Expérience à Lille1

Je vais vous expliquer quelle a été notre démarche et comment elle a évolué en fonction de la connaissance acquise progressivement sur Grouper (et nuxeo en parallèle).

Version grouper utilisée : grouper 1.6.3 disponible depuis le 4 janvier 2011.

Démarche entamée fin mars 2011 - arrêt en juin pour installation nuxeo - redémarrage en juillet - prévision de mise en production : septembre 2011. Mise en production réelle : 4 novembre 2011.

Démarche

Nous avions 2 motivations pour tester Grouper :

1. remplacer une portlet du portail qui nous permet depuis plusieurs années de gérer les droits d'accès aux applications dans le portail : portlet ldaproles (voir incubateur esup). Nous n'avions pas réussi à la migrer vers un esup 3.2.1 de test.
2. préparer le passage vers la gestion des documents via nuxeo qui nécessite une gestion des groupes améliorée.

Par contre, nous avions les contraintes suivantes :

1. impossible de fédérer l'établissement actuellement autour de ce projet (voir si le DSI qui arrivera en octobre fera changer cela). Au départ, il n'était même pas question de publier dans la branche ou=groups. Et au fur et à mesure de l'avancée du projet Grouper et de sa meilleure connaissance, il est apparu nécessaire de devoir publier dans cette branche ou=groups (notamment pour nuxeo). Nous en avons eu l'autorisation à condition de ne pas gêner les autres usagers. Donc, ne publier que ses propres groupes dans la branche ou=groups sans perturber ceux déjà existants (et nombreux).
2. la portlet ldaproles met à jour un attribut dans la branche ou=people : attribut "ustlRole". Il faut que le passage vers Grouper permette de remplacer cette fonctionnalité.
3. générer automatiquement à partir de Grouper le fichier "pagsGroupStoreConfig.xml" du portail esup et ceux d'infoglue (live_pags et cms_pags). Le lien direct Grouper-esup-portail n'est pas envisagé dans un premier temps, étant donné le projet de restructuration complète et passage d'une esup 2.5.1 en un esup 3.2.4 prévue initialement en juillet 2011 actuellement reportée en début d'année 2012.

Choix d'organisation

S'affranchir des groupes esup-portail

J'ai souhaité ne pas devoir me calquer sur l'organisation (ou plutôt la pseudo-organisation...) des groupes de notre portail actuel. Les groupes esup-portail étant finalement davantage utilisés pour la publication des fragments et des canaux/portlets, il semblait plus pertinent de créer une structure indépendante et propre à Grouper.

Une organisation évolutive

Mais surtout, il fallait une organisation orientée usagers c'est-à-dire lisible par eux. Il fallait également qu'elle soit évolutive. Nous savions que nous "partirions au fil de l'eau" c'est-à-dire que ce seront les usages qui feront progresser la gestion des groupes et particulèrement les usages sur nuxeo. Il fallait donc que la structure retenue puisse répondre aux demandes ponctuelles (et souvent urgentes !) de création de nouveaux groupes.

Ne pas déléguer la création et la suppression des groupes

Nous avons choisi de ne pas déléguer la création et la suppression des groupes dans un premier temps et sur aucune branche de la structure Grouper choisie, ceci pour garantir une cohérence de cette structure.

Par contre, le peuplement des groupes devait être délégué, à l'image de ce que faisait la portlet ldaproles (voir ci-dessus). Il fallait également bien identifier la gestion des groupes. On abandonne le vocable "gestion des rôles" par "gestion des groupes". Cela paraît anodin mais nous nous sommes aperçu que nos usagers avaient beaucoup de difficultés à comprendre cette gestion des rôles...

Schéma prévisionnel de l'arborescence Grouper

Voir fichier joint.

Il a fallu intégrer les groupes déjà existants : ceux déclarés dans le PAGSGroupStoreConfig.xml d'esup-portail et ceux déclarés dans des fichiers similaires dans infoglue.

Quels éléments Grouper installés ?

Grouper-API - Grouper-UI - ldappcng - grouper-ws en prévision du test d'ESCO-Grouper.

Pour l'instant, je n'ai pas installé ESCO-Grouper mais la question se pose sérieusement. En effet, dans un premier temps, je pensais offrir seulement LiteUi aux usagers responsables du peuplement des groupes (77 environ). Cet outil est suffisant pour eux, il convient très bien.

Cependant, en installant nuxeo et en expérimentant la vision des groupes Grouper dans nuxeo depuis ldap (nuxeo passe par la branche ou=groups du ldap pour afficher les groupes), je m'aperçois d'un manque pour l'usager : il ne voit pas la structure des groupes. Il faut donc qu'il connaisse le nom du groupe (cn du ldap, cf mail sur la liste esup-ecm) pour attribuer les droits sur ses documents. Dans nuxeo, il ne peut faire qu'une recherche à plat. Par conséquent, je m'interroge pour lui donner un accès en lecture à la structuration des groupes Grouper (même ceux qu'il ne voit pas encore dans nuxeo), afin qu'il puisse mieux "choisir" le groupe convenant à sa situation. Point à revoir car le plugin Shibboleth dans nuxeo peut changer cela, à compléter une fois la connaissance de Nuxeo plus avancée.

La question est de savoir si cette vue sera depuis nuxeo (dans l'idéal aussi mais peut-être pas forcément dans un 1er temps) ou si cette vue sera depuis Grouper : LiteUi ou ESCO-Grouper. L'idée est de faire au plus simple dans un 1er temps, surtout pour l'installation/paramétrage car le temps passe et il faut mettre en production Grouper et faire "démarrer" nuxeo. A suivre. Une autre piste se dégage également : voir les groupes Grouper depuis esup-portail. Un projet est en cours au niveau de uPortal pour mieux intégrer Grouper dans uPortal. Je suis de près ce projet avec un développeur uPortal.

L'arrivée de la version 2 de Grouper (septembre 2011) bouscule un peu les choses... LiteUi a notamment été enrichi, à tester.

Trucs et Astuces

Voici quelques éléments à prendre en compte pour s'organiser :

1. il est possible de typer les groupes dans Grouper. A un type, on peut associer un ou plusieurs attributs de type string et une ou plusieurs listes. J'ai pu facilement faire des listes de personnes et/ou de groupes. Par contre, pour faire des listes d'autre chose, ce doit être possible mais cela m'a semblé difficile à mettre en oeuvre : voir avec SubjectAPI (https://spaces.internet2.edu/display/Grouper/Subject+API).
2. Par contre, impossible de nommer un attribut du même nom même sur des types de groupes différents. Il faut donc bien penser ses types de façon à ne pas avoir à réutiliser un même attribut : il faut factoriser. Notamment, cela est important quand on publie dans la branche ou=groups du ldap. J'ai été amenée par ce fait à découper les types déclarés.

   - Par exemple, j'ai un type "GroupLDAP" qui pointe les groupes à publier dans le ldap branche ou=groups et rassemble les attributs communs à tous les groupes ldap de la branche ou=groups. Il porte 3 attributs : cnLDAP, ownerLDAP, PubLDAPGroup. En effet, le cn et le owner sont des attributs communs à tous les groupes à publier dans ou=groups, l'attribut PubLDAPGroup (valeurs actuelles : "aec" ou "gof", cf ci-après explications sur ces types) me permet de publier l'un ou l'autre des types de groupes lors de la synchronisation ldappcng (filtre de publication).
   - J'ai ensuite déclaré 3 types : type "AEC", type "supannGroupe" et type "GroupOfURLs" "qui me permettent de publier des groupes avec des objectClass différents dans le ldap mais aussi de déclarer des attributs spécifiques à chacun de ces types.
   - Par exemple, le type "AEC" supporte un attribut "ustlPresident",
   - le type "supannGroupe" supporte 3 attributs : supannGroupeAdminDN, supannGroupeLecteurDN et supannGroupeDateFin.
   - Le type GroupOfURLs permet de déclarer des groupes dynamiques dans la branche ou=groups. Je n'ai déclaré qu'un seul attribut pour ce type : memberURL. La valeur de cet attribut est un filtre ldap, par exemple : ldap://nom-serveur.univ-xxx.fr/ou=people,dc=univ-xxx, dc=fr??sub?(ustlDepartement=SUAIO). De ce fait, dans le ldap dans la branche ou=groups, ce groupe est déclaré dynamiquement : aucun membre n'apparaît en visualisation dans l'annuaire mais les applications qui appellent ce groupe peuvent en déduire "à la volée" ses membres. C'est ce que fait Nuxeo par exemple.

   - Un autre exemple pour bien penser ses types de groupes et les attributs associés : une partie des groupes pags d'esup-portail et d'infoglue ont une partie commune : l'attribut "ustlRole" à publier dans le ldap, attribut qui permet de rassembler des personnes sur simple déclaration du/des responsables de ce groupe (exemple fonctionnel : liste des ACMO ou encore liste des chargés de mission recherche).
   - J'ai déclaré 2 types :
     - un type "PeopleLDAP" : permet de sélectionner pour ldappcng les groupes qui supportent un attribut à publier dans la branche ou=people. Ce type comporte un seul attribut : "PubLDAPPeople" qui donne le nom de l'attribut ldap qui sera valorisé. Pour l'instant, je le valorise qu'à une seule valeur : "ustlRole" mais je pourrais imaginer d'utiliser ce type pour mettre à jour d'autres attributs du ldap (si l'administrateur du ldap m'en donnait l'autorisation). C'est cet attribut qui permet de positionner le filtre ldappcng dans le fichier ldappc-resolver.xml.
     - un type "PAGSRole" : permet de déclarer la valeur de cet attribut "ustlRole" pour ce groupe (par exemple "ACMO-acteurs" ou "IntranetCMR") , qu'il soit pags-esup-portail, pags-infoglue voire ni l'un ni l'autre (cas d'une application qui vient lire directement dans le ldap, comme la portlet annuaire pour avoir la liste des correspondants antivirus, logiciels ou gestionnaire de parc par exemple).
   - Ensuite, j'ai un type "PortailEsup" qui donne le esup-groupName et le esup-groupKey pour esup-portail et un type "Infoglue" qui donne des attributs spécifiques à Infoglue.
   - J'ai également un type "PAGS" qui donne la valeur du selectionTest pour esup-portail, à utiliser uniquement pour les groupes pags du portail qui sont déclarés à partir d'une selectionTest et d'attributs "classiques" du ldap (comme objectClass ou eduPersonAffiliation par exemple),

   ---

3. Attention aux noms des attributs que vous ajoutez : ils peuvent entrer en collision avec des propriétés internes à Grouper. Par exemple, j'avais utilisé un nom d'attribut "groupName" pour un type "PortailEsup" mais il s'affichait mal car Grouper utilise pour ses propres besoins ce nom... Je l'ai donc renommé en "esup-groupName". Donc, de façon générale, pensez à utiliser des noms d'attributs spécifiques (autre exemple: j'ai utilisé "ownerLDAP" plutôt que "owner" pour un attribut).
4. Attention au paramètre Required pour les attributs de Type. Je dois modifier mes déclarations de types pour les rendre tous Not Required car impossible d'enlever le type de groupe quand un des attributs est required et que cet attribut avait déjà été renseigné. Cf bug jira : https://bugs.internet2.edu/jira/browse/GRP-620
5. Si vous choisissez de publier dans le ldap et selon votre degré d'autonomie dans la publication ldap, vous passerez beaucoup de temps à paramétrer le ldappcng.xml. De là à le comprendre complètement... Mais il s'avère que son paramétrage influera également sur l'organisation de vos groupes. Dans mon cas, j'ai été amenée à modifier les types de groupes initialement envisagés de nombreuses fois, au fur et à mesure de mon implémentation ldappcng (notamment, le filtrage des groupes à publier qui utilise la valeur d'un attribut et non le typage lui-même des groupes cf ci-après).

6. Ne chargez pas tous vos groupes au départ ! Surtout, je vous conseille de bien réfléchir à votre structure et de faire des tests de publications dans le ldap avant de mettre votre structure complète dans Grouper. J'ai été amenée tellement de fois à modifier mes groupes que j'ai été contente de n'avoir que quelques groupes à surveiller...
7. Pour la publication des groupes dans la branche ou=groups du ldap via ldappcng, il est possible maintenant de ne publier que partiellement (authoritative=false dans ldappcng.xml) donc ne pas écraser les groupes déjà existants. Il est également possible de ne publier que certains groupes de Grouper dans cette branche ou=groups. Pour cela, il existe des filtres à positionner (dans ldappc-resolver.xml) mais ces filtres ne permettent pas de sélectionner sur le type de groupe mais sur la valeur exacte d'un attribut ou sur une branche de la hiérarchie Grouper (cf http://www.internet2.edu/grouper/release/1.5.0/doc/api/edu/internet2/middleware/grouper/shibboleth/filter/BaseGroupQueryFilter.html). Par conséquent, j'ai créé un peu "artificiellement" (car dans le fond, un peu en double par rapport au type lui-même) un attribut au type et lui ai affecté une valeur pour pouvoir sélectionner tous les groupes de ce type. Mais finalement, cela m'a servi quand j'ai cherché à publier dans le ldap 2 types de groupes différents (c'est-à-dire qui n'ont pas les mêmes attributs dans le ldap et notamment pas les mêmes objectClass comme les types "AEC" ou "GOF", cf ci-dessus). En effet, j'ai alors valué différemment cet attribut du type selon les spécificités du groupe dans le ldap. Par exemple, j'ai des groupes de type "AEC" (Avancement des Enseignants Chercheurs) et des groupes de type "GroupOfURLs" (ceux pour nuxeo). Pour publier ces 2 types de groupes dans le ldap dans la branche ou=groups en sachant qu'ils n'y auront pas les mêmes attributs particulièrement pas les mêmes objectClass, j'ai ajouté un type "GroupLDAP" qui dispose entre autres d'un attribut "PubLDAPGroup" que je valorise (par l'interface ou par groupershell) à la chaîne de caractère "AEC" quand il s'agit d'un groupe de type "AEC" et à "GOF" pour les groupes de type "GroupOfURLs". Mais attention, je ne peux plus publier tous les objets en même temps, je dois utiliser le paramètre entityName pour sélectionner ce que je veux publier : member ou groupAEC ou groupGOF.

Contributions techniques

Adaptation de LiteUI

A noter : J'ai dû rentrer dans le code de LiteUI fin août car je n'obtenais pas correctement les accents ce qui était bloquant pour la mise en production (normalement, résolu depuis : https://spaces.internet2.edu/display/Grouper/Grouper+UI+internationalization). Du coup, j'ai utilisé les connaissances acquises sur le fonctionnement du code LiteUI pour apporter les modifications rapides que je souhaitais. Je vous les livre, elles ne suivent pas forcément ce qui est écrit sur le wiki ici : https://spaces.internet2.edu/display/Grouper/Customising+the+Grouper+UI - Si un autre établissement peut apporter des rectifications/conseils/éclairages, ils sont les bienvenus !

Attention, problème avec IE9 pour le paging : je vais donner l'astuce de cliquer sur le bouton "affichage de compatibilité" juste après la saisie de l'url, à côté de la loupe.

1. Franciser l'interface

J'ai francisé l'interface en modifiant le fichier [chemin d'installation de Grouper-UI]/conf/resources/grouper/nav.properties : toutes les clés simpleMembershipUpdate (je n'ai francisé que LiteUI car AdminUI est vraiment orienté administrateur Grouper donc par des informaticiens). Voir le fichier joint : listesModificationsFrancaises.txt Il faut intégrer ces clés dans votre fichier nav.properties et commenter celles correspondantes en anglais. --- Ou mieux : créer un fichier nav.properties français avec déjà ces clés, voir comment il peut surcharger celui en anglais (où le mettre pour qu'il soit pris en compte ?) --

Attention au tri avec les accents, j'ai également modifié un source pour faire le tri correctement, voir sur la liste grouper-users nov 2011 (modif du SubjectSortWrapper.java)

2. Ne pas afficher le lien vers AdminUI + réduire les détails sur le groupe et les membres

ne pas afficher le lien vers AdminUI : modifier la jsp simpleMembershipUpdateMain.jsp (dans ../grouper-ui/webapp/WEB-IF/grouperUi/templates) en supprimant :

```
<grouper:message valueTooltip="${simpleMembershipUpdateContainer.text.
viewInAdminUiTooltip}"  value="${simpleMembershipUpdateContainer.text.viewInAdminUi}"  />
```

Pour réduire les détails sur le groupe, utiliser les propriétés du fichier media.properties, par exemple pour Lille1 :

```
#if the id row should show on the screen by default
simpleMembershipUpdate.showIdRowByDefault=false

#if the id path row should show on the screen by default
simpleMembershipUpdate.showIdPathRowByDefault=false

#if the alternate id path row should show on the screen by default
simpleMembershipUpdate.showAlternateIdPathRowByDefault=false

#if the uuid row should show on the screen by default
simpleMembershipUpdate.showUuidRowByDefault=false
```

pour ne pas afficher tous les détails sur les personnes (notamment l'uid qui peut être une donnée sensible), modifier simpleMembershipUpdateSubjectDetails.jsp :

```
...
        <%-- this is a map of key value pairs Map.Entry --%>
        <c:forEach var="attribute" items="${simpleMembershipUpdateContainer.subjectDetails}">
              <%-- this could be misleading, but put in some common labels from nav.properties and tooltips

          subject.summary.displayName=Path
          subject.summary.extension=ID
          subject.summary.createTime=Created
          subject.summary.createSubjectId=Creator ID (entity ID)
          subject.summary.createSubjectType=Creator entity type
          subject.summary.modifyTime=Last edited
          subject.summary.modifySubjectId=Last editor ID (entity ID)
          subject.summary.modifySubjectType=Last editor entity type
          subject.summary.subjectType=Entity type

              --%>
              <c:set var="subjectSummaryNavKey" value="subject.summary.${attribute.key}"  />
              <%-- change some common ones so they dont overlap --%>
              <c:if test="${simpleMembershipUpdateContainer.subjectForDetails.type.name == 'group'}">
                <c:set var="subjectSummaryNavKey" value="subject.summary.group.${attribute.key}"  />
                <c:set var="lacleG" value="${attribute.key}" />
                <c:set var="afficheG" value="0" />
              </c:if>
              <c:choose>
                <c:when test="${! empty navNullMap[subjectSummaryNavKey]}">
                   <c:if test="${lacleG == 'displayExtension'}">
                      <tr>
                      <td>
                       <c:out value="Nom" />
                       <c:set var="afficheG" value="1" />
                      </td>
                   </c:if>
                   <c:if test="${lacleG == 'extension'}">
                      <tr>
                      <td>
                       <c:out value="Nom court" />
                       <c:set var="afficheG" value="1" />
                      </td>
                   </c:if>
                   <c:if test="${lacleG == 'displayName'}">
                      <tr>
                      <td>
                       <c:out value="Adresse longue" />
                       <c:set var="afficheG" value="1" />
                      </td>
```

```
                  </c:if>
                  <c:if test="${lacleG == 'name'}">
                     <tr>
                     <td>
                      <c:out value="Adresse courte" />
                      <c:set var="afficheG" value="1" />
                     </td>
                  </c:if>
               </c:when>
               <c:otherwise>
                  <c:set var="lacle" value="${attribute.key}" />
                  <c:set var="affiche" value="0" />
                  <c:if test="${lacle == 'screenLabel'}">
                       <tr>
                       <td>
                       <c:out value="Nom" />
                       <c:set var="affiche" value="1" />
                       </td>
                  </c:if>
                  <c:if test="${lacle == 'mail'}">
                       <tr>
                       <td>
                       <c:out value="Courriel" />
                       <c:set var="affiche" value="1" />
                       </td>
                  </c:if>
                  <c:if test="${lacle == 'supannAffectation'}">
                       <tr>
                       <td>
                       <c:out value="Affectation ou Formation" />
                       <c:set var="affiche" value="1" />
                       </td>
                  </c:if>
                  <c:if test="${lacle == 'supannCivilite'}">
                       <tr>
                       <td>
                       <c:out value="Etat civil" />
                       <c:set var="affiche" value="1" />
                       </td>
                  </c:if>
               </c:otherwise>
            </c:choose>
            <c:if test="${affiche == '1'}">
               <td>
                    <c:out value="${attribute.value}" />
               </td>
               </tr>
            </c:if>
            <c:if test="${! empty navNullMap[subjectSummaryNavKey]}">
               <c:if test="${afficheG == '1'}">
               <td>
                    <c:out value="${attribute.value}" />
               </td>
               </tr>
            </c:if>
            </c:if>

         </c:forEach>
      </table>
    </div>
  </div>
</div>
<!-- End: simpleMembershipUpdateSubjectDetails.jsp -->
```

3. Afficher le type de personne (étudiant, personnel, enseignant, etc)

J'ai modifié la méthode "convertSubjectToLabel" dans le fichier GrouperUiUtils.java (sous ...java/src/.../grouper/ui/util) :

```
public static String convertSubjectToLabel(Subject subject) {
    String label = null;
    // ajout bw
    String typeMember = null;
    String typeMemberF = null;
    HashMap franciserType = new HashMap();
    franciserType.put("student","étudiant");
    franciserType.put("faculty","enseignant");
    franciserType.put("employee","personnel");
    franciserType.put("affiliate","affilié");
    franciserType.put("researcher","chercheur");
    franciserType.put("retired","retraité");
    franciserType.put("emeritus","emérite");

....

    typeMember = subject.getAttributeValue("eduPersonPrimaryAffiliation");
    if (franciserType.get(typeMember)==null) {
        typeMemberF = typeMember;
     }
    else {
       typeMemberF = franciserType.get(typeMember).toString();
    }
    if (!StringUtils.isBlank(typeMember)) {
         label = label + " - " + typeMemberF;
     }
    return label;
  }
```

Ne pas oublier de déclarer l'attribut concerné dans le sources.xml côté grouper-API : ici, eduPersonPrimaryAffiliation

Nota : si vous souhaitez également avoir cette information dans AdminUI, il faut modifier la jsp : subjectView.jsp ainsi que le fichier media.properties de Grouper-UI pour déclarer la valeur de subject.display.bw.lille\:ldap (ligne subject.display.bw.lille1\:ldap=eduPersonPrimaryAffiliation dans media.properties), "lille1:ldap" représentant la source déclarée dans le fichier sources.xml de Grouper-API :

```
<tiles:importAttribute ignore="true"/><c:set var="attrKey2" value="*subject.display.bw.${viewObject.source.id}"
/><c:set var="attrKey" value="subject.display.default"/><c:set var="attrKeyGroup" value="subject.display.g"/>
<%--<tiles:importAttribute ignore="true"/><c:set var="attrKey2" value="*subject.display.bw.${viewObject.source.
id}"/><c:set var="attrKey" value="*subject.display.${viewObject.source.id}"/><c:if test="${empy mediaMap
[attrKey]}"><c:set var="attrKey" value="subject.display.default"/></c:if> --%>
<c:if test="${viewObject.isGroup}"><grouper:tooltip key="group.icon.tooltip"/>
    <img src="grouper/images/group.gif" alt="Group"
    /><c:out value="${viewObject[mediaMap[attrKeyGroup]]}" /></c:if><c:if test="${empty inLink}"
><span${viewObject.subjectType}"/>Subject"></c:if><c:out value="${viewObject[mediaMap[attrKey]]}" /> [<c:out
value="${viewObject[mediaMap[attrKey2]]}" />]<c:if test="${empty inLink}"></span></c:if>
```

4. Modifier l'import/export du menu "Plus de choix..."

4.1 l'import

Je voulais que l'import se base sur les courriels universitaires.. J'ai donc modifié dans sources.xml le searchSubjectByIdentifier en indiquant rechercher sur l'attribut mail :

```
<search>
        <searchType>searchSubjectByIdentifier</searchType>
        <param>
            <param-name>filter</param-name>
            <param-value>
                (&amp;amp; (mail=%TERM%) (objectClass=supannPerson))
            </param-value>
        </param>
        <param>
etc
```

J'ai également masqué la demande de la source pour l'import car par défaut, à Lille1, nous n'avons que le ldap en sources de subject : dans simpleMembershipUpdateImport.jsp, j'ai commenté la partie concernée :

```
<%-- <tr>
             <td><grouper:message  value="${simpleMembershipUpdateContainer.text.importAvailableSourceIds}"  /><
/td>
             <td><%-- ${fn:escapeXml(simpleMembershipUpdateContainer.sourceIds) }
               <table>
                 <tr>
                   <th>sourceId</th><th>Source name</th>
                 </tr>
                 <c:forEach items="${contextContainer.sources}" var="source">
                   <tr>
                     <td><c:out value="${source.id}" escapeXml="true"/></td>
                     <td><c:out value="${source.name }" escapeXml="true"/></td>
                   </tr>
                 </c:forEach>
               </table>
             </td>
           </tr> --%>
```

4.2 l'export

Je ne souhaitais plus afficher l'export des identifiants (information sensible). J'ai commenté la partie concernée dans le fichier SimpleMebershipUpdateMenu.java dans .../java/src/.../grouperUI/serviceLogic) :

```
/** else if (StringUtils.equals(menuItemId, "exportSubjectIds")) {
        guiResponseJs.addAction(GuiScreenAction.newAlertFromJsp(
            "/WEB-INF/grouperUi/templates/simpleMembershipUpdate/simpleMembershipUpdateExportSubjectIds.jsp"));
      } **/
      else if (StringUtils.equals(menuItemId, "exportAll")) {
        guiResponseJs.addAction(GuiScreenAction.newAlertFromJsp(
            "/WEB-INF/grouperUi/templates/simpleMembershipUpdate/simpleMembershipUpdateExportAll.jsp"));

      } else if (StringUtils.equals(menuItemId, "import")) {
        guiResponseJs.addAction(GuiScreenAction.newDialogFromJsp(
            "/WEB-INF/grouperUi/templates/simpleMembershipUpdate/simpleMembershipUpdateImport.jsp"));
      } else {
        throw new RuntimeException("Unexpected menu id: '" + menuItemId + "'");
      }
```

et je n'exporte que le nom et le mail, à modifier dans media.properties :

```
#simpleMembershipUpdate.exportAllSubjectFields=sourceId, screenLabel, entityId, name, description
simpleMembershipUpdate.exportAllSubjectFields=name, mail
```

Voir sur la liste grouper-users nov 2011 comment exporter un attribut ldap multivalué (modif de SimpleMembershipUpdateImportExport.java mais sera mis dans les prochaines versions).

Paramétrage de ldappcng

Un peu compliqué le paramétrage de cette publication. Les 4 fichiers essentiels sont ceux ci-dessous précisés.

0. sources.xml

C'est un fichier que vous devez déjà connaître car il permet de déclarer la source de vos données (entity, groupes notamment), vous avez dû le paramétrer au moment d'installer Grouper-API. Je le mets en point 0 car il ne fait pas partie réellement du paramétrage de ldappcng mais il faut le garder en mémoire et il faut aussi savoir qu'il est référencé à un moment dans le ldappc-resolver.xml, notamment pour l'identifiant ldap utilisé (champs "uid" du ldap à Lille1).

1. ldappc.properties

Vous allez définir dans ce fichier quelques propriétés, notamment l'adresse de votre annuaire ldap. Quelques propriétés référencées dans les fichiers ci-après sont définies dans ce fichier. Je vous pointe les principales pour comprendre l'exemple de lille1 :

```
# Base DN for members
peopleOU=ou=people,dc=univ-lille1,dc=fr
# Base DN for members etudiants
etudiantsOU=ou=etudiants,dc=univ-lille1,dc=fr
# Base DN for groups
groupsOU=ou=groups,dc=univ-lille1,dc=fr

groupObjectClassAEC=ustlComAdHocAECGroupe
groupObjectClassGOF=groupOfURLs
```

2. ldappcng.xml

Celui-là décrit les objets qui sont à publier dans le ldap : il a beaucoup de références au contenu du fichier ldappc-resolver.xml, il faut donc les lire en parallèle pour pouvoir comprendre comment cela se passe. Je vous ai joint les 2 fichiers et vous conseille de lire ma "prose" au paragraphe "Trucs et astuces" pour comprendre comment ils appliquent la publication dans le ldap ainsi que mon schéma de publication joint.

Je vais quand même vous aider à le décrypter sur 2 types d'objets différents publiés :

A- pour les groupes destinés à l'application qui gère l'avancement des enseignants-chercheurs dans la branche ou-groups,

B- pour les membres des groupes pour lesquels le champ multivalué "ustlRole" de notre ldap doit être mis à jour dans la branche ou=people

A- groupes Avancement des Enseignants-chercheurs :

A savoir : dans Grouper, j'ai déclaré un type de groupe "GroupLDAP" qui indique si le groupe Grouper est à publier dans la branche ou=groups du ldap. Il porte un attribut "PubLDAPGroup" de type string et dont les valeurs possibles, par convention, sont "AEC" pour Avancement des Enseignants-chercheurs ou "GOF" pour GroupOfUrls. J'ai également créé un type de groupe "AEC" (Avancement des enseignants-chercheurs) qui apporte un attribut "ustlPresident". Enfin, j'ai également créé un type de groupe "supannGroupe" qui apporte 3 attributs : supannGroupeAdminDN, supannGroupeDateFin, supannGroupeLecteurDN

Je vais donc demander au module ldappcng via ce fichier ldappcng.xml de publier tous les groupes qui sont de type "AEC" de la façon suivante :

```
<object id="groupAEC" authoritative="false">
      <identifier ref="group-dnAEC" baseId="${groupsOU}">
        <identifyingAttribute name="objectClass" value="${groupObjectClassAEC}" />
      </identifier>
```

L'id "groupAEC" sera utilisé au moment de l'ordre de publication dans le ldap :

```
bin/gsh.sh -ldappcng -bulkSync -entityName groupAEC
```

Le mot "authoritative" permet d'indiquer au module ldappcng qu'il n'est pas le seul à publier dans la branche ou=groups dans le ldap et donc de ne faire qu'ajouter ou modifier les groupes déclarés dans Grouper. Il ne supprimera pas les groupes non présents dans Grouper. -- D'ailleurs, je me demande s'il supprimerait un groupe créé dans Grouper ? Je n'ai pas essayé, à étudier. --

La partie "identifier" permet d'indiquer quels sont les groupes qui vont être concernés par cette publication : quels groupes Grouper mais aussi quels groupes ldap :

- ref="group-dnAEC" renvoie à la partie concernée dans le fichier ldappc-resolver.xml voir ci-après, c'est lui qui va expliciter le lien entre le groupe Grouper et le groupe ldap ou=groups.
- identifyingAttribute indique quels sont les groupes ldap à sélectionner, ici, ceux dont l'objectClass est égal à "ustlComAdHocAECGroupe" (valeur de la propriété dans ldappc.properties)

Ensuite, ce fichier va déclarer quels sont les attributs à publier dans le ldap pour ce groupe et comment obtenir sa valeur :

```
<attribute name="objectClass" ref="objectClassAEC" />
<attribute name="cnLDAP" ref="cnLDAP1"/>
<attribute name="description" />
<attribute name="owner" ref="ownerScript" />
<attribute name="ustlPresident" ref="ustlPresidentScript"/>
<attribute name="supannGroupeDateFin" />
```

```
<references name="member" emptyValue="" >
    <reference ref="members-lille1:ldap" toObject="member" />
</references>
<references name="supannGroupeAdminDN" emptyValue="" >
    <reference ref="admin" toObject="member" />
</references>
```

Il y a 2 types d'attribut publié : un attribut n'ayant qu'une seule valeur "<attribute.." et les attributs qui représentent plusieurs valeurs "<references..".

La valeur donnée au paramètre "name" représente soit le nom du champs ldap (exemple ici "ObjectClass") soit le nom de l'attribut du groupe dans Grouper (par exemple ici "cnLDAP"). La valeur du paramètre "ref" donne l'id du <resolver:AttributeDefinition id=...> déclaré dans le fichier ldappc-resolver.xml, c'est lui qui déclare quelle est la valeur précise à mettre dans le champ ldap et aussi le nom du champ ldap (dans sourceAttributeID) s'il n'est pas le même que celui passé dans le paramètre "name".

Par exemple, ici, objectClass est le nom de l'attribut ldap à renseigner, objectClassAEC est l'id du <resolver:AttributeDefinition..> que vous allez retrouver dans le fichier ldappc-resolver.xml

cnLDAP est le nom de l'attribut du groupe qui stocke la valeur à passer au ldap, cnLDAP1 est l'id du <resolver:AttributeDefinition..> que vous allez retrouver dans le fichier ldappc-resolver.xml et vous verrez plus bas dans le fichier ldappc-resolver.xml que l'attribut ldap à renseigner avec cette valeur est le cn (indiqué dans l'attribut "sourceAttributeID" du <resolver:AttributeDefinition..>)

B- champ multivalué "ustlRole", branche ou=people du ldap

A savoir : les membres des groupes Grouper qui sont de type "PAGSRole" vont être affecté de la valeur de l'attribut "ustlRole" du groupe Grouper dans leur attribut ldap "ustlRole" :

```
<object id="member">
      <identifier ref="member-dn" baseId="${peopleOU}">
        <identifyingAttribute name="objectClass" value="eduPerson" />
      </identifier>
         <attribute name="ustlRole" ref="memberIsMemberOf" />
</object>
```

L'id "member" sera utilisé au moment de l'ordre de publication dans le ldap :

```
bin/gsh.sh -ldappcng -bulkSync -entityName member
```

La partie "identifier" permet d'indiquer quels sont les personnes qui vont être concernés par cette publication : quelles personnes côté Grouper (ref "member-dn") mais aussi quelles personnes côté ldap groupes ldap :

- ref="member-dn" renvoie à la partie concernée dans le fichier ldappc-resolver.xml voir ci-après, c'est lui qui va expliciter le lien entre la personne côté Grouper et la personne ldap ou=people.
- identifyingAttribute indique quels sont les personnes ldap à sélectionner, ici, ceux dont l'objectClass est égal à "eduPerson".

Ensuite, l'attribut à modifier dans le ldap : ici "ustlRole" est indiqué ainsi que comment obtenir la valeur (ref="memberIsMemberOf à détailler dans le fichier ldappc-resolver.xml ci-après).

3. ldappc-resolver.xml

Je détaille uniquement les parties référencées ci-dessus.

A- groupes Avancement des Enseignants-chercheurs :

```
<resolver:AttributeDefinition id="group-dnAEC" xsi:type="ldappc:LdapDnPSOIdentifier"
                  structure="${DNstructure}" sourceAttributeID="cnLDAP" rdnAttributeName="cn"
base="${groupsOU}">
   <resolver:Dependency ref="GroupDataConnectorAEC" />
</resolver:AttributeDefinition>
```

Selon le type d'AttributeDefinition, vous n'aurez pas les mêmes paramètres à renseigner. Ici, nous indiquons que nous définissons la référence à group-dnAEC (cf fichier ldappcng.xml), qu'il s'agit de l'attribut "cnLDAP" du groupe Grouper à prendre en compte pour alimenter l'attribut "cn" de la branche ou=groups du ldap (cf ldappc.properties). <resolver:Dependency..> indique quels groupes Grouper et quels membres de ces groupes vont être pris en compte :

```
<resolver:DataConnector id="GroupDataConnectorAEC" xsi:type="grouper:GroupDataConnector">
        <grouper:GroupFilter xsi:type="grouper:ExactAttribute" name="PubLDAPGroup" value="AEC" />
    <grouper:Attribute id="members:immediate" />
    <grouper:Attribute id="members:immediate:supannGroupeAdminDN" />
</resolver:DataConnector>
```

Ici, j'indique que je ne prends que les groupes Grouper dont l'attribut nommé "PubLDAPGroup" a la valeur "AEC" et je vais utiliser la liste de ses membres direct ("members:immediate") et la liste des membres contenus dans l'attribut de type list "supannGroupeAdminDN".

Ensuite, je vais prendre l'exemple sur 2 attributs à renseigner pour ce groupe :

```
<resolver:AttributeDefinition id="objectClassAEC" xsi:type="ad:Simple" sourceAttributeID="objectClass">
    <resolver:Dependency ref="StaticDataConnectorAEC" />
</resolver:AttributeDefinition>
```

Ici, j'indique que l'attribut référencé objectClassAEC dans le ldappcng.xml consiste à mettre à jour l'attribut ldap "objectClass" avec des valeurs statiques indiquées dans "StaticDataConnecterAEC" :

```
<resolver:DataConnector id="StaticDataConnectorAEC" xsi:type="dc:Static">
    <dc:Attribute id="objectClass">
      <dc:Value>ustlComAdHocAECGroupe</dc:Value>
      <dc:Value>ustlPrivGroupe</dc:Value>
      <dc:Value>groupOfNames</dc:Value>
      <dc:Value>ustlGroupe</dc:Value>
      <dc:Value>supannGroupe</dc:Value>
    </dc:Attribute>
</resolver:DataConnector>
```

Ici, les groupes ldap qui seront publiés dans le ldap à partir des groupes Grouper dont l'attribut "PubLDAPGroup" est égal à "AEC" auront tous les objectClass cités : ustlComAdHocAECGroupe,ustlPrivGroupe, groupOfNames, ustlGroupe, supannGroupe.

Un autre attribut mentionné est le cn du groupe ldap. Ici, il est calculé à partir de la valeur de l'attribut "cnLDAP1" déclaré dans ldappcg.xml et donc de l'attribut "cnLDAP" du groupe Grouper.

```
<resolver:AttributeDefinition id="cnLDAP1" xsi:type="ad:Simple" sourceAttributeID="cn">
    <resolver:Dependency ref="GroupDataConnectorAEC" />
</resolver:AttributeDefinition>
```

B- champ multivalué "ustlRole", branche ou=people du ldap

Etablir le lien entre la personne membre du groupe Grouper et la personne présente dans le ldap branche ou=people :

```
<resolver:AttributeDefinition id="member-dn" xsi:type="ad:Simple" sourceAttributeID="psoID" >
    <resolver:Dependency ref="SpmlDataConnector" />
</resolver:AttributeDefinition>

<resolver:DataConnector id="SpmlDataConnector" provider="ldap-provider"
    xsi:type="ldappc:SPMLDataConnector" scope="subTree" base="${peopleOU}" returnData="identifier">
    <resolver:Dependency ref="MemberDataConnector" />
    <ldappc:FilterTemplate>(uid=${id.get(0)})</ldappc:FilterTemplate>
</resolver:DataConnector>
```

Ici, c'est l'identifiant de la personne membre du groupe Grouper qui sera en relation avec l'attribut "uid" du ldap dans la branche ou=people.

Cela ne sera réalisé que pour les membres des groupes Grouper dont l'attribut "PubLDAPPeople" a la valeur "ustlRole".

```
<resolver:DataConnector id="MemberDataConnector" xsi:type="grouper:MemberDataConnector">
        <grouper:GroupFilter xsi:type="grouper:ExactAttribute" name="PubLDAPPeople" value="ustlRole"/>
```

```
  <grouper:Attribute id="groups" />
</resolver:DataConnector>
```

Ensuite, je définis comment obtenir la valeur à mettre dans l'attribut ldap (déclaré dans ldappcng.xml):

```
<resolver:AttributeDefinition id="memberIsMemberOf" xsi:type="grouper:Group" sourceAttributeID="groups">
    <resolver:Dependency ref="MemberDataConnector" />
    <grouper:Attribute id="ustlRole" />
</resolver:AttributeDefinition>
```

Ici, la valeur à mettre dans l'attribut ldap est celle qui se trouve dans l'attribut "ustlRole" du groupe Grouper.

Quelques scripts

Pour vous aider dans vos scripts, je vous ai joint plusieurs scripts et vous explicite ci-dessous leur fonction.

1. Publier dans la branche ou=people du ldap et garder une copie de ce qui se fait

pourPublierViaLdappcng.sh et wakscriptSync

Ce script permet d'interroger le ldap pour connaître les personnes qui sont à resynchroniser par rapport à ce qui existe dans Grouper. Ceux qui sont à modifier sont enregistrer dans le fichier uidATraiter et sont synchronisés l'un après l'autre par un sync. S'il y a eu des personnes synchronisées, le fichier uidATraiter est mémorisé dans un répertoire à part avec la date de publication. Cela permet de suivre ce qu'il se fait, c'est rassurant au début !

Il faut savoir que la synchronisation Grouper-Ldap version 1.6.3 n'est pas optimum car elle interroge systématiquement ldap pour chaque personne présente dans un groupe Grouper, ce qui est consommateur côté ldap. L'administrateur ldap ayant des craintes, il n'a pas souhaité que j'utilise un bulkSync mais un sync sur ceux à modifier. Finalement, en connaissant mieux cette synchronisation, je me rends compte que cela revient au même mais au moins, cela l'a rassuré et m'a permis de continuer Grouper. Il faut attendre les futures versions de ce connecteur pour avoir une version incrémentale : plutôt que d'interroger ldap, le connecteur interrogera Grouper pour savoir ce qui a été modifié depuis la dernière synchronisation (voir la version 2.1 de Grouper qui devrait sortir fin 2011).

2. Mettre tous les membres des groupes (publiant dans la branche ou=people du ldap) dans un groupe témoin

NettoyerPuisDonnerLeRoleTemoin.sh - pourNettoyerRoleTemoin.sh - nettoyerLeRoletemoin.gsh - pourDonnerLeRoleTemoin.sh - donnerLeRoleTemoin.gsh -

Il y a actuellement un problème avec le couple Grouper/ldappcng (sera corrigé avec l'arrivée du "real-time and incremental provisioning" prévu à la roadmap). Le problème est qu'il ne publie les modifications dans le ldap QUE pour les membres des groupes. Cela fait que si la personne n'est plus membre d'aucun groupe, il ne sera pas publié et l'on pourrait donc avoir un décalage entre ce qui est dans le ldap et ce qui est dans Grouper. Par exemple, user1 est membre de mongroupe1 et mongroupe2. Il est publié dans le ldap avec l'attribut ustlRole (pour Lille1) égal à 2 valeurs : mongroupe1-users et mongroupe2-users. Après sa publication dans le ldap, il est retiré de ces 2 groupes et ne figure plus dans aucun autre groupe : à ce moment-là, il n'est pas republié dans le ldap car le module ldappcng n'interroge et met à jour ldap que pour ceux qui sont dans des groupes Grouper...

Cela est dû au fonctionnement actuel : le module ldappcng interroge ldap, uid par uid présent dans un groupe, pour savoir si ce qu'il a côté grouper pour cet uid est conforme à ce qu'a ldap pour cet uid. Mais comme il n'interroge que les uid présents dans un groupe, il n'interroge plus ceux qui ne sont plus dans aucun groupe... il vaudrait mieux que ldappcng interroge Grouper des modifications qui ont eu lieues depuis la dernière publication ldap, je pense que c'est ce qui sera fait dans la nouvelle version de ce module ldappcng.

En attendant, j'ai prévu une "rustine" pour éviter que cela se produise. Je mets systématiquement tout membre d'un groupe publiable dans le ldap (mes fameux groupes de type ustlRole) dans un groupe qui sert de témoin de présence : groupe témoinustlRole.

Attention, j'ai dû créer 2 sources.xml différents : l'un pour ce groupershell et l'un pour LiteUI car mon administrateur ldap souhaitait que j'optimise mes requêtes ldap pour ces scripts. Donc, dans le script pourDonnerLeRoleTemoin.sh, j'utilise la fonction SubjectFinder.findAll("*") qui utilise la recherche sur l'attribut "ustlRole" paramétrée dans sources.xml(.gsh) :

```
<search>
       <searchType>search</searchType>
         <param>
            <param-name>filter</param-name>
            <param-value>(ustlRole=%TERM%)</param-value>
etc
</search>
```

Si je laissais le sources.xml comme cela, LiteUI ne fonctionnerait pas car il rechercherait sur l'attribut ustlRole plutôt que sur l'uid et le displayName comme paramétré dans sources.xml(.liteui) :

```
<search>
       <searchType>search</searchType>
         <param>
            <param-name>filter</param-name>
            <param-value>(&amp;amp; (|(uid=%TERM%)(displayName=*%TERM%*)))</param-value>
        </param>
etc
</search>
```

3. Initialisation des données : peupler les groupes depuis le ldap

chargerustlRoles.gsh - chargerAdmin.gsh

Ceux-ci permettent de peupler les groupes Grouper à partir des valeurs déjà présentes dans l'annuaire (reprise de l'existant dans ldap dans Grouper).

Le premier permet de mettre comme membres du groupe Grouper de type "PAGSRole" toutes les personnes du ldap dont l'attribut ustlRole a la même valeur que l'attribut "ustlRole" du groupe Grouper.

Le second permet de mettre le droit Grouper "UPDATE" à toutes les personnes du ldap dont l'attribut ustlRole est égal à admin-+"valeur de l'attribut "ustlRole" du groupe Grouper. Par exemple, ceux qui ont ustlRole=admin-IntranetEquipeDirection auront le droit UPDATE sur le groupe Grouper dont l'attribut "ustlRole=IntranetEquipeDirection-users". Ils rejoindront aussi le seul groupe d'administration d'application que nous avons créé dans Grouper : le groupe des administrateurs d'application dont l'attribut "ustlRole=admin-appli".

Brigitte WALLAERT-TAQUET