Projet Dojo « jSearch »

M2 MIAGE NTDP 2009-2010
Kahina BERKANI – Ludovic DEVILLERS

L'application a été baptiser jSearch. Il s'agit d'un moteur de recherche multi-service et multi-média (photo, vidéo, carte, texte, etc) . Entrez le ou les termes que vous voulez rechercher dans le champ prévu à cet effet et cliquer sur rechercher. L'application ira rechercher l'information via différent service comme Youtube, Dailymotion, Flickr, Picasa, Google Maps.

jSearch a été développer de façon a pouvoir rajouter des nouveaux services de recherche quel qu'il soit. Ces nouveaux services peuvent être déployés par le développement de plugins. Une fois ajouté à l'application, ils seront automatiquement utilisés. On peut imaginer utiliser des services autres comme PageJaunes, Twitter, FaceBook, Bing, Yahoo, etc ..

JSearch en bref

- Refonte total du code par rapport à ce qui avait été présenté à dernière fois.
- Glassfish 2.1
- Recherche multi-service
- Interface graphique Dojo
- Extensible via plugins
- Introspection – Réflexivité
- JSON

Des screenshots et une vidéo montre le fonctionnement de jSearch.
Toutes les librairies nécessaires au fonctionnement de l'application se trouvent dans le dossier « lib » du projet Netbeans. Il se peut qu'il faille les rechargées dans le projet Netbeans.

Fonctionnement

Le schéma suivant résume le fonctionnement de l'application.

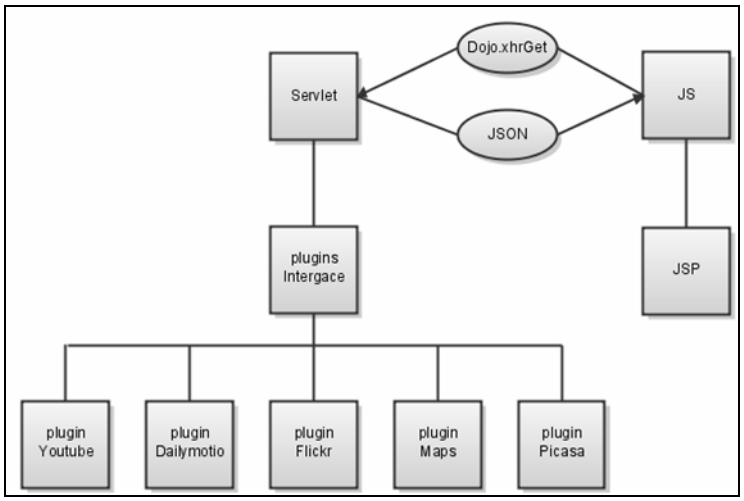


Figure 1 - Schéma de fonctionnement

Lors d'une recherche via jSearch, la JSP demande à la JS d'effectuer la recherche. La JS envoi une requête à la Servlet. La Servlet utilise l'introspection pour aller chercher les plugins à charge pour l'application. Puis grâce à la réflexivité, les instances de plugins sont créées à la volée. Chaque plugins renvoi les résultats en fonction de la recherche demandé. Le tout est stocké dans un objet qui est transmis au format JSON à la JS. La JS s'occupe de la mise en forme des données dans structure DOM de notre page web.

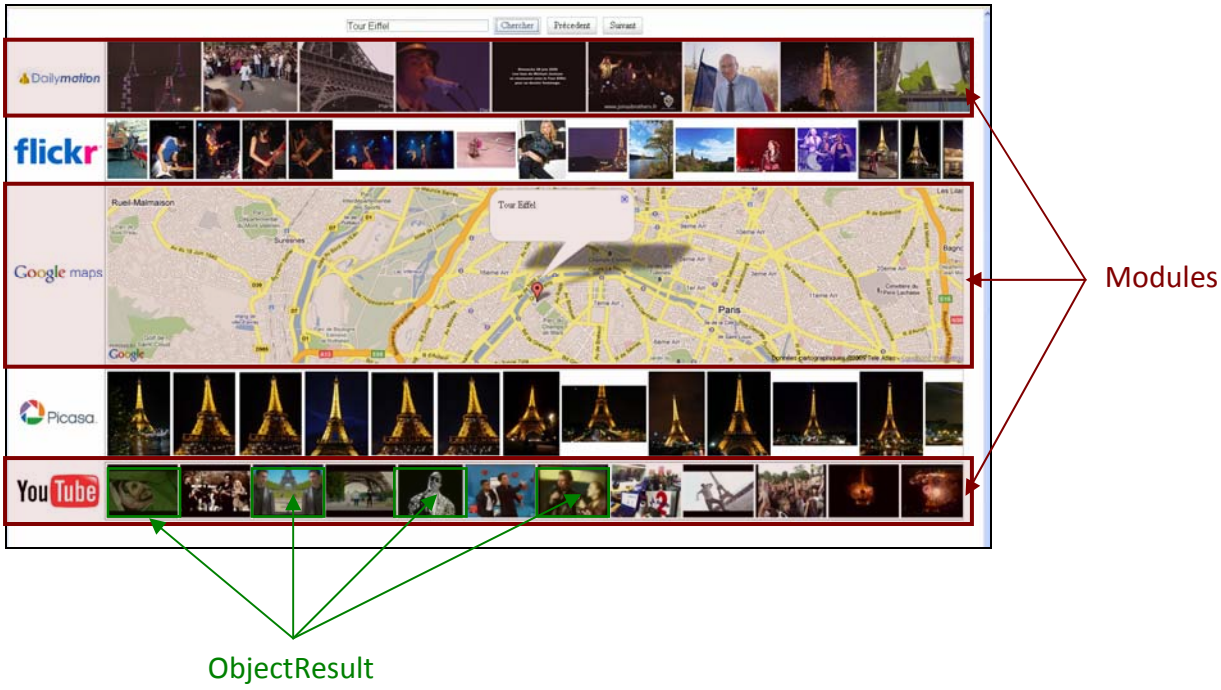
Grâce à la création d'un plugin qui implémente l'interface « pluginInterface », il est possible de faire appel à tout autre service web et de l'intégrer à l'application. Ceci sont modifier le code de l'application coté serveur et coté client.

```
public interface pluginInterface {

    public boolean loadPlugin();

    public String getType();

    public ArrayList<ObjectResult> search(String query, int n, int m);

    public String getLogoURL();

    public int getCountResult();

    public String getPreJS();

    public String getPostJS();

    public HashMap<String,String> getFileToLoad();
}
```

Figure 2 - détail de la classe plugInterface

- loadPlugin : retourne si vrai si le plugin doit être chargé.
- getType : retourne le type de média utilisé (Video, Photo, Texte, Maps, etc.)
- search : retourne les données concernant la recherche « query »
- getLogoURL : retourne l'url du logo du service
- getCountResult : retourne le nombre total de résultat de la recherche
- getPreJS : retourne sous forme de String le code JS à exécuté coté client avant le « peuplement » la page avant avec les données retournée par la Servlet.
- getPostJS : retourne sous forme de String le code JS à exécuté coté client après le « peuplement » la page avant avec les données retournée par la Servlet.
- getFileToLoad : Retourne la liste des url des fichiers .js ou .css à charger coté client pour l'utilisation du plugin.

Chaque résultat fourni par un service correspond à un « module » lui-même composé d'un ou plusieurs « ObjectResult ».



Faiblesse du projet

- Le code n'est pas commenté
- Un petit loading serait bienvenue lorsque l'on clique sur le bouton « chercher »
- Il faudrait réorganiser l'affichage des informations en utilisant un dojo.Tabcontainer pour organiser les résultats par type de média.
- Les ressources externes comme les fichiers .js nécessaire au fonctionnement des plugins ne sont pas encore chargé à la volée. (Bug dans le code)