JMMC-MEM-2600-0001

Revision : 1.0

Date : 07/05/2004

JMMC

EVALUATION DES OUTILS CDS : PARSER XML ET INTERFACE SOAP

Yannick Vanderschueren (yannick.vanderschueren@obs-azur.fr)
OCA(Observatoire de la Cote d'Azur

CHANGE RECORD

REVISION	DATE	AUTHOR	SECTIONS/PAGES AFFECTED
	REMARKS		
1.0	07/05/2004	Y. Vanderschueren	all
	Première version		

TABLE OF CONTENTS

1 Exposé du service web (utilisation)

1.1 Definitions

SOAP protocole de transmission de messages. Il définit un ensemble de règles pour structurer des messages qui peuvent être utilisés dans de simples transmissions unidirectionnelles, mais il est particulièrement utile pour exécuter des dialogues requête-réponse du type RPC (Remote Procedure Call).

Service Web Un Web Service est une application logicielle identifiée par un URI(Uniform Resource Identifier), dont les interfaces et associations peuvent être définies, décrites et découvertes par des méthodes XML. Il peut également interagir directement avec d'autres applications en utilisant des messages XML via les protocoles Internet standards (HTTP...).

1.2 Principe

```
Client--->SoapRequest--->Serveur--->SoapResponse--->Client
```

Exemple :

- le Client peut être une application Java,
- qui utilise un package Soap
- qui se charge de la sérialisation des données en XML, forme l'enveloppe Soap, et l'encapsule dans une requete Post Http,
  - transport par TCP/IP,
  - un serveur "ouvert à ce type de requête" reçoit la requête,
  - puis donne l'enveloppe à son module Soap qui parse le XML
  - et appelle la méthode java de "service" qui, en tant que composant métier, peut appeler une base de données,
  - la méthode renvoie une valeur résultat,
  - encodée XML par soap,

1.3 Utilisation

Packages :

- Package axis-1_1
  Il contient de nombreux sous-packages permettant l'utilisation du protocole SOAP et de service web:
  - axis.jar
  - jaxrpc.jar
  - saaj.jar
  - commons-logging.jar
  - commons-discovery.jar
  - wsdl4j.jar

  Téléchargeable sur le site : http://ws.apache.org/axis/

- Package VizieR
  Ce package permet d'utiliser les différentes méthodes du service web du cds. Il est généré suite a la commande :

```
% java org.apache.axis.wsdl.WSDL2Java http://cdsws.u-strasbg.fr/axis/services/VizieR?wsdl
```

  Cette commande créé un répertoire "VizieR_pkg" contenant le code java des classes et méthodes du service bébé du d'il faut ensuite les compiler afin d'obtenir les fichiers classes (java *.java).

  Ces packages doivent être spécifier et donc doivent être ajoutés a la variable CLASSPATH.

  Il existe deux méthodes pour utiliser le protocole SOAP : soit par une requête XML, soit en utilisant un client.
  L'utilisation d'un client java permet une lecture plus simple du code et permet également de ne pas utiliser le langage XML-SOAP qui est un peu plus complexe à maintenir.
  Cela permet également de ne pas gérer l'envoi de requête qui se fait automatiquement grâce aux différentes méthodes que l'on trouve dans les packages (Call...).
  Cette solution a donc été choisie.

1.4 Algorithme

```
Service = VizieRService
Objet = Service.getVizieR
fichierVOTable = Objet.catalogueData(...coordonnees...)
Affichage (fichierVOTable)
```

1.5 Problèmes rencontres

L'utilisation de la méthode catalogueData ne permet pas de spécifier les champs désirées ou les contraintes soumis a la recherche.
Cette même méthode ne semble pas permettre d'utiliser plusieurs catalogues dans une même recherche.
Suite a ces différentes interrogations, un mail d'aide a été envoyé au CDN afin de résoudre ces problèmes.(attente de réponse).

1.6 Lancement

Le script "cli.sh" s'effectue, pour l'instant, que le changement de la variable CLASSPATH.
La commande pour exécuter le script est la suivante :

```
% . cli.sh
```

Suggestion :
Si l'utilisation du service web est confirme, le script cli devrait exécuter le client java en recevant les paramètres de la requête a envoyer au cds.

1.7 Résultats obtenus

Après avoir parser le fichier VOTable récupérer par le client java, le fichier ASCII obtenu est le même que celui obtenu via Internet (emplacement des champs est différent mais le nombre d'étoiles et leurs valeurs sont identiques).

1.8 Code source

Voir Appendix A.

2 Exposé du parser XML

2.1 Principe

La réception du fichier VOTable (XML) nécessite l'utilisation d'un parser(analyseur) XML afin de récupérer les différentes données contenues dans ce fichier.
Ce fichier XML est un document contenant toutes les caractéristiques de l'étoile calibrée.
L'utilisation d'un analyseur de fichier XML est donc indispensable.
Ce parser devait être implementé en java afin d'utiliser les packages développés par l'équipe informatique du cds (centre de données de strasbourg).
L'algorithme du parser est assez simple.En effet, il suffit de parcourir le fichier VOTable grâce à des balises spécifiques et de récupérer les données encapsulées par des balises standards.

2.2 Utilisation

Packages :

- cds.savot.model.jar
- cds.savot.pull.jar
- kxml2-min.zip

Version : 2.0

Ces packages décrivent toutes les méthodes nécessaires pour parser un fichier VOTable.
Par exemple, la méthode getTDs de la classe SavotTR permet de récupérer une instance de la classe TDSet qui sera utilisée, par la suite, pour retourner le contenu d'un TD (balise xml).

Ces packages sont disponibles sur le site :

`http://cdsweb.u-strasbg.fr/devcorner.gml` (download area)

Ils doivent être ajoutés à la variable CLASSPATH.

kxmleditor : (logiciel libre, distribution linux)
KXMLEditor est un logiciel permettant d'afficher, créer et modifier des documents au format XML et cela de manière visuelle, avec un navigateur (en arbre) de la structure du document.
Il a été utilisé, dans notre cas, pour étudier la structure des fichiers VOTables permettant ainsi de trouver un algorithme complet.
Installer par l'ingénieur système David Chapeau.

2.3 Algorithme

```
\n-->retour chariot
\t--->tabulation
Entrée--->fichier VOTable
Sortie--->fichier résultat

DEBUT
TantQue (BaliseRessource != NULL)
 |   Pour i de 0 a NbreChamps-1
 |    |   si i == NbreChamps-1 Alors Affichage(Champs \n)
 |    |   sinon Affichage(Champs \t)
 |   FinPour
 |
 |   Pour j de 0 a NbreChamps-1
 |    |   Selon TypeChamps
 |    |    |   Affichage(TailleChamps "de fois" "-" \t)
 |    |   FinSelon
 |    |   Affichage (\n)
 |   FinPour
 |
 |   Pour k de 0 a NbreTR-1
 |    |   Pour m de 0 a NbreTD-1
 |    |    |   si      m      ==      NbreTD-1      Alors
Affichage(ContenuTD \n)
 |    |    |   sinon Affichage(ContenuTD \t)
 |    |   FinPour
 |   FinPour
FinTantQue
FIN
```

Une redirection dans un fichier texte et on obtient le résultat attendu. Le résultat est ensuite traité par le logiciel JMMC SearchCal codé en C.
Il est donc nécessaire d'implémenter une méthode permettant l'appel à du code java depuis du code C.

Deux propositions se présentent à nous :

- La première, la plus complexe, consiste à utiliser une JNI (Java Native Interface).
  Cela permet de coupler du code java et du code C, C++.Cette solution est très efficace mais elle ne permet pas un maintient du code assez facile.
- La deuxième, plus simple (un peu lourde), consiste à utiliser un appel système en C. Il suffit, pour cela, d'utiliser la fonction C : system (cmd).D'autres fonctions permettant le même appel système peuvent être utilisées (execv, execvp, execl, execlp).

La deuxième proposition est la proposition la mieux adaptée à notre situation.En effet, il est possible d'avoir du code java ou C ou même les packages à modifier afin d'optimiser au maximum le parser.Il est donc indispensable d'utiliser des fonctions simples puisque le code peut être modifier a tout moment et par tout type de personne.

2.4 Lancement

Le script "parser.sh" effectue le changement de la variable CLASSSPATH, exécute le Makefile afin de compiler le java et exécute le code C qui fait appel au Parser java.
Il faut taper la ligne de commande suivante :

```
% . parser.sh
```

Le changement de la variable CLASSPATH s'effectue par la commande :

```
% export CLASSPATH=chemin
```

L'execution du Makefile se fait grâce a la commande :

```
% make
```

L'appel a un exécutable C :

```
% ./lancerparser
```

Cet exécutable contient le code permettant l'appel système. La commande suivante est exécutée : system ("java Parser NOM_FICHIER_XML >RESULTAT.txt")

Pour utiliser ce script dans un programme C, il suffit de faire un appel système avec lancement d'un nouveau processus (forker).L'utilisation d'un nouveau processus est nécessaire car la fonction "system" ne permet pas l'exécution des lignes de code qui la succède.

2.5 Résultats obtenus

Le fichier txt que l'on récupère est identique à celui du logiciel JMMC SearchCal généré.

2.6 Code source

Voir Appendix B.

3 Conclusion

L'utilisation du service web du cds ne nous apporte pas encore les mêmes résultats que ceux obtenus avec les fonctionnalités existantes. Il est préférable de continuer a vouloir integrer ce service a l'application JMMC SearchCal car ce service utilise un protocole (SOAP) qui est un protocole robuste et il permet un maintient du code moins contraignant (grand avantage).
Le parser java permet de lire le fichier XML de façon plus structuré. Il est donc plus simple a maintenir. Cette application java est mieux adaptée, lorsque l'on utilise le protocole SOAP, que les fonctionnalités C existantes.

Appendix A. Code source Client java (SOAP)

Constructeur de la classe Client

```
public Client() {

      try {

          // locator creation
          VizieRService locator = new VizieRServiceLocator();

          // VizieR object
          VizieR myv = locator.getVizieR();

          // recuperation donnees du catalogue - VOTable
          String   star   =   myv.cataloguesData("eta   tau",10.0,   "arcmin",
"II/7A");
          System.out.println(star);

          /***AUTRES FONCTIONS DU***/
          /*******SERVICE WEB********/
          // recuperation metadonnees du catalogue - VOTable
          //      String      metastar      =      myv.cataloguesMetaData("eta
tau",10.0,"arcmin","II/7A");
          // System.out.println(metastar);

          // toute metadonnees - VOTable
          // String metadata = myv.metaAll();
          // System.out.println(metadata);

      }
      catch (Exception e ) {System.out.println("VizieR XML WS client : " +
e);}
  }
```

Appendix B. Code source Parser java

Constructeur de la classe Parser

```
public Parser(String source, String target, int type) {

        TRSet tr = null;
        FieldSet field = null;
        SavotField SF = null;
        try {

            // debut du parsing avec creation objet parsing
            SavotPullParser sb = new SavotPullParser(source, type);

            //si sb n est pas null alors le document existe
            // on peut continuer le parsing
            if (sb != null) {
               // recupere la premiere ressource pour ensuite faire la boucle
               SavotResource currentResource = sb.getNextResource();
               while (currentResource != null) {

               //determine si il y a une table dans la ressource
               if (currentResource.getTableCount() != 0) {
                     // recupere le nombre de tr de la  table de la ressource
et on boucle
                     for (int i = 0; i < currentResource.getTableCount(); i++)
{

                         // recupere ressource du tr de i
                         tr = currentResource.getTRSet(i);

                         // recupere champs afin d'avoir les parametres
                         field = currentResource.getFieldSet(i);
                         int count = field.getItemCount();

                         // affichage des parametres
                         for (int j=0;j<count;j++) {
                            SF = (SavotField)(field.getItemAt(j));
                            if (j==count-1)System.out.println(SF.getName());
                            else System.out.print(SF.getName()+"\t");
                         }

                         // affichage des -
                         for (int m=0;m<count;m++) {
                            SF = (SavotField)(field.getItemAt(m));
                            if
(SF.getDataType().equals("float")||SF.getDataType().equals("short")||SF.getDataType
().equals("int")||SF.getDataType().equals("double")){
                                String s = SF.getWidth();
                                int taille =Integer.parseInt(s);

                                for  (int  p=0;p<taille;p++)System.out.print("-
");
                            }
                            else{
                                if (SF.getDataType().equals("char")){
                                   //String tab = SF.getArraySize();
                                   //tab                                    =
tab.toString();System.out.print(tab);
                                   //Integer n = Integer.valueOf(""+tab+"");
                                   //System.out.println(n);
                                   //int taille2 = n.intValue();
                                   //System.out.println(taille2);
                                   System.out.print("----------");
```

```
                                        //for                                   (int
q=0;q<taille2;q++)System.out.print("-");

                                    }
                                }



                                if (m==count-1)System.out.println();
                                else System.out.print("\t");
                            }

                            if (tr != null) {

                                // recupere le nombre de td pour chaque tr et on
boucle
                                for (int j = 0; j < tr.getItemCount(); j++) {

                                    // recupere les donnees de la ligne
                                    TDSet theTDs = tr.getTDSet(j);

                                    String currentLine = new String();


                                    // pour chaque ligne
                                    for (int k = 0; k < theTDs.getItemCount(); k++)
{
                                        currentLine       =       currentLine       +
theTDs.getContent(k);
                                        if (k==theTDs.getItemCount()-1)

      System.out.println(theTDs.getContent(k));
                                        else    System.out.print(theTDs.getContent(k)
+ "\t");
                                    }
                                }
                            }
                        } //table vide
                        else     System.out.println("la  table  est  vide  ->  aucune
information recue");
                        // obtenir la ressource suivante
                        currentResource = sb.getNextResource();
                    }
                    /******************AUTRES FONCTIONS DU PACKAGE**************/
                    //     System.out.println("resource   counter          :    "   +
sb.getResourceCount());
                    //           System.out.println("table  counter           :  "  +
sb.getTableCount());
                    //     System.out.println("table    per    resource    :    "    +
(float)sb.getTableCount() / (float)sb.getResourceCount());
                    //     System.out.println("row  counter                  :   "  +
sb.getTRCount());
                    //     System.out.println("row  per  table               :   "  +
(float)sb.getTRCount() / (float)sb.getTableCount());
                    //     System.out.println("data  counter                 :   "  +
sb.getDataCount());
                    //     System.out.println("data  per  raw                :   "  +
(float)sb.getDataCount() / (float)sb.getTRCount());
                    //     System.out.println("data  per  table              :   "  +
(float)sb.getDataCount() / (float)sb.getTableCount());
                }
                //probleme lie a la connexion ->erreur socket
                else     System.out.println("Erreur   liee   a   la   connexion   ->
probleme de socket");
            }
            catch(Exception e) {System.err.println("Parser : " + e);};
        }
```