Chapter 1

Using LDAP

1.1 Présentation du protocole LDAP

Le terme LDAP (Lightweight Directory Access Protocol) désigne un protocole permettant l'accès en lecture et en écriture à des annuaires d'entreprises. Ces bases de données, optimisées pour la lecture, sont des référentiels d'informations sur les composants d'une organisation (individus, ressources, organisation fonctionnelle). Le plus souvent, les annuaires LDAP sont exploités pour l'identification des utilisateurs aux services de messagerie ou aux applications intranet. Dans le cadre du développement d'applications e-business, les annuaires LDAP sont des composants techniques essentiels et critiques : il est le plus souvent obligatoire d'interfacer un logiciel avec ces annuaires.

Pharo dispose de différents frameworks pour le développement web (Seaside, AIDAweb, etc.) ainsi que pour l'accès aux bases de données (DBXTalk). Quelle est la situation avec LDAP ? Peut on créer des applications capables d'identifier un utilisateur ou de récupérer des informations dans un annuaire LDAP ?

Installation de LDAPlayer

Pour communiquer avec un annuaire LDAP, Pharo dispose de la bibliothèque de classes LDAPlayer créée par Ragnar Hojland Espinosa. Elle disponible via Squeaksource à l'adresse suivante: http://www.squeaksource.org/LDAPlayer/.

Pour l'installer, lancez Pharo et définissez un nouveau dépôt HTTP à l'aide de Monticello.

```
MCHttpRepository
```

```
location: 'http://www.squeaksource.com/LDAPlayer'
user: '' password: ''
```

Stéf ►*better use a Gofer expression!*◄

1.2 Les principales classes

La plupart des classes que nous allons découvrir sont stockées dans le paquet LDAP-Core. Une connexion à un annuaire LDAP instancie un objet LDAPConnection. Des requêtes vers l'annuaire retourne des objets LDAPResult. La classe LDAPAttrModifier fournie plusieurs méthodes de classe permettant de modifier les attributs d'une entrée de l'annuaire. La classe LDAPFilter a pour finalité de définir des filtres de recherche et de limiter ainsi le volume d'informations retourné au client.

Définir une connexion. Créons une nouvelle catégorie nommée 'AppLDAP' pour découvrir les fonctionnalités de LDAPlayer. Notre objectif est de mettre en situation les principales fonctionnalités de la bibliothèque LDAPlayer et de vous permettre de les réutiliser facilement dans une véritable application. Cette catégorie contient une classe unique nommée 'Ldap' qui renfermera l'ensemble de nos méthodes d'instances.

Pour définir la connexion, nous allons utiliser cinq variables d'instance qui représentent le nom du serveur LDAP (hostname), le port TCP de connexion (port), le compte utilisateur (bindDN), le mot de passe de l'utilisateur (password) et la base de la recherche (baseDN) qui sera utilisée pour construire un DN (Distinguished Name).

```
Object subclass: #Ldap
    instanceVariableNames: 'baseDN bindDN hostname password port'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Ldap'
```

Stéf ►*I do not like DN*◄ Stéf ►*We should explain that basically everything resolves around the call.* ◄

```
Ldap>>baseDN
    ↑ baseDN
Ldap>>baseDN: anObject
    baseDN := anObject
Ldap>>bindDN
    ↑bindDN
Ldap>>bindDN: anObject
    bindDN := anObject
Ldap>>hostname
```

```
  ↑hostname
Ldap>>hostname: anObject
  hostname := anObject
Ldap>>password
  ↑password
Ldap>>password: anObject
  password := anObject
Ldap>>port
  ↑ port
Ldap>>port: anObject
  port := anObject
```

Pour initialiser ces variables d'instances, créons une méthode initialize dans le protocole initalize-release.

```
Ldap>>initialize
  super initialize.
  self hostname: 'ldap.mondomaine.org'.
  self port: 389.
  self bindDN: 'cn=admin,dc=domaine,dc=org'.
  self password: 'mysecretpassword'.
  self baseDN: 'ou=people,dc=domaine,dc=org'.
```

Stéf ►*what is ou=people,dc=domaine,dc=org'?*◄ Stéf ►*is there a ldap that we could query to try*◄

Nous avons également besoin d'une variable d'instance permettant de sauvegarder un objet décrivant la connexion vers l'annuaire LDAP.

```
Object subclass: #Ldap
  instanceVariableNames: 'connection baseDN bindDN hostname password port'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'Ldap'
```

Bien évidemment, nous avons besoin de deux accesseurs pour écrire et lire dans cette variable d'instance. Créons un protocole 'accessing' et définissons les deux méthodes suivantes :

```
Ldap>>connection
  ↑ connection
Ldap>>connection: anObject
  connection := anObject
```

Nous pouvons maintenant définir un protocole 'action' qui contiendra l'ensemble des méthodes d'instances ayant une interaction avec l'annuaire LDAP. Commençons par définir une méthode connect qui assure la connexion et initialise la variable d'instance connection.

```
| req |
self connection: (LDAPConnection to: self hostname port: self port).
req := connection bindAs: self bindDN credentials: self password.
req wait.
↑ self.
```

Stéf ►*je ne comprends pas pourquoi on utilise le self connection: et apres on accede directement*◄

Pour travailler proprement, il nous faut également une méthode permettant de déconnecter notre application une fois que la connexion à l'annuaire LDAP n'est plus utile. Pour cela, définissez simplement une méthode 'disconnect' qui sera chargée de cette tâche.

```
Ldap>>disconnect
  self connection disconnect.
```

Est-ce que cela fonctionne ? Un petit test rapide pour vérifier. Ouvrons le Workspace et exécutons le code suivant :

```
ann := Ldap new connect.
ann disconnect.
```

Si tout c'est bien passé, félicitations ! Notre application s'est connecté à votre annuaire LDAP. Passons à la suite et tentons de récupérer des informations.

Chercher des informations

Nous pouvons maintenant essayer de lire des données dans notre annuaire LDAP. Pour cela, nous allons définir une première méthode 'searchAll', très simple, qui récupère l'intégralité des entrées à partir du DN de base. Facile n'est-ce pas ? Par contre, ce n'est pas une méthode très recommandable si votre annuaire contient plusieurs milliers d'entrées. Lorsque l'on fait une recherche dans un LDAP, on travaille plutôt avec des filtres afin de limiter le volume d'information récupéré par l'application cliente. Définissons une méthode 'search' recherchant les entrées pour lesquelles un attribut à une certaine valeur.

```
Ldap>>searchAll
  | req result |
  req := self connection
          newSearch: self baseDN
          scope: (LDAPConnection wholeSubtree)
          deref: (LDAPConnection derefNever)
          filter: nil
          attrs: nil
```

```
            wantAttrsOnly: false.
    ↑req result
```

Dans le workspace, nous pouvons maintenant lire et afficher ces informations. A titre d'exemple, nous n'affichons que l'attribut cn :

```
| ann result |
ann := Ldap new connect.
result := ann searchAll.
result do: [ :each |
    Transcript show: (each attrAt: #cn); cr ].
ann disconnect.
```

```
Ldap>>search: aValue attribute: aAttribute
    | req result |
    req := self connection
        newSearch: self baseDN
        scope: (LDAPConnection wholeSubtree)
        deref: (LDAPConnection derefNever)
        filter: (LDAPFilter with: aAttribute equalTo: aValue)
        attrs: nil
        wantAttrsOnly: false.
    ↑req result
```

Dans le Workspace, nous pouvons maintenant rechercher les entrées pour lesquelles le champ 'equipe' est égal à 'INFORMATIQUE'.

```
| ann result |
ann := Ldap new connect.
result := ann search: 'INFORMATIQUE' attribute: 'equipe'.
result do: [ :each |
    Transcript show: (each attrAt: #cn); cr ].
ann disconnect.
```

Modifier des attributs

La modification des attributs consiste à ajouter une valeur à un attribut ou à supprimer un attribut. Tout d'abord, définissons une méthode pour ajouter un attribut à une entrée existante. Cette méthode reçoit trois paramètres qui sont l'identifiant unique d'une entrée dans l'annuaire (DN) construit à partir de l'UID (User ID) et du DN de base, le nom du paramètre modifié et la valeur affecté à ce paramètre.

```
Ldap>>addValueTo: aDN attribute: aAttribute with: aValue
    | req ops |
    ops := { LDAPAttrModifier addTo: aAttribute values: { aValue }.}.
    req := connection modify: (aDN, ',', self baseDN) with: ops. req wait.
```

Essayons maintenant cette méthode dans le Workspace en ajoutant une valeur à l'attribut 'givenName' de l'utilisateur 'dupont' :

```
| ann |
ann := Ldap new connect.
ann addValueTo: 'UID=dupont' attribute: 'givenName' with: 'Alfred'.
ann disconnect.
```

Nous allons ensuite créer une méthode permettant de changer la valeur d'un attribut. Cette méthode reçoit trois paramètres qui sont le DN de l'entrée devant être modifiée, le nom de l'attribut et la valeur affectée.

```
Ldap>>changeValueOf: aDN attribute: aAttribute with: aValue
   | req ops |
   ops := { LDAPAttrModifier set: aAttribute to: { aValue }}.
   req := connection modify: (aDN, ',', self baseDN) with: ops.
   req wait.
```

L'exemple suivant modifie l'attribut 'Statut' de l'utilisateur 'dupont'.

```
| ann |
ann := Ldap new connect.
ann changeValueOf: 'UID=dupont' attribute: 'Statut' with: 'programmer'.
ann disconnect.
```

Créons maintenant une méthode qui supprime un attribut dans le LDAP. Cette méthode reçoit un DN en paramètre afin de spécifier l'entrée concernée par la suppression de l'attribut.

```
Ldap>>deleteAttribute: aAttribute from: aDN

   | req ops |
   ops := { LDAPAttrModifier del: aAttribute}.
   req := connection modify: (aDN, ',', self baseDN) with: ops.
   req wait.
```

Nous pouvons maintenant supprimer l'entrée 'stage' pour l'UID 'dupont' :

```
| ann |
ann := Ldap new connect.
ann deleteAttribute: 'stage' from: 'UID=dupont'.
ann disconnect.
```

Il est également possible d'effacer un attribut en fonction de sa valeur. Ecrivons une méthode deleteAttribute:value:from: qui réalise cela :

```
Ldap>>deleteAttribute: aAttribute value: aValue from: aDN | req ops |
   ops := { LDAPAttrModifier delFrom: aAttribute values: { aValue }.}.
   req := connection modify: (aDN, ',', self baseDN) with: ops.
   req wait.
```

L'exemple suivant supprime l'attribut 'givenName' si sa valeur est égale à 'Alfred'. La recherche est basée ici sur un DN constitué d'un UID utilisateur et du DN de base.

```
| ann |
ann := Ldap new connect.
ann deleteAttribute: 'givenName' value: 'Alfred' from: 'UID=dupont'.
ann disconnect.
```

Créer une entrée dans l'annuaire

Si nous voulons ajouter un nouvel élément à notre annuaire, il nous faut créer une nouvelle entrée. Nous allons créer une méthode createEntry:with: qui reçoit en paramètre un UID ainsi qu'une collection contenant les informations devant être insérées dans la nouvelle entrée.

```
Ldap>>createEntry: aUID with: aCollection
   | req |
   req := self connection addEntry: aUID attrs: aCollection.
   req wait.
```

Nous pouvons alors ajouter une nouvelle entrée de type 'inetOrgPerson' pour l'utilisateur 'dupont'. Les attributs 'cn' et 'sn' sont fixés à 'dupont' également.

Stéf ►*c'est quoi cn et sn?*◄

```
   | ann attrs |
   attrs := Dictionary new
            at: 'objectClass'
            put: (OrderedCollection new add: 'inetOrgPerson'; yourself);
            at: 'cn' put: 'dupont';
            at: 'sn' put: 'dupont'; yourself.
   ann := Ldap new connect.
   ann createEntry: ('uid=dupont,' , ann baseDN) with: attrs.
   ann disconnect.
```

Supprimer une entrée dans l'annuaire

Effacer une entrée dans l'annuaire est simple et rapide. Attention aux mauvaises manoeuvres ! Pour cela, nous pouvons définir une méthode deleteEntry: recevant en paramètre l'UID de l'entrée qui sera effacée.

```
Ldap>>deleteEntry: aUID

   | req |
   req := self connection delEntry: (aUID , ',' , self baseDN).
   req wait.
```

Essayons notre méthode en supprimant dans l'annuaire, l'entrée précédemment créée pour l'utilisateur 'dupont' :

```
| ann |
ann := Ldap new connect.
ann deleteEntry: 'uid=dupont'.
ann disconnect.
```

1.3 Présentation des classes principales

1.4 Conclusion

Nous venons de découvrir les principales fonctionnalités de LDAPlayer et vous disposez maintenant des outils essentiels pour exploiter un annuaire LDAP avec Pharo. Vous pouvez intéroger un annuaire, modifier ou effacer des entrées selon vos besoins.

Liens utiles http://www.openldap.org/, http://en.wikipedia.org/wiki/LDAP http://fr.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol