Projet de programmation Internet

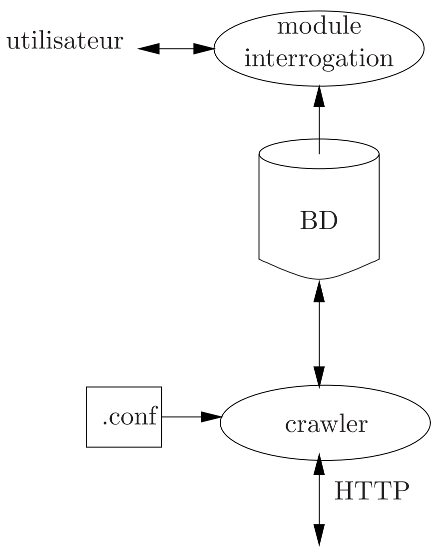
IUP3 année 2003-2004

Date de remise du projet : vendredi 12 décembre 2003 à 16h

1 Description du projet

Le but de ce projet est d'implémenter un robot de recherche WEB de type GOOGLE[1] simplifié. Pour cela, vous utiliserez les formulaires HTML, les CGIs et la programmation en Java.

L'architecture générale du robot est spécifiée dans la figure ci-dessous. Le robot comporte trois parties : (1) le *crawler* (araignée) qui explore le pages HTML du WEB afin de constituer (2) la base de données liant les URI aux mots qui les décrivent et (3) le module d'interrogation qui cherche dans la base des mots et affiche les URIs correspondants.



1.1 La base de données

Elle sert de tampon de communication entre le *crawler* et le module d'interrogation. Le *crawler* enrichit cette BD avec les informations recueillies sur les URIs explorées. Le module d'interrogation recherche dans cette base de données les mots données par l'utilisateur. Ces opérations peuvent être exécutées en parallèle. Il résulte que deux des propriétés souhaitées de cette BD sont : (1) efficacité de mise à jour, (2) efficacité de la recherche et (3) stockage efficace. Vous devez donc

[1] `www.google.fr`

implémenter dans votre projet une BD qui permet de stocker les URI et les mots qui apparaissent comme contenu de la balise HTML `a` de l'URI.

Pour cela, vous pouvez utiliser un système de gestion de BD comme Oracle ou MySQL.

Dans la documentation pour cette partie du projet, vous devez :

- Expliquer la solution choisie pour l'implémentation de la BD.
- Evaluer la complexité des opérations de mise à jour et de recherche de votre solution.

1.2 Le *crawler*

Il est le coeur du robot. Il doit explorer le WEB et recueillir les informations correspondant à chaque URI.

Plusieurs solutions sont envisageables pour l'implémentation du *crawler*. Dans ce projet, on s'intéresse à celle qui explore le WEB en suivant les liens hypertexte de chaque page, récursivement. Pour chaque lien rencontrée, le *crawler* doit stocker les mots qui constituent le *contenu* (c'est-à-dire, le texte entre l'étiquette de fin et de début) de ce lien. Par exemple, pour le lien hypertexte suivant :

```
Un exemple de robot WEB est <A HREF="http://www.google.com">Google</A>.
```

le *crawler* doit associer à l'URI `http://www.google.com` le mot `Google`.

Plus précisément, votre *crawler* doit, en partant d'une URI racine R, explorer les liens de cette URI récursivement sur une profondeur donnée p.

Les paramètres R et p doivent être fixées dans un fichier `crawler.conf` au format `Property` qui est donnée comme paramètre du *crawler*. De plus, le fichier `crawler.conf` peut être crée/modifié par l'utilisateur à partir d'un formulaire HTML (et son CGI).

Pour l'analyse lexicale et syntaxique des pages HTML, vous pouvez utiliser lÁPI `javax.swing.text.html.*` dont un exemple d'utilisation est donné dans l'annexe A.

L'API `java.net.*` fournit des méthodes pour accéder à la ressource dénotée par un URI, pour résoudre un URI relatif, etc.

Le crawler doit accepter plusieurs options :

`--init` faire une exploration des URI de manière inconditionnelle à partir de la racine et sur la profondeur données dans le fichier de configuration. Cette option permet de constituer la BD initiale, donc la base sera vidée avant remplissage.

`--img` introduire dans la base les URI correspondant à des images. Les mots correspondant doivent être ceux de l'attribut `ALT` ou le nom du fichier image.

`--mail` introduire dans la base les URI correspondant à des adresses e-mail.

`--cond` faire une exploration des URI de la base de manière conditionnelle pour mettre à jour la BD déjà remplie (à partir de la racine et sur la profondeur données dans le fichier de configuration). Uniquement les ressources de l'arborescence initiale qui ont été modifiées depuis une date donnée en paramètre seront parcourues.

Pour cette option, vous pouvez chosir entre une mise à jour exacte de l'arborescence initiale (difficile, pourquoi ?) ou une mise à jour qui correspond à une sur-approximation de l'arborescence initiale par rapport à la date donnée.

Le *crawler* doit faire un rapport des opérations effectuées dans un fichier de journalisation `crawler.log`. Ce fichier doit contenir une ligne par URI explorée et le résultat de cette exploration (p.e., `http://somewhere/path OK 16 liens`).

1.3 Le module d'interrogation

Ce module contient :

- Une interface utilisateur (formulaire HTML et CGI) qui permet de saisir le(s) mot(s) à chercher dans la BD.
- Une procédure d'interrogation de la BD.
- Une procédure d'affichage des résultats de l'interrogation de telle manière qu'on peut suivre les liens trouvés (voir Google).

Une extension appréciée de ce module consiste à accepter des expressions régulières et booléennes pour la recherche des mots.

2 Documentation du projet

La documentation fournie avec le projet **DOIT** respecter les contraintes suivantes:

- Elle doit être écrite en HTML.
- Elle doit expliquer la mise en oeuvre et les fonctionnalités de chaque partie de votre projet.
- Elle doit présenter de façon abstraite quels sont les problèmes rencontrés lors de la mise en oeuvre du projet (type de parcours, concurrence, fiabilité, efficacité) et la justification des solutions choisies.
- Elle doit présenter comment vous avez testé les fonctionnalités de votre projet. En particulier, vous préciserez l'approche de test utilisée pour vérifier le comportement correct des logiciels développés : tests unitaires de chaque partie/fonction et tests d'intégration par la vérification des interactions entre parties/fonctions.
- Elle doit préciser pour chaque étudiant du groupe les tâches accomplies.

3 Etapes du projet

Ces étapes ainsi que leurs dates limite sont indicatives. Toutefois, le respect de ces dates limite vous permettra d'échelonner correctement votre travail sur le projet durant le semestre.

1ère étape : la semaine lundi 27 octobre 2003.

- Définir l'architecture du projet (communication entre composantes, organisation de la base de données de l'information).
- Mettre en place les formulaires HTML.

- Ecrire le CGI pour la configuration du *crawler* et implémenter cette configuration à partir du fichier de propriétés.
- Ecrire le CGI pour la recherche dans la base de donnée et l'affichage des résultats.

2ème étape : la semaine du lundi 10 novembre 2003.

- Implémenter le *crawler* avec option `--init`.
- Mettre en place le fichier de journalisation du *crawler*.

3ème étape : la semaine du lundi 24 novembre 2003.

- Implémenter les autres options du *crawler*.

Pour tester votre projet, vous disposez de deux serveurs Apache instalés sur les machines `nivose` et `lenteja`.

4 Organisation

- Le projet doit être réalisé par groupe de **TROIS** étudiants.
- Date de remise du projet : vendredi 12 décembre 2003 à 16h.

  Une archive `projet.tar.gz`[2] contenant le projet doit être donnée comme argument de la commande `/ens/sighirea/Internet/remise_projet` qui sera disponible jusqu'à la date indiquée ci-dessus. Toute autre forme de remise du projet sera ignorée.

  L'archive `projet.tar.gz` doit avoir **obligatoirement**[3] la structure suivante :

  - les sources HTML dans le sous-répertoire `html`,
  - les sources CGI dans le sous-répertoire `cgi-bin`,
  - les sources Java ou C dans le sous-répertoire `src`,
  - un ou plusieurs Makefile Unix pour compiler les sources ci-dessus et
  - une documentation HTML dans le sous-répertoire `doc`.
- Le projet sera soutenu la semaine du 15 décembre 2003.

Tout retard dans la remise du projet et/ou toute copie constatée entre les groupes de projet seront sanctionnés sévèrement.

A Exemple d'utilisation du module `javax.swing.text.html`

A.1 `CrawlerCallbackHandler.java`

Le premier fichier définit la classe qui effectue le traitement spécifique pour chaque type d'élément HTML.

[2] Si le répertoire contenant votre projet s'appelle `projet`, utiliser la commande `tar cvf projet.tar projet; gzip projet.tar`

[3] L'absence de la structure ou de l'un de ces éléments aura des conséquences sur la note du projet.

```java
/*
 * Example of use for the API javax.swing.text.html.parser to parse
 * the HTML file in order to collect ancors
 * Author: M. Sighireanu
 */
import java.io.*;
import javax.swing.text.html.*;
/**
 * Class extending the Parser for HTML to do a specific treatment
 * of ancors.
 */
public class CrawlerCallbackHandler extends HTMLEditorKit.ParserCallback {
    boolean isInsideLink = false;

    public CrawlerCallbackHandler(OutputStream os) throws Exception {
    }
    public void handleStartTag(HTML.Tag tag,
                               javax.swing.text.MutableAttributeSet set,
                               int i) {
        // System.out.println("start: " + tag.toString());
        if (tag.equals(HTML.Tag.A)) {
            isInsideLink = true;
            Object link = set.getAttribute(HTML.Attribute.HREF);
            if (link != null)
                System.out.println("Link to: " + link.toString());
        }
    }
    public void handleEndTag(HTML.Tag tag, int i) {
        // System.out.println("end: " + tag.toString());
        if (tag.toString().compareToIgnoreCase("a") == 0) {
            isInsideLink = false;
        }
    }
    public void handleSimpleTag(HTML.Tag tag,
                                javax.swing.text.MutableAttributeSet set,
                                int i) {
        // System.out.println("simple: " + tag.toString());
        if (tag.equals(HTML.Tag.IMG)) {
            Object src = set.getAttribute(HTML.Attribute.SRC);
            Object alt = set.getAttribute(HTML.Attribute.ALT);
            if (src != null)
                System.out.println("Img: " + src.toString());
            if (alt != null)
                System.out.println("\t" + alt.toString());
        }
```

```
    }
    public void handleText(char[] aChar, int i) {
        if (isInsideLink)
            System.out.println(aChar);
    }
}
```

A.2 CrawlerParser.java

Le deuxième fichier instancie le parseur avec le traitement implémenté ci-dessus.

```
/*
 * Example of use for the API javax.swing.text.html.parser to parse
 * the HTML file in order to collect ancors
 * Author: M. Sighireanu
 */
import java.io.*;
import javax.swing.text.html.parser.*;
import CrawlerCallbackHandler;

public class CrawlerParser {
    public static void convert(InputStream inps,
                               OutputStream outps) throws Exception {
        BufferedReader rdr = new BufferedReader(new InputStreamReader(inps));
        // Create the handler
        CrawlerCallbackHandler handler = new CrawlerCallbackHandler(outps);
        // Call the Parser with the new hadler
        new ParserDelegator().parse(rdr, handler, true);
        // Close the stream
        rdr.close();
    }
    public static void main(String [] args) throws Exception {
        // Verify parameters
        if(args.length < 2) {
            System.err.println("Usage : java Crawler <HTMLfile> <OutputFile>");
            System.exit(1);
        }
        // Create the input and output streams
        InputStream InFile = new FileInputStream(args[0]);
        OutputStream OutFile = new FileOutputStream(args[1]);
        convert(InFile, OutFile);
        // Close the streams
        InFile.close();
        OutFile.close();
    }
}
```