

Documentation générale - Archimède

Nous aborderons d'abord Torque (db.apache.org/torque/), la couche de persistance pour la base de données relationnelle. La version 3.0.2 est utilisée pour le projet. Lors de la conception d'Archimède, Torque a été testé contre Cayenne (objectstyle.org/cayenne/) et DBFramework (drdb.fsa.ulaval.ca/sujets/dbfw/).

Torque a été choisi pour son API simple, son langage de requête efficace et pour le fait qu'il s'agit d'un projet Apache. Torque, comme les deux autres frameworks testés, génère les classes pour la couche de persistance à partir d'un fichier de « mapping ». Dans le cas de Torque, il s'agit d'un fichier XML devant être édité manuellement. Heureusement, celui se rapproche fortement du SQL et est donc facile à éditer. Il s'agit du fichier **/schema/project-schema.xml**. Chaque table y est identifiée par l'élément **<table>** et ses colonnes par l'élément **<column>**. Les colonnes uniques sont identifiées dans l'élément **<unique>**. Finalement les clés étrangères sont situées dans l'élément **<foreign-key>**.

Le fichier **build-torque.properties** spécifie les paramètres pour la génération des classes (dans **/src/java**) et des fichiers SQL (dans **/src/sql**). Celui-ci est utilisé par Ant automatiquement lors de l'exécution du fichier **build-torque.xml** (fichiers homonymes). On peut effectuer la génération des classes et du SQL en tapant **"ant -f build-torque.xml"** dans le dossier racine d'Archimède. Si on ajoute **"create-db"** ou **"insert-sql"** on créera respectivement la base de données et les tables. Les classes sont créées dans le dossier **"ca.ulaval.bibl.archimede.model.torque"**, tel que spécifié dans le fichier de propriété **build-torque.properties**.

Le contenu du « package » **"map"** est généré par Torque, ainsi que les fichiers commençant par **"Base"**. On ne modifie donc jamais ces fichiers puisqu'ils sont continuellement écrasés (mais pas automatiquement effacés) lors de modifications au schéma XML. Les autres fichiers de classe sont tout simplement vides au départ. Ceux-ci sont créés qu'une seule fois. Ils héritent des fichiers **"Base"**. On peut y ajouter nos méthodes personnalisées. Aussi, chacun des « beans » créés ont un **Peer**, soit une classes qui a le nom du bean + **"Peer"**. Les Peers sont en quelque sorte l'équivalent de DAOs. Ils fournissent les fonctions de CRUD (create, read, update, delete) et plusieurs fonctions complémentaires. Ils permettent notamment d'effectuer les requêtes pour la base de données. Passablement de méthodes ont été ajoutés aux classes héritant des **"Base"** durant le développement d'Archimède afin de regrouper la logique d'affaires à cet endroit.

Torque dispose d'une API simple et relativement puissante pour faire des requêtes :

```
Criteria criteria = new Criteria();
criteria.add(UsersPeer.USERNAME, username);
List li = UsersPeer.doSelect(criteria);
```

Pour avoir les groupes (par une jointure) d'un utilisateur :

```
Criteria criteria = new Criteria(); criteria.add(UsersPeer.OI, this.getOid());
criteria.addJoin(UsersPeer.OI, GroupRoleUserPeer.OIUSERS);
criteria.addJoin(GroupRoleUserPeer.OIGROUPS, GroupsPeer.OI);
return GroupsPeer.doSelect(criteria);
```

Le Peer sert à exécuter la requête. Il spécifie également le type des objets de la liste retournée. Notons que pour les *delete*, on peut effacer des tables qui ne sont pas celles du Peer (voir `UploadsPeer.doCascadeDelete(int oi, ServletContext application)`).

Soulignons que trois problèmes sont liés Torque :

1. D'abord, celui-ci n'est caché par aucune couche d'interface dans Archimède. De sorte que les classes de Torque se retrouvent partout dans l'application et qu'il serait difficile de se débarrasser de lui.
2. Le développement de Torque a stagné depuis 1 an. Avec le standard JDO qui est apparu après la conception d'Archimède et la montée de l'outil Hibernate, l'avenir est peu certain pour l'outil. Par contre, Torque supporte plusieurs bases de données et s'est montré très solide jusqu'à maintenant.
3. Une autre difficulté avec Torque est qu'il ne supporte pas adéquatement les cascades. De sorte que celles-ci ont été implémentées manuellement dans Archimède.

La version 1.1 du framework de développement d'applications Web Struts (struts.apache.org/) a été utilisée. Les classes du contrôleur sont situées dans le « package » "ca.ulaval.bibl.archimede.controller". Une seule action (méthode `execute()`) est contenu par classe. Cette approche a le défaut de créer beaucoup de classes dans le projet. Vous pourrez tout de même vous retrouver facilement à partir du fichier `\war\WEB-INF\config\struts-config.xml` puisque les noms d'action correspondent à ceux des classes.

Le module Struts Validator a été utilisé pour la validation des « beans » de formulaires sauf pour les cas spéciaux. Ceux-ci sont abordés plus loin. L'internationalisation de Struts a été utilisée pour sa simplicité. Les fichiers de traduction sont contenus dans le package "resources" soit les fichiers `application*.properties`. Notez que Struts détecte automatiquement si l'on ajoute un nouveau fichier de langue. Il faudra cependant modifier la classe

ca.ulaval.bibl.archimede.utils.TokensServlet pour aviser les pages JSPs qu'une langue a été ajoutée. De plus il faudra modifier le fichier par défaut de langues et ajouter la clé pour la langue ajoutée.

Les tags de Struts ont été utilisés uniquement. Les quelques tags JSTL qui été présents dans l'application ont été retirés à cause d'un problème avec la dernière version de Tomcat 5. L'utilisation de scriptlets a été minimisée. Cependant, celle-ci s'est avérée notamment utile pour cacher certaines sections de pages selon les rôles des utilisateurs. Pour les Tiles, un cadre général a été créé pour l'application. Celui est situé dans le dossier /war/layouts.

La classe ca.ulaval.bibl.archimede.utils.MailerServlet sert à envoyer les messages aux personnes qui sont abonnées à des collections ou à des groupes. Elle est également utilisée pour les rappels de mots de passe. Elle tire ses paramètres du fichier /ressources/Mailer.properties. Velocity a été utilisé pour le contenu des courriels. Les templates de Velocity sont situés dans le dossier /war/WEB-INF/config sous *.vm.

La structure de répertoire est décrite dans la section « Guide de personnalisation ».

Le formulaire pour le téléversement des dépôts est particulier. Il gère deux sections délicates: soit le téléversement de fichiers et les champs dynamiques pour le DublinCore. D'abord la variable **MAXSIZE = 8000*1024** indique le nombre d'octets maximum traité par le formulaire. Le champ **FormFile formFile** contient le fichier soumis dans le formulaire. Si celui-ci ne fait pas dépasser la limite fixée, il est ajouté au **Map files**. Les noms de fichiers constituent les clés, afin d'éviter de soumettre deux fichiers identiques (et d'éviter des conflits de noms pour Archimède). Les fichiers sont transférés dans le dossier de dépôt à partir du contenu du Map.

Pour le Dunblin Core, le **Map dcElements** gère les 15 champs. Les clés sont les noms des éléments du DublinCore et le contenu est une List. On remarque dans le fichier /war/pages/restricted/user/ UploadMetadata.jsp :

```
<textarea name="<%= "value(" +dcElementName+"_0) "%>" cols="30" rows="2">
</textarea>
```

Pour Struts, *value()* indique que le champs est représenté par un Map. Le formulaire peut donc gérer de multiples occurrences sans autant de méthodes set/get. On a qu'à filtrer les paramètres de la méthode setValue() pour les placer dans une List d'occurrence du champs qui lui, est placé dans un Map.

Quelques notes sur le framework de sécurité développé par Vincent Dussault (drdb.fsa.ulaval.ca/sujets/security/). Celui-ci est configuré par le fichier `/war/WEB-INF/config/application-users-jdbc-config.xml`. Il s'agit en fait d'un filtre J2EE qui est spécifié dans le fichier `/war/WEB-INF/web.xml`. Les contraintes sont spécifiées dans le fichier `/war/WEB-INF/config/security-constraints.xml`. Elles sont basés sur des noms de dossiers C'est donc pourquoi les actions comportent des noms de dossiers devant elles et que les fichiers JSPs sont stockés dans des répertoires homonymes. Les tags du framework de sécurité ont été principalement utilisés dans la page `/war/pages/BasicHeader.jsp` pour sécuriser les menus. Le reste des contraintes de sécurité de l'application a été implanté de façon programmatique. C'est à dire que du code dans les actions s'assure que seuls les utilisateurs autorisés peuvent y accéder. Le désavantage de cette méthode c'est que les règles de sécurité et la structure des rôles est très complexe à changer. Par contre, cette approche simplifie la vérification de relations complexes entre les rôles. Par exemple : Luc est l'administrateur du groupe x et peut y effacer des dépôts.

Les classes d'action Exploration utilisent la classe Exploration qui permet de trier des listes pour des noms de colonnes donnés. Cette fonctionnalité permet d'implanter facilement les tris pour les pages d'exploration. Voici l'extrait JSP effectuant le regroupement par pages d'une page d'exploration :

```
<!-- Begin of List Pages Index -->
<%
String startIndex = request.getParameter("startIndex");
int indexOff = 0;
if (startIndex != null && !startIndex.equals(""))
    indexOff = Integer.parseInt(startIndex);

ca.ulaval.bibl.archimede.utils.ListOffset offset =
    new ca.ulaval.bibl.archimede.utils.ListOffset(groups.size(), 10,
indexOff, 1);

pageContext.setAttribute("offset", offset);
pageContext.setAttribute("backIndexOff",
    new Integer(indexOff-offset.getElementsPerPage()));
pageContext.setAttribute("nextIndexOff",
    new Integer(indexOff+offset.getElementsPerPage()));
%>

<logic:equal name="offset"
    property="isBackwardNeeded"
    value="true"
    scope="page">
    [<html:link forward="exploreAllGroups"
        paramName="backIndexOff"
```

```

        paramId="startIndex"
        paramScope="page">
    <bean:message key="exploreAllGroups.backPage" /></html:link>]
</logic:equal>

<logic:equal name="offset"
    property="isPageMissingBackward"
    value="true" scope="page">
    ...
</logic:equal>

<% for (int j=offset.getFirstVisiblePage(); j<=offset.getLastVisiblePage();
j++){ %>

    <logic:equal name="offset"
        property="currentPage"
        value="<%=new Integer(j).toString()%>">
        <b><%=j+1%></b>
    </logic:equal>

    <logic:notEqual name="offset"
        property="currentPage"
        value="<%=new Integer(j).toString()%>">
    <bean:define id="index"
        value="<%=new Integer(j*offset.getElementsPerPage())
        .toString()%>" />
    <html:link forward="exploreAllGroups"
        paramName="index"
        paramId="startIndex"
        paramScope="page">
        <%=j+1%>
    </html:link>
</logic:notEqual>

<% } %>

<logic:equal name="offset"
    property="isPageMissingForward"
    value="true"
    scope="page">
    ...
</logic:equal>

<logic:equal name="offset"
    property="isForwardNeeded"
    value="true"
    scope="page">
    [<html:link forward="exploreAllGroups"
        paramName="nextIndexOff"

```

```

        paramId="startIndex"
        paramScope="page">
        <bean:message key="exploreAllGroups.nextPage"/>
    </html:link>]
</logic:equal>

<!-- End of List Pages Index -->

```

Voici maintenant la section qui itère sur la liste :

```

<logic:iterate id="currentGroup"
    name="allGroups"
    scope="request"
    offset="<%=new Integer(indexOff).toString()%>"
    length="<%=new Integer(offset.getElementsPerPage())
        .toString()%>"
    type="ca.ulaval.bibl.archimede.model.torque.Groups">

```

Bien que beaucoup de code compose la division par pages. Celui-ci est tout de même assez simple. La première section (scriplets) construit un objet ListOffset (public ListOffset(int numberOfElements, int numberOfElementsPerPage, int offset, int numberOfPagesToShow)), selon les paramètres dans la requête. Ensuite, on place les variables nécessaires dans la page (pageContext). Dans les sections suivantes, on vérifie si on peut revenir à une page précédente, si des pages ne sont pas affichées, on affiche ensuite les numéros de page disponibles (la page courante n'est pas cliquable), on termine avec l'inverse (pages suivantes non affichées + page suivante disponible). Les numéros de pages disponibles sont cliquables et passent en paramètres les nouvelles valeurs pour l'affichage de la liste d'occurrences.