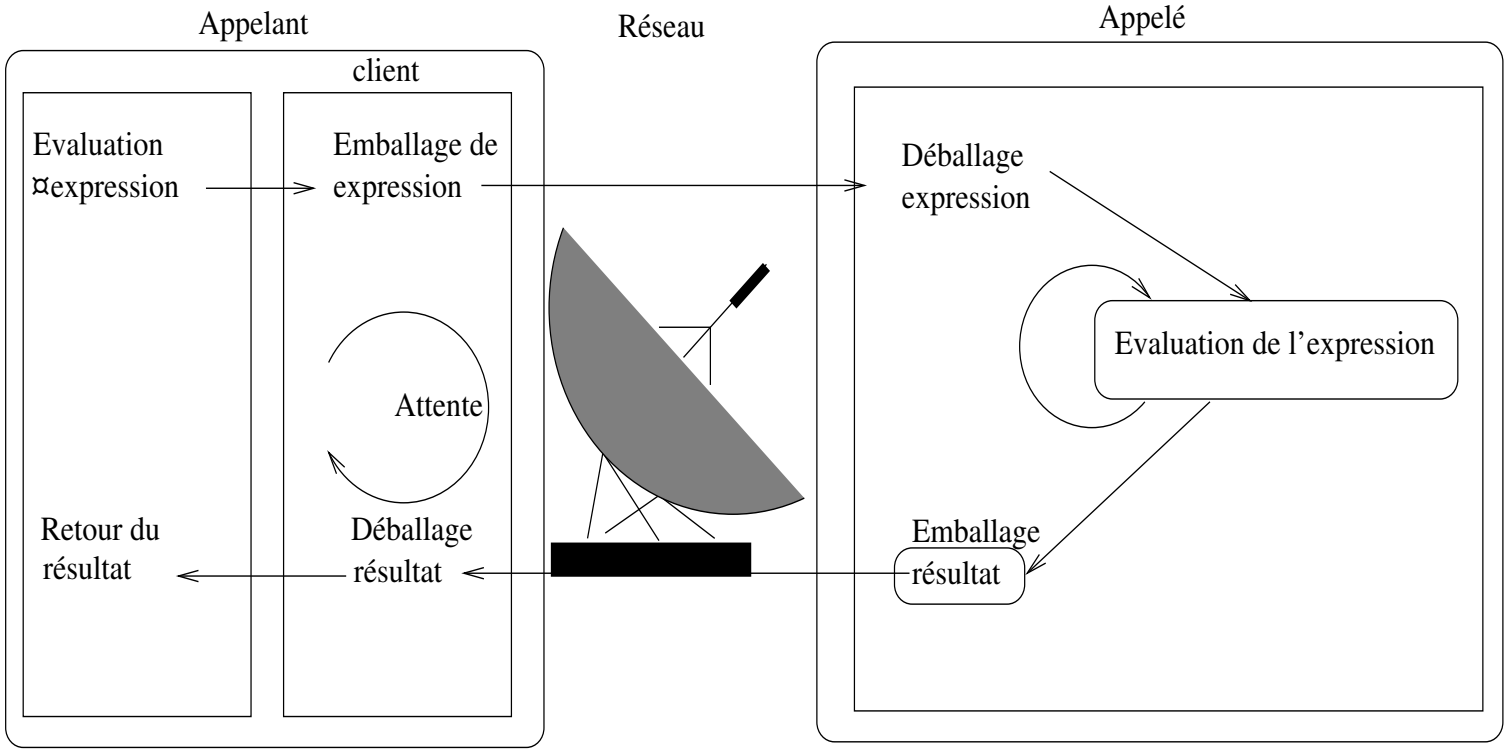


FIG. 9 – Servlets

CGI vs Servlets. Les programmes CGI ainsi que les servlets sont capables de générer des données dynamiques pour le client. Cependant les servlets peuvent :

- Continuer à s'exécuter en tâche de fond une fois le traitement de la requête fini, puis reprendre sur une nouvelle requète. De plus un servlet est capable de gérer des threads et donc traiter plusieurs requêtes simultanées.
- Interagir avec une applet.
- S'exécuter sur la machine cliente.
- Agent primitif ? ! !

5.2.5 Agents mobiles

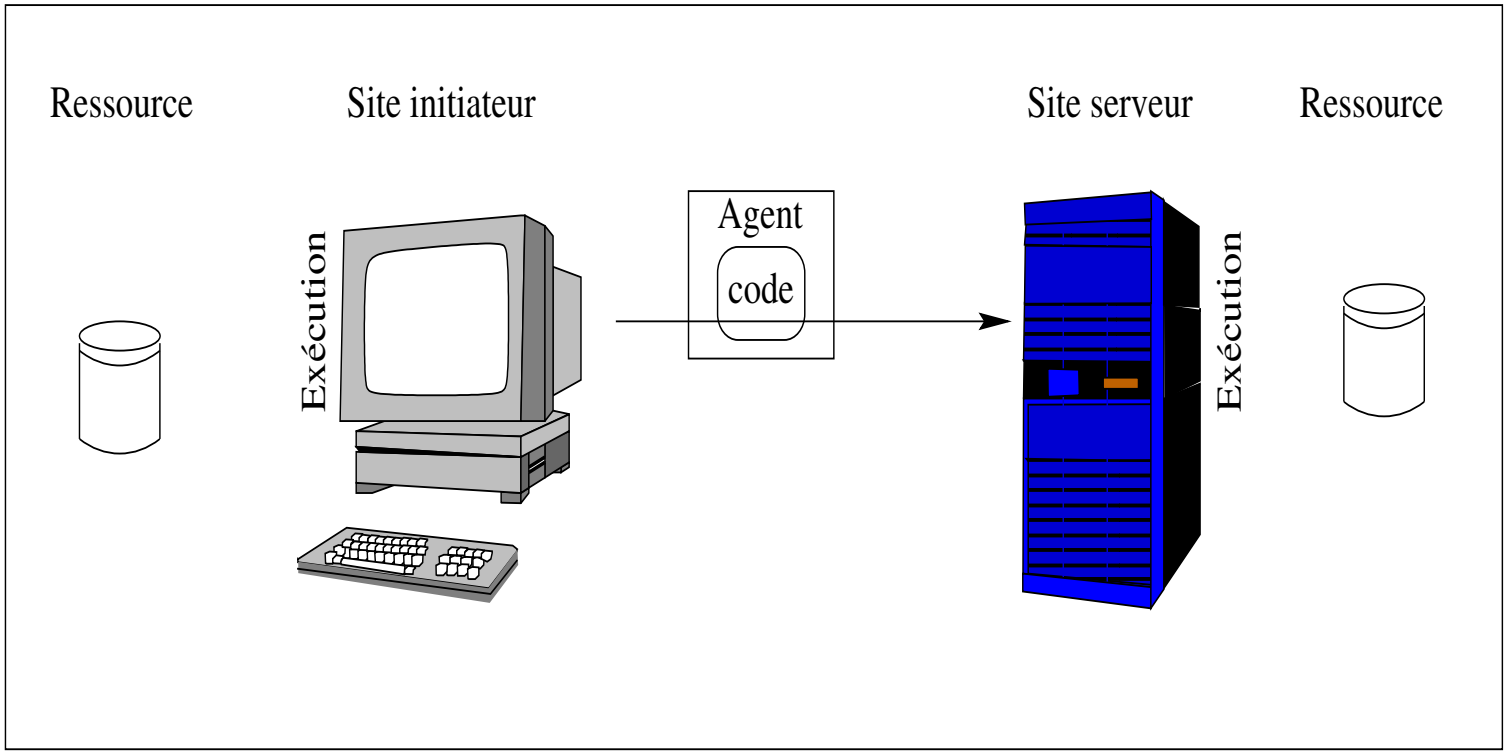


FIG. 10 – Principe des agents mobiles

5.2.5 Agents mobiles

Quel principe ?

1. L'exécution débute sur un site ;
2. Le code migre ainsi que ses données sur un site distant ;
3. Reprise de l'exécution du code.

5.2.5 Agents mobiles

Quels agents ?

- Agents mobiles
  - Unité d'execution **mobile autonome** possédant ses propres ressources.
- Agents "intelligents"
  - Système informatique, **situé** dans un environnement, et qui agit d'une façon **autonome** et **flexible** pour atteindre les objectifs pour lequel il a été conçu.

– Agents “intelligents” implémentés par des agents mobiles !

1. **Agent mobile** :
   – Agent : objet actif succeptible de communiquer potentiellement avec d'autres agents ;
   – Mobile : il se déplace en fonction de ses besoins et il peut suivre dans certains cas un itinéraire.

   Processus, incluant du code et des données, pouvant se déplacer entre des unités d'exécution pour réaliser une tâche.
2. **Agent "intelligent"** : l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement.
   – autonome : l'agent est capable d'agir sans intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne.
   – flexible : l'agent est
     – réactif ;

- proactif ;
- social.

3. **Agent “intelligent” mobile** 1 + 2

5.2.5 Agents mobiles

Les types d'agents mobiles :

- Agents de notification ;
- Agents itinerants ;
- Agents d'adaptation.

– Agents de notifications, ils attendent une information ou un événement et préviennent l'usager ou déclenchent une action (exemple : gestion d'un portefeuille d'actions, surveillance de connection ou d'activité sur une machine ...).
– Agents itinerants, ils réalisent une suite d'interactions avec les serveurs. On privilégie, par exemple, les accès locaux. Par exemple, l'agent va consulter un premier serveur qui lui fournit une liste de livres concernant les agents et il va ensuite consulter un deuxième serveur qui lui fournit une liste de prix. L'agent réalise ensuite la jointure.
– Agents d'adaptation, ils étendent les fonctions du service, ou s'adapte aux besoins spécifiques des clients. Par exemple, un client demande un document à un serveur et lui fournit également un algorithme de compression et de chiffrement. L'agent de compression et de cryptage s'exécute sur le serveur et ren-

voie au client le document chiffré et compressé. L'adaptation peut concerner également une répartition de la charge.

5.2.5 Agents mobiles

Quelle mobilité ?

1. mobilité faible (non transparente) ;
2. mobilité forte (transparente).

1. Mobilité faible. L'agent doit préparer sa migration en sauvegardant son état dans des variables et une fois sa migration effectuée, il doit restaurer son état et reprendre son exécution, en général à un nouveau point du programme.
2. Mobilité forte. L'agent après sa migration repart dans le même état et exactement au même point dans son code que celui avant migration.

5.2.5 Agents mobiles

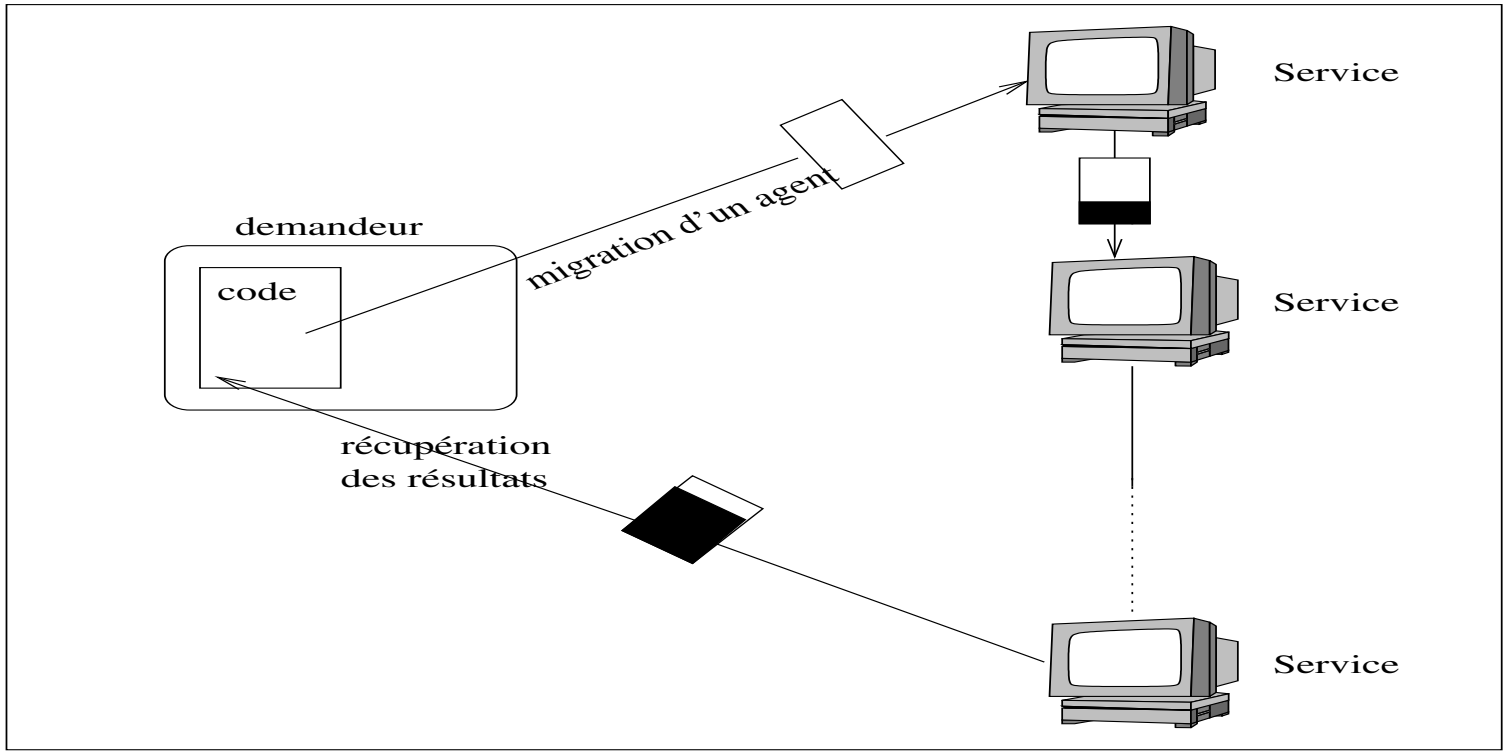


FIG. 11 – Exemple de scénario de mobilité d'un agent

L’agent est activé, il exécute sa mission en migrant et retourne sur le site demandeur. La connection a lieu uniquement durant la phase de migration.

5.2.5 Agents mobiles

Quelques caractéristiques

- Opérations à gros grains, code défini par l'utilisateur ;
- Connection uniquement durant la phase de migration $\Rightarrow$ asynchronisme, le site de départ peut se deconnecter ;
- Utilisable dans le cadre des réseaux de grande échelle
  - migration où se trouvent les données ;
  - accepte les latences ;
  - reduit le traffic de données.

- Localisation explicite
  - migration *proactive* : **l'agent décide** où aller et quand ;
  - migration *réactive* : réponse à une requête externe

5.2.5 Agents mobiles

Agents mobile vs. migration de processus

Migration de processus

- C'est le mouvement d'une entité *en cours d'éxécution* sur une machine distante ;
- Utilisé pour la répartition de charge, le partage d'information, la tolérance aux pannes ;
- Mécanisme puissant mais complexe à metre en œuvre (hétérogénéité, capture d'état . . .) ;

5.2.5 Agents mobiles

Agents mobile vs migration de processus

	Migration de processus	**Agents mobiles**
But de la migration	Partage de charge, survivance aux arrêts	Naviguer en fct. de l'appli., supporter des op. déconnectées.
Initiative	Système d'exploitation	Application
Prise en compte de la répartition	Implicite	Explicite
Portée	LAN	WAN

5.2.5 Agents mobiles

Quelle infrastructure ? **L'agence**

– Environnement local pour accueillir les agents et leur permettre de réaliser leur tâches ;

- Elle fournit un support pour :
  - L'exécution (allocation de ressources par le runtime) ;
  - La communication agents-serveurs (locaux ou distants) ;
  - La migration (transfert d'état) ;
  - La désignation et la localisation des agents ;
  - Le contrôle de l'utilisation des ressources ;
  - La sécurité.

5.2.5 Agents mobiles

Quelle infrastructure ?

- Admission ;
  - Reception de l'agent (code, état, attributs) ;
  - Déballage des attributs liés à la sécurité.

- Allocation de ressources
  - Vérification du code ;
    - Compilation dynamique ;
    - Interprétation (directe Tcl ,Scheme ou indirecte Java) ;
  - 1 agent = 1 processus ou 1 thread ;
  - Autres attributs (besoins mémoires, limitation (durée de vie . . .)).

- Action en cas d'erreurs
  - rejet ;
  - notification à une entité ;
  - routage vers une autre agence ;

Compilation dynamique verif facile, optimisation locale par contre durée de compil, code source visible, dépendance machine.

5.2.5 Agents mobiles

Communication, quels sont les problèmes et comment les résoudre ?

- Avec des services (locaux ou non), d'autres agents (locaux ou non), le propriétaire ;
- Utilisation de mécanisme type RPC ou objets distribués ;

- Service de localisation
  - local - Info sur les agents et services ;
  - distant - Info sur les agents et services ;
  - naming services (CORBA, DCE) peuvent être utilisés.
- Communication avec le propriétaire identifié par les attributs
  - Synchrone / Asynchrone (en Gal) (boîte aux lettres) ;

- Communication entre agents
  - locaux/distants ;
  - synchrone/asynchrone ;
  - diffusion multicast local, distant, global.

- Communication avec un service ou un agent distant
  - Communication distante ou migration ?
  - Notion de coûts, compromis entre coûts de communication et migration
  - Role de l'agent, du propriétaire de l'agence dans la décision

5.2.5 Agents mobiles

La migration

- 3 étapes
  - Sérialisation du code, des données et de l'état d'exécution ;
  - Transfert ;
  - Désérialisation.

- Migration d'objets ou de processus
  - Pb de l'état (dépend de la plateforme - pile, état des registres -) ;
  - Le système décide quand et où ! $\Rightarrow$ état quelconque.

- Migration d'agents
  - L'agent décide quand et où
  - Quand
    - Pour faire autre chose, se rapprocher d'une ressource, minimiser les communications, s'agréger, augmenter ses performances …
  - Où
    - Choix statique (liste d'hôtes à visiter)
    - Choix dynamique (en fonction de son état, des résultats, des dispo …)

- Que fait l'agence ?
  - Suppression de l'agent de la liste locale ;
  - Emballage de l'agent et envoi sur le réseau ;
  - Le système d'allocation recupère les ressources.

5.2.5 Agents mobiles

La migration et le problème des ressources

- Modification des liaisons aux ressources ou migration de la ressources ;
- Types des ressources
  - Transférables ;
    - libres (ex : fichiers) ;
    - fixe (ex : BD).
  - Non transférables (ex : imprimantes) ;

- Pb relocalisation des ressources, reconfiguration des liaisons (bindings) ;
- Mecanismes :
  - perte de la liaison ;
  - déplacement de la ressource ;
  - copie de la ressource ;
  - référence réseau,
  - nouvelle liaison.

5.2.5 Agents mobiles

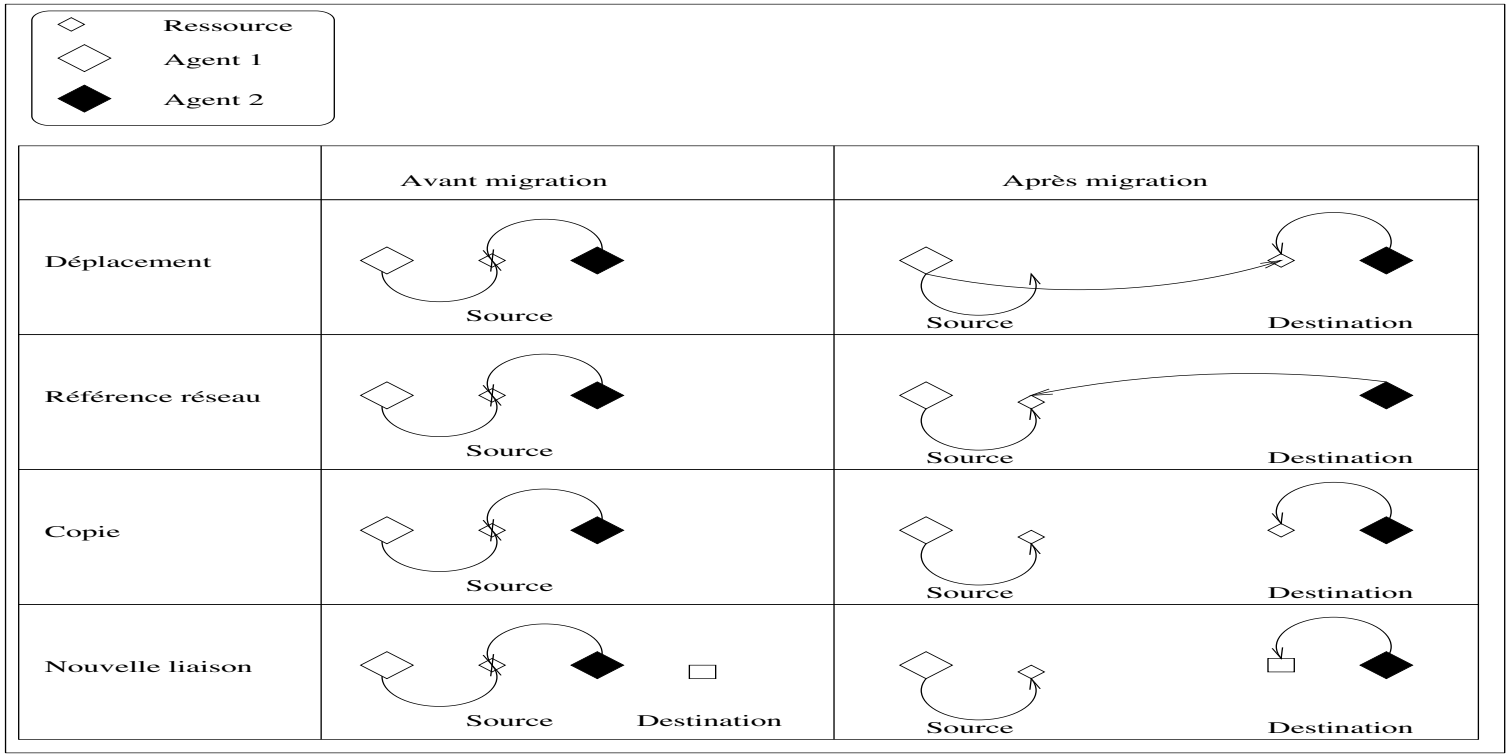


FIG. 12 – Mecanisme de liaison aux ressources

5.2.5 Agents mobiles

Contrôle et utilisation des ressources

- Au niveau des agents
  - Durée de vie ;
  - Nombre de migration ;
  - Nombre maximum de clones ;
  - Historique de migration.

- Au niveau des agences ou de l'environnement d'exécution
  - Nombre max d'agents ;
  - Gestion des accès fichiers, écran ...
  - Utilisation CPU, mémoire ...
  - Liste de contrôle d'accès (provenance (lieu, propriétaire)).

5.3 Exemples

- Avec Java
  - Aglets(IBM) http ://sourceforge.net/projects/aglets
  - Concordia (Mitsubishi) `http://www.meitca.com/HSL/Projects/Concordia`
  - Voyager (Object Space) `http://www.objectspace.com/products/voyager`
  - MOA (OSF/OpenGroup) `http://www.osf.org/RI/java/moa`
- Autres environnement (Tcl, Python . . .)
  - D'Agents (Dartmouth College)

    `http://agent.cs.dartmouth.edu/software/agent2.0/`
  - Ara (Université de Kaiserslautern)

    `http://wwwagss.informatik.uni-kl.de/Projekte/Ara/index_e.html`
  - Tacoma (Universite de Tromsø et Cornell)

    `http://www.tacoma.cs.uit.no/`

5.3 Exemples

5.3.1 Code mobile en Java

5.3.2 Aglets

5.3.3 Une plate-forme à votre choix !

5.3.1 Code mobile en Java

- Java + sérialisation + RMI $\Rightarrow$ code mobile ;
- On définit :
  - Un code mobile donc sérialisable ;
  - Un serveur qui met à disposition ce code mobile ;
    Le serveur est un objet distribué RMI ;
  - Un client (eventuellement) distant.

5.3.1 Code mobile en Java

- `Agent` code mobile ;
- `AgentFactory` interface de l'objet serveur distribué, invoquée par le client et les appels de méthodes sont gérés par RMI ;
- `AgentFactoryImpl` implémentation de l'interface, crée une instance de `Agent` et l'envoie sur demande au client ;
- `AgentClient` client du serveur, fait migrer l'objet mobile `Agent`.

5.3.1 Code mobile en Java

La classe `Agent`

```
package mobile;
import java.net.*;
public class Agent
   implements java.io.Serializable
{
 int donnee;
 String origine;

 public Agent()
  throws UnknownHostException{
   origine =
    InetAddress.getLocalHost().toString();
 }

 public void calcule(){
   donnee += 113;
 }

 public int getDonnee(){
   return donnee;
 }

 public void setDonnee(int donnee){
   this.donnee = donnee;
 }

 public String douViensTu(){
   return origine;
 }

 public String ouEsTu() throws Exception{
   return
     InetAddress.getLocalHost().toString();
 }
}
```

5.3.1 Code mobile en Java

La classe `AgentFactory`

```
package mobile;
import java.rmi.*;

public interface AgentFactory extends Remote
{
    public Object getAgent()    throws Exception;
    public void setAgent(Object a) throws RemoteException;

}
```

– On fait migrer des `Object` → généricité ;
– Il n'est pas nécessaire de connaître la classe agent.

5.3.1 Code mobile en Java

La classe `AgentFactoryImpl`

```
package mobile;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class AgentFactoryImpl
   extends UnicastRemoteObject
   implements AgentFactory {
 int data = 0;

 public AgentFactoryImpl()
    throws RemoteException{
   super();
 }

 public Object getAgent()
  throws Exception{
   Agent a = new Agent();
   a.setDonnee(this.data);
   return a;
 }

 public void setAgent(Object a)
    throws RemoteException{
   this.data += ((Agent)a).getDonnee();
 }

 public static void main(String[] argv){
   try{
    LocateRegistry.createRegistry(1099);
    System.out.println("Reg. RMI lancé");
    LocateRegistry.getRegistry(1099).bind(
     "AgentFactory",
      new AgentFactoryImpl());
    System.out.println(
      "Agent factory enregistré");
   } catch(Exception e)
   {
    System.out.println(
      "Exception interceptée : " + e);
    e.printStackTrace();
   }
 }
}
```

5.3.1 Code mobile en Java

La classe `AgentClient`

```
package mobile;
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;
import java.lang.reflect.*;

public class AgentClient{
 public static void main(String[] argv){
  try{
   String url =
     "rmi://"+argv[0]+":"+argv[1];
   System.setSecurityManager(
     new RMISecurityManager());
   AgentFactory af = (AgentFactory)
     Naming.lookup(url+"/AgentFactory");
   Object a = af.getAgent();
   Class cl = a.getClass();
   Method ou = cl.getMethod("douViensTu",
                            null);
   System.out.println(‘‘Je viens de ‘‘ +
                      ou.invoke(a,null));
   ou = cl.getMethod("ouEsTu", null);
   System.out.println(‘‘Je suis en ‘‘ +
                      ou.invoke(a,null));
   Method getDonnee =
     cl.getMethod("getDonnee", null);
   System.out.println("Donnee avant calcul:"
     + ((Integer)getDonnee.invoke(
                      a, null)).intValue());
   cl.getMethod("calcule", null)
                            .invoke(a, null);
   getDonnee =
     cl.getMethod("getDonnee", null);
   System.out.println("Donnée après calcul      : "
     + ((Integer)getDonnee.invoke(
                      a, null)).intValue());
   af.setAgent(a);
  }catch (Exception e)
  {
   System.out.println("Excep. interceptée "
                      + e);
   e.printStackTrace();
  }
 }}
```

5.3.1 Aglets

- **Ag**ents App**lets** ;
- IBM Japon 96 ;
- Porté en Java 2 depuis que c'est en Open Source ;
- Aglet = objet Java comportant un flot d'exécution et des méthodes pour la mobilité ;
- Contexte = espace d'adressage sur une machine pouvant abriter un ensemble d'aglets ;
- Proxy = protège l'accès direct, gère la localisation

- **L'aglet** c'est un agent mobile qui se déplace de machine en machine avec son code et ses données ;
- **Le contexte** c'est l'agence. C'est donc un objet statique lié à une plate-forme. Il permet de gérer les aglets en cours d'exécution et d'assurer la sécurité de l'hôte.
- **Le proxy** C'est le représentant, en fait de l'Aglet. Il protège l'Aglet vis à vis de ses méthodes publiques et masque la localisation réelle de l'Aglet.

5.3.2 Aglets

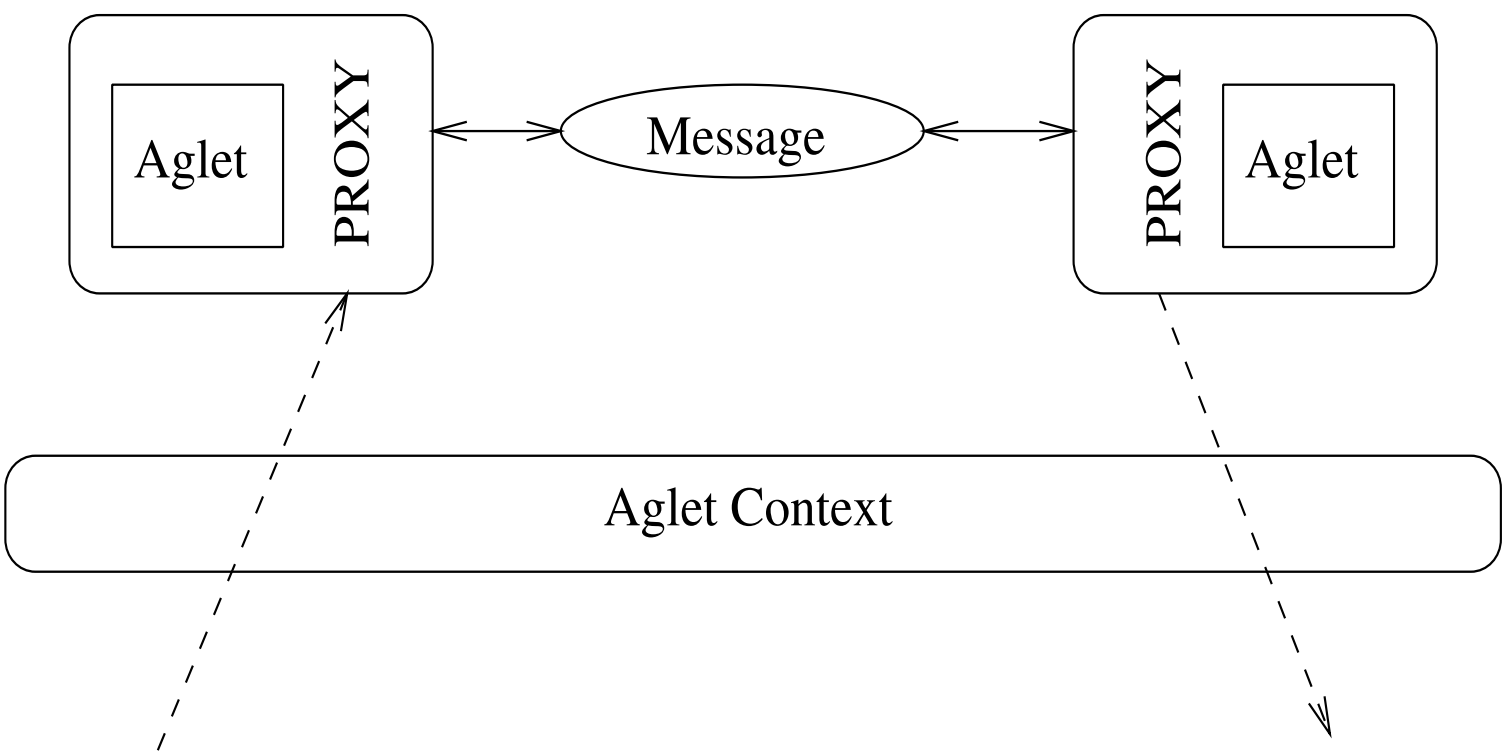


FIG. 13 – Aglets

5.3.2 Aglets

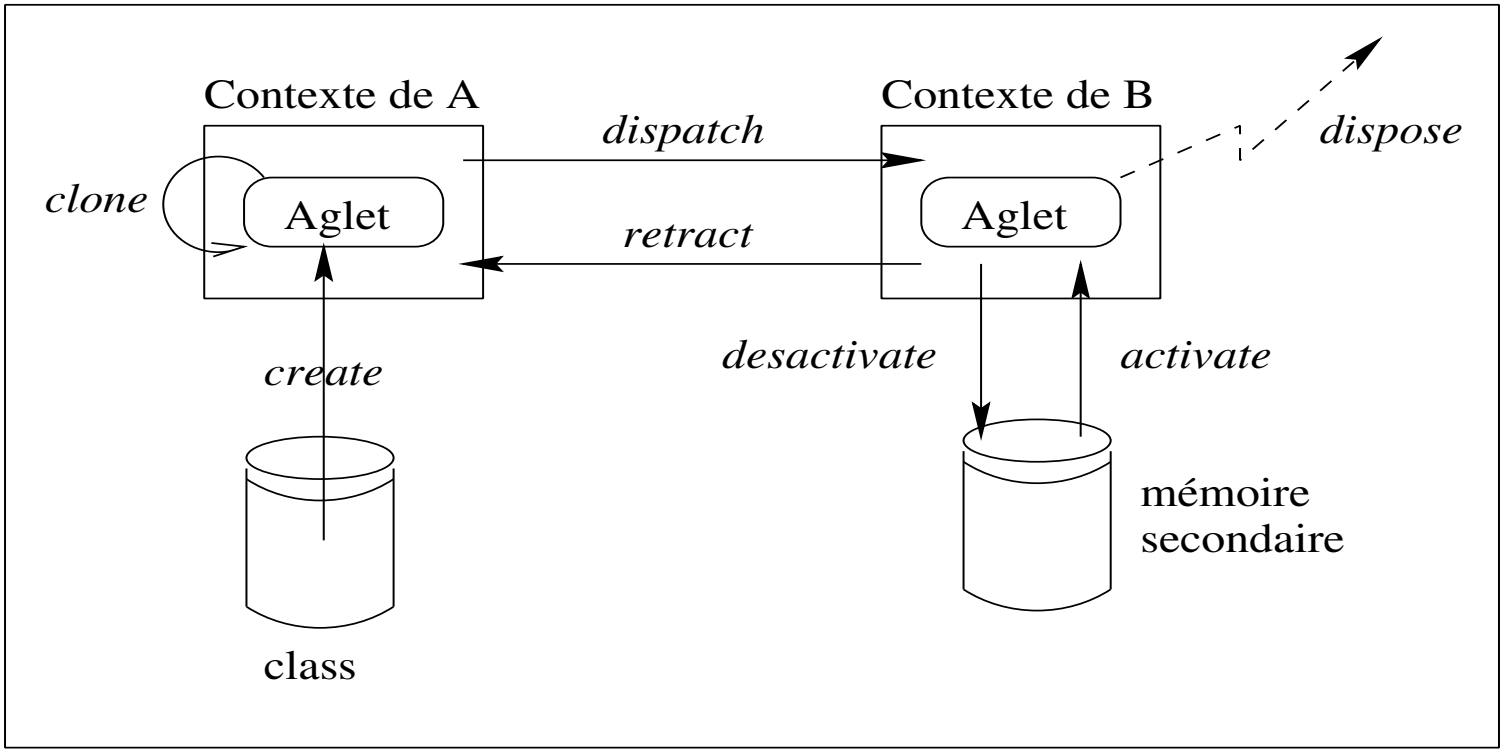


FIG. 14 – Cycle de vie d'une aglet

- **Création**, elle s'effectue dans le cadre d'un contexte. A sa création l'Aglet se voie attribuer un identifiant unique `AgletIdentifier` et à un objet d'accès `AgletProxy` ;
- **Clonage**, il y a duplication de l'Aglet dans un contexte. L'Aglet cloné à un nouvel identifiant et il repart d'un état initial i.e il ne conserve pas le thread d'exécution.
- **L'envoi**, l'Aglet migre du contexte A vers le contexte B.
- **Rétraction**, l'Aglet est retiré de son contexte afin de l'insérer dans le contexte qui invoque l'opération.
- **Activation et désactivation**, l'Aglet peut être stocké momentanément sur disque et réactivé à l'expiration d'un timeout ou d'un événement externe.
- **Suppression** fin de vie de l'Aglet, son exécution est interrompue et il est supprimé de son contexte d'exécution.

5.3.2 Aglets

Un Aglet c'est donc :

- Une classe qui dérive de la classe `Aglet` ;
- Un graphe d'objets sérialisable ;
- Des méthodes prédéfinies pour la gestion des Aglets ;
  - `Aglet.dispatch(URL url)`
  - `Aglet.deactivate(long time)`
  - `Aglet.clone()`
  - `getAgletContext()`

- Des méthodes à définir :
  - `Aglet.onCreation(Object init)`
  - `Aglet.run()`
  - `Aglet.handleMessage(Message message)`
  - `Aglet.onDisposing()`

Programmation d'un Aglet

```
public class AgletCoucou extends Aglet
{
 public void onCreation(Object init) {
  System.out.println(''j'existe'');
 }

 public void run() {
  System.out.println(''coucou'');
 }

 public boolean handleMessage(Message message) {
  if (message.sameKind(''Dit coucou''))
     {
      System.out.println(''coucou'');
      return true;
     }
  return false;
 }

 public void onDisposing() {
  System.out.println(''Adieu monde cruel'');
 }
}
```

```
public void onCreation(java.lang.Object init)
       Initializes the new aglet. This method is called only once in the life cycle of an aglet.
       Override this method for custom initialization of the aglet.
       Parameters:
              init - the argument with which the aglet is initialized.

public void run()
       Is the entry point for the aglet's own thread of execution.
       This method is invoked upon a successful creation, dispatch, retraction,
       or activation of the aglet.

public boolean handleMessage(Message message)
       Handles the message form outside.
       Parameters:
              msg - the message sent to the aglet
       Returns:
              true if the message was handled. Returns false if the message was not handled.
              If false is returned, the MessageNotHandled exception is thrown
              in the FutureReply.getReply
              and AgletProxy.sendMessage methods.
  public boolean sameKind(java.lang.String k)
         Checks if the message has same kind as given string.

 public void onDisposing()
       Is called when an attempt is made to dispose of the aglet. Subclasses may override
       this method to implement
       actions that should be taken in response to a request for disposal.
       Throws:
              java.lang.SecurityException - if the request for disposal is rejected.
```

5.3.2 Aglets

Proxy d'un Aglet :

- c'est le moyen d'appeler une méthode de l'Aglet (déplacer, envoyer un message . . .) ;
- cela masque la localisation de l'Aglet ;
- cela protège l'Aglet sur ses méthodes.
- c'est donc un objet intermédiaire qui permet la manipulation de l'Aglet.

- On obtient un proxy d'une Aglet par :
  - `AgletContext.getAgletProxies()`
  - `AgletContext.getAgletProxy(AgletID)`
  - ou en adressant un contexte (`URL`) à distance
  - ou en le recevant dans un message.

```
public java.util.Enumeration getAgletProxies()
       Gets an enumeration of all aglets in the current context including deactivated aglets.
       Returns:
              a list of proxies.

public AgletProxy getAgletProxy(AgletID id)
       Gets a proxy for an aglet in the current context. The selected aglet is specified by its identity
       Parameters:
              id - the identity of the aglet.
       Returns:
              the proxy.
```

5.3.2 Aglets

Message

- Il est composé de 2 champs, le type `String` et un objet `Object`
- Il peut être envoyé de 3 façons différentes :
  - synchrone
    `AgletProxy.sendMessage(Message msg)`
  - asynchrone
    `AgletProxy.sendAsyncMessage(Message msg)`
  - asynchrone sans attente de résultat
    `AgletProxy.sendOnewayMessage(Message msg)`

```
public Message(java.lang.String kind,
               java.lang.Object arg)
       Constructs a message with an argument value.
       Parameters:
              kind - a kind of this message.
              arg - an argument of this message.

public java.lang.Object sendMessage(Message msg)
                            throws InvalidAgletException,
                                   NotHandledException,
                                   MessageException
       Sends a message in synchronous way. This waits for finishing the message handing.
       Parameters:
              msg - a message to send.
       Returns:
              the result object if any. null if not.

public FutureReply sendAsyncMessage(Message msg)
                            throws InvalidAgletException
       Sends a message in asynchronous way.
       Parameters:
              msg - a message to send.
       Returns:
              a future object that will give you the reply of the message.

public void sendOnewayMessage(Message msg)
                      throws InvalidAgletException
       Sends a oneway message to the aglet. No acknowledgement will be sent back to the sender.
       Parameters:
              msg - a message to send.
```

5.3.2 Aglets

Programmation de la mobilité

- Programmation basée sur des événements
  - gestion d'événement de mobilité associée à un Aglet

    interface `MobilityListener`
  - définies par le programmeur
    - `MobilityListener.onArrival(MobilityEvent event)`
    - `MobilityListener.onDispatching(MobilityEvent event)`
    - `MobilityListener.onReverting(MobilityEvent event)`

```
class monGestionnaire implements MobilityListener
{
 public void onDispatching(MobilityEvent evenement) {
  System.out.println(‘‘Je m’en vais’’);
 }

 public void onReverting(MobilityEvent evenement) {
  System.out.println(‘‘Je reviens au bercail’’);
 }

 public void onArrival(MobilityEvent evenement) {
  System.out.println(‘‘Je suis arrivé’’);
 }
}

public class AgletCoucou extends Aglet
{
 public void onCreation(Object init) {
  System.out.println(‘‘j’existe’’);
  MobilityListener gestionnaire =
                       new monGestionnaire();
  addMobilityListener(gestionnaire);
 }
//etc
}
```

```
public void onArrival(MobilityEvent event)
       Invoked just after the aglet arrived at the destination.

public void onDispatching(MobilityEvent event)
       Invoked when the aglet is attempted to dispatch.

public void onReverting(MobilityEvent event)
       Invoked when the aglet is retracted.

public final void addMobilityListener(MobilityListener listener)
       Adds the specified mobility listener to receive mobility events from this aglets
```

5.3.2 Aglets

Programmation d'un contexte

- Programmation basée sur les événements, interface `ContextListener`
  - événements concernant le contexte
    - `ContextListener.contextStarted(ContextEvent evt)`
    - `ContextListener.contextShutdown(ContexEvent evt)`
  - événements concernant les aglets
    - `ContextListener.agletCreated(ContextEvent evt)`
    - `ContextListener.agletCloned(ContextEvent evt)`
    - `ContextListener.agletArrived(ContextEvent evt)`
    - `ContextListener.agletDispatched(ContextEvent evt)`
    - `ContextListener.agletReverted(ContextEvent evt)`
    - `ContextListener.agletStateChanged(ContextEvent evt)`

```
public void contextStarted(ContextEvent ev)
       Called when the context is started

public void contextShutdown(ContextEvent ev)
       Called when shutting down

public void agletCreated(ContextEvent ev)
       Called when an aglet has been created

public void agletCloned(ContextEvent ev)
       Called when an aglet has been cloned

public void agletArrived(ContextEvent ev)
       Called when an aglet has arrived

public void agletActivated(ContextEvent ev)
       Called when an aglet has been activated

public void agletDispatched(ContextEvent ev)
       Called when an aglet has been dispatched

public void agletReverted(ContextEvent ev)
       Called when an aglet has been reverted.

public void agletStateChanged(ContextEvent ev)
       Called when the state of an aglet has changed.
```

5.3.2 Aglets

- messages : communication synchrone et asynchrone ;
- migration proactive *Dispatch()*, notion d'itinéraire ;
- migration réactive *Retract()* ;
- Mobilité faible : sérialisation du code et des données, mais pas de l'état du thread. L'exécution reprend au début.