Séance 03 : TP – Service de Nommage

Une question importante se posant lors du développement d'applications distribuées lorsque les clients et serveurs se trouvent dans des postes différents au sein d'un réseau est relative à l'accès de la référence du serveur par le client.

Lors des exemples précédents, l'approche utilisée consistait à stocker l'IOR du serveur sous-forme de chaîne de caractères au sein d'un fichier sur le dossier courant du serveur et ensuite se charger manuellement de le transmettre sur le poste du client. Ensuite le client se charge de la lecture et la reconversion du contenu en objet. [Fastidieuse]

Le service de Nommage « *Naming Service* » spécifié dans le modèle CORBA propose un moyen simple et interopérable qui permet de :

- Nommer des objets et en retour retrouver un objet par son nom
- Création d'hiérarchies de noms à travers l'usage des contextes de noms «*naming contexts*»

Par ailleurs, deux types de liens [bindings] seront à distinguer :

- Liens objet qui permettent d'associer un nom une référence d'objet.
- Liens qui permettent d'associer un nom à un nom de contexte.

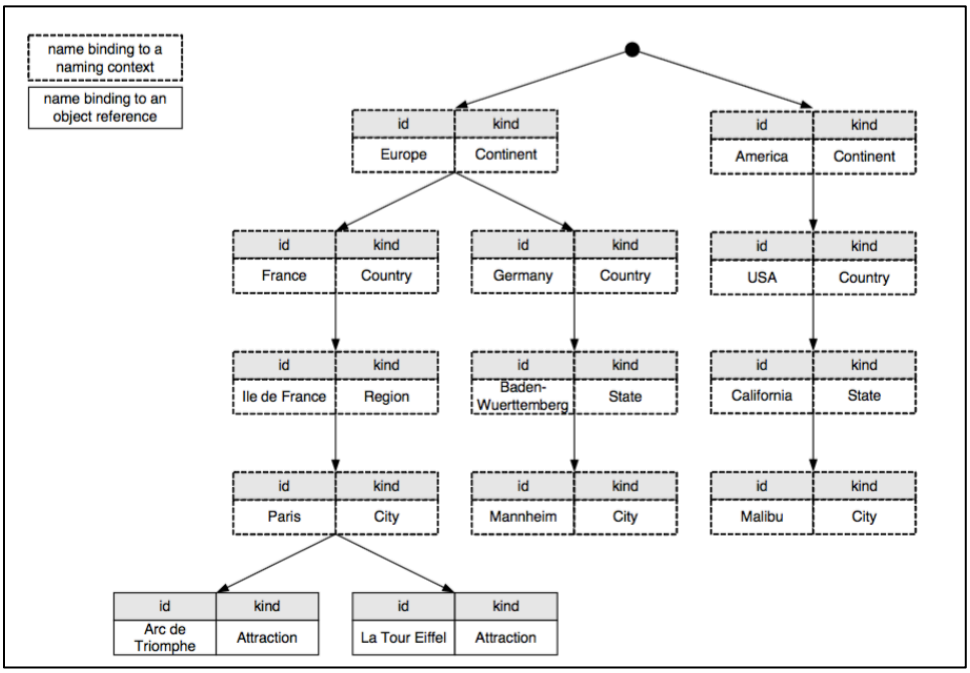
La définition IDL définie pour les noms et les noms de contextes dans la spécification du service de nommage :

```
module CosNaming
  {
    typedef string Istring;
    struct NameComponent {

           Istring id;
           Istring kind;
    };

    typedef sequence<NameComponent> Name;
... };
```

- `Istring` est un paramètre fictif pour un type IDL « internationalized string » qui figurait dans les anciennes versions antérieures de la spécification mais maintenu pour des raisons de compatibilité.
- Un nom géré par le NS consiste en une séquence de «`NameComponent`».
- Chaque «`NameComponent`» comporte comporte deux membres chaînes de caractère `id` et `kind`.
- Un nom avec un seul composant est dit *simple name* (nom simple) et un nom avec composants multiples est dit *compound name* (nom composé).
- Chaque composant d'un nom composé à l'exception du dernier est utilisé pour nommer un contexte, le dernier représente la référence de l'objet.

- Dans la représentation en chaîne, les noms des composants sont séparés par des slash '/', les `id` et `kind` par des points '.'.

Exemple d'une hiérarchie :

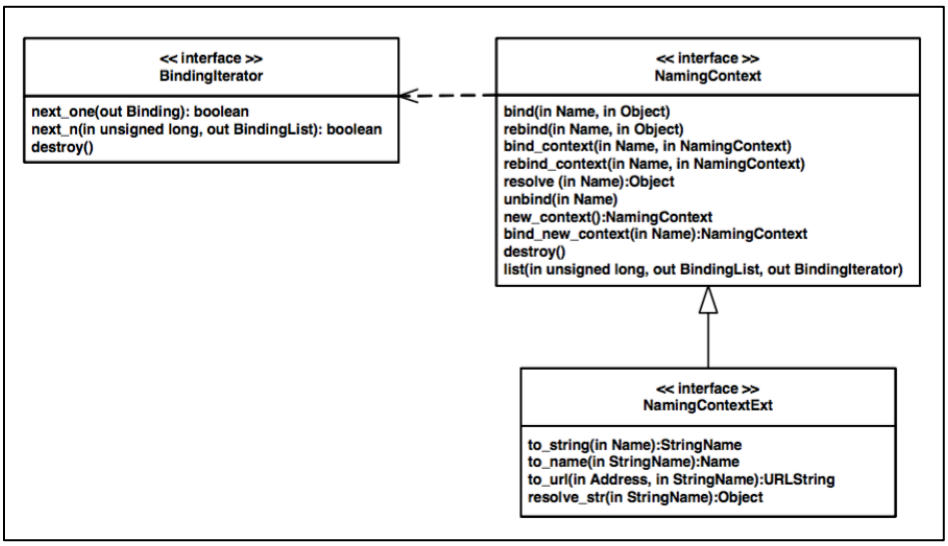


- Le point au top représente le contexte de nommage initial non nommé qui représente le point d'entrée à l'hiérarchie de noms.
- Deux contextes de nommages sont définis, associés aux noms `"Europe.Continent"` et `"America.Continent"`.
- En dessous, plusieurs niveaux sont définis ; le contexte de nommage `"Europe.Continent/France.Country/Ile de France.Region/Paris.City"` ne contient plus de contextes de nommage ultérieurs.
- Deux associations d'objets `"Arc de Triomphe.Attraction"` et `"La Tour Eiffel.Attraction"` sont définis.
- Un exemple de nom composé : `"Europe.Continent/France.Country/Ile deFrance.Region/Paris.City/ArcdeTriomphe.Attraction"`.
  Celui-ci comporte 4 contextes de noms, `"Europe.Continent"`, `"France.Country"`, `"Ile de France.Region"`, et `"Paris.City"`, suivis par le nom simple `"Arc de Triomphe.Attraction".`

Définition IDL du service de Nommage :

- Trois interfaces définissent le service de nommage :
  - `NamingContext`
  - `BindingIterator`
  - `NamingContextExt`

- L'interface fondamentale « `NamingContext` » fournit des opérations de création et de suppression de liens de noms ainsi que la récupération d'objets liés à un nom. Elle founit également des moyens de création et de suppression de contextes de noms au sein d'un contexte de nom



Les opérations `bind()` et `rebind()` permettent de créer un lien (binding) de nom pour un objet CORBA dans le `NamingContext` pour lequel ils sont invoqués.

`bind_context()` et `rebind_context()` opèrent avec des sémantiques similaires ; elles permettent de créer un nouveau `NamingContext` dans le contexte au sein duquel elles sont invoquées. (En supposant que l'objet passed `NamingContext` a déjà été créé à travers une invocation `new_context()`.

L'opération `unbind()` supprime un lien (binding) d'un contexte.

L'opération `destroy()` permet de supprimer définitivement un contexte de nom qui ne sera plus utilisé.

L'opération `resolve()` permet de retrouver un objet nommé ou un contexte nommé à partir du nom. Elle retourne la référence de l'objet associé.

L'opération `list()` permet au client d'itérer sur les noms contenus au sein d'un contexte. Elle retourne un objet de type `BindingIterator`. La navigation est poursuivie à travers l'invocation des opérations `next_one()` et `next_n()` sur l'objet retourné.

L'interface `NamingContextExt` est une sous-interface de l'interface `NamingContext`. En plus des formats chaînes de noms, un format URL est aussi défini avec les opérations conversions associées.

- Approche

Les applications serveur client ont besoin d'obtenir une référence au service de nommage afin d'associer et de résoudre des noms avec:

- Commande Standard :

```
-ORBInitRef <ObjectID>=<ObjectURL>
```

- `ObjectID` :corresponds au nom du service qui sera passé en invocation à `resolve_initial_references()`
- `ObjectURL` : un des URLs CORBA

- URLs CORBA :

La spécification CORBA définit un nombre de schémas d'adressage URL.

Ces schémas permettent d'identifier une référence initiale au service de nommage sur la base de son adresse réseau et numéro de port.

Scheme	Description	Status
`IOR:`	Standard stringified IOR format	Required
`corbaloc:rir:`	Simple object reference; implicitly resolved via `resolve_initial_references()`	Required
`corbaloc::` or `corbaloc:iiop:`	IIOP-specific stringified IOR format	Required
`corbaname:rir:`	Name to be resolved relative to the initial naming context	Required
`corbaname::` or `corbaname:iiop:`	Name to be resolved relative to the initial naming context	Required

La syntaxe générale d'un schéma URL `corbaloc` est la suivante :

```
corbaloc:[iiop]:[version@]host[:port][/object_key]
```

Une URL `corbaloc pour` IIOP comporte les éléments suivants :

- L'identificateur `corbaloc`
- Protocole identificateur (par défaut `iiop` donc optionnel)
- Version du protocole (optionnel aussi, valeur défaut `1.0`)
- Nom ou adresse IP du poste
- Numéro du port (Optionnel, valeur défaut 2089)
- Clé d'objet optionnelle

Example :

```
-ORBInitRef NameService=corbaloc::localhost:777/NameService
```

- Implémentation Serveur :

Server.java

```java
// Server.java
//import Count.*;
import java.util.Properties;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import static java.lang.System.*;
public class Server {
  private ORB orb;
  private POA rootPOA;
  private void initializeORB(String[] args) {
   Properties props = getProperties();
    orb = ORB.init(args, props);
    try {
      rootPOA = POAHelper.narrow(orb.
        resolve_initial_references("RootPOA"));
    } catch (org.omg.CORBA.ORBPackage.InvalidName ex) { }
  }
  public Server(String[] args) {
    try {
      initializeORB(args);
      NamingContext nc = NamingContextHelper.narrow(
        orb.resolve_initial_references("NameService"));
      CounterImpl c_impl = new CounterImpl();
      Counter c = c_impl._this(orb);
      NameComponent[] name = new NameComponent[1];
      name[0] = new NameComponent();
      name[0].id = "Counter";
      name[0].kind = "IIOP";
      nc.rebind(name, c);
      out.println("Server started. Stop: Ctrl-C");
      rootPOA.the_POAManager().activate();
      orb.run();
    } catch(Exception ex) {
      out.println("Exception: " + ex.getMessage());
      exit(1);
} }
  public static void main(String args[]) {
    new Server(args);
} }
```

- Appel de `resolve_initial_references()` avec argument `"NameService"` (Obtention réf NS)
- Transformation en objet `NamingContext (nc)`
- Création de l'objet c
- Association de l'objet c au nom `Counter.IIOP` au sein du contexte de nom initial `nc`. (méthode rebind)

Abréviation possible :

```
nc.rebind(new NameComponent[] {
      new NameComponent("Counter", "IIOP") }, c);
```

- Implémentation Client :

GUIClient.java

```java
// GUIClient.java
//import Count.*;
import java.awt.GridLayout;
import java.awt.event.*;
import java.util.Properties;
import javax.swing.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import static java.lang.System.*;

public class GUIClient extends JPanel {
  private Counter c;
  private ORB orb;

  private void initializeORB(String[] args) { ...
  }

private void createGUI() { ...
}


  public GUIClient(String[] args) {
    try {
      initializeORB(args);
      NamingContext nc = NamingContextHelper.narrow(
        orb.resolve_initial_references("NameService"));
      NameComponent[] name = new NameComponent[1];
      name[0] = new NameComponent();
      name[0].id = "Counter";
      name[0].kind = "IIOP";
      org.omg.CORBA.Object obj = nc.resolve(name);
      c = CounterHelper.narrow(obj);
      createGUI();
    } catch (BAD_PARAM ex) {
      out.println("Narrowing failed");
      exit(3);
    } catch(Exception ex) {
      out.println("Exception: " + ex.getMessage());
exit(1); }
  }

  public static void main(String[] args) {
    JFrame f = new JFrame("Counter Client");
    f.getContentPane().add(new GUIClient(args));
    f.pack();
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    f.setVisible(true);
} }
```

- Traitement analogue au cas du Serveur
- Appel de la méthode `resolve()` (Récupération de la référence de l'objet du serveur du contexte de nommage initial)
- Retour de type objet transformé en type `Counter`.

Abréviation possible :

```
org.omg.CORBA.Object obj = nc.resolve(
    new NameComponent[] {
      new NameComponent("Counter",
"IIOP") });
```

- Exécution

Usage de l'outil `orbd` du JDK et invocation avec spécification du numéro de port (Valeur défaut 1049)

```
orbd –ORBInitialPort 777
```

Exécution du Serveur

```
java Server -ORBInitRef
NameService=corbaloc::localhost:777/NameService
```

Exécution Client :

```
java GUIClient -ORBInitRef
NameService=corbaloc::localhost:777/NameService
```