BibTeXo

a small community bibtex references management software

Institut Galilée L3 Informatique 2010

Enseignant encadrant : François Malgouyres

Auteur : Vincent Devillierse

vincent.devillierse@gmail.com

1er juillet 2010

Table des matières

1 Présentation du sujet

Dans ce projet nous allons nous intéresser à la gestion de références au format BiBTeX. Ces références sont normalement utilisées dans le cadre de la rédaction d'un document avec LaTeX.

1.1 LaTeX

LaTeX est un système logiciel de création de document. Il est dérivé de TeX, créé en 1977 par le mathématicien Donald E Knuth, lors de la rédaction de sa thèse. La philosophie de LaTeX est de laisser au rédacteur la possibilité de se concentrer uniquement sur le fond de son document, le logiciel assurant la mise en page selon des modèles canoniques (paramétrables à souhait) à l'aide de macro commandes. Il est très utilisé dans le monde scientifique pour la liberté et l'efficacité qu'il apporte ainsi à la rédaction de documents.

1.2 BibTeX

BibTeX est un logiciel et un format de fichier comprenant des références bibliographiques utilisées en collaboration avec LaTeX. Le fichier comprend une liste de références selon un format prédéfini, chacune comprenant le type de document, une clé de référence et un certain nombre de champs (obligatoires ou non suivant le type de document).

Exemple de référence :

```
@Book{LC-fr,
@MASTERTHESIS{e1-207a-f00,
author = {Max Rydahl Andersen and Claus Nyhus Christensen and Kristian Lykkegaard Sorensen},
title = {Internal documentation in an Elucidative environment},
school = {Aalborg University},
year = {2000},
month = {June},
note = {Available from http://dopu.cs.auc.dk}
}
```

Ainsi avec LaTeX, les références bibliographiques seront automatiquement générées. Pour ce faire, l'auteur utilise la commande `\cite{nom_ref}`, là où doivent apparaitre ces citations, `\bibliographystyle{smfplain}` pour en définir le style et`\bibliography{biblio}` pour indiquer quel est le fichier de bibliographies à consulter. Lors de la première compilation du document LaTeX, les appels de références LaTeX `\cite{}` sont lus et conservés dans un fichier .aux. Le fichier .aux est alors compilé avec BibTeX, pour obtenir un fichier .bbl et un fichier .blg (fichier de log), à partir de la bibliographie contenue dans le fichier BibTeX. Lors de la prochaine compilation avec LaTeX, les références seront créées.

La syntaxe des références bibtex est extrêmement permissive et soumise à de grands nombres de variantes, aussi a-t-il fallu légèrement la restreindre pour les besoins du projet. Un exemple, les champs doivent être encadrés par des accolades, sauf dans le cas du champ "month" si le mois est représenté par ses trois lettres normalisées (on est un peu dubitatif sur la simplicité d'utilisation apportée à l'utilisateur par le biais de ce genre d'exceptions).
La structure, les types de références ainsi que leurs champs optionnels et obligatoires sont décrits en Annexes 1 et 2 du manuel d'installation et d'utilisation.

Il nous est apparu absolument nécessaire que le logiciel gère l'unicité des clefs (comme dans un SGBD). En effet, il nous fallait pouvoir déterminer, de manière unique la référence qu'un utilisateur souhaitait modifier ou supprimer. Il aurait été inadéquat d'imposer l'unicité d'un champ auteur ou titre, plusieurs livres pouvant avoir le même titre, un même auteur pouvant écrire plusieurs livres etc... Le choix s'est donc tout natrurellement porté sur le champ clef. De plus, dans le cadre de l'utilisation de ces références par la commande `\cite{}`, il est évident que posséder plusieurs références ayant la même clef risque d'occasioner si non des erreurs, du moins un résultat éloigné de ce que l'utilisateur désirait réellement.

1.3 Environnements logiciels et matériels

Le logiciel est programmé en langage C, un langage de bas niveau. Correctement utilisé, il permet, entre autres, de manipuler la mémoire avec une très grande précision et d'augmenter ainsi la performance et l'efficacité des programmes.

L'interface graphique a été réalisée à l'aide de GTK+, un ensemble de bibliothèques logicielles libres. Cette bibliothèque est notamment utilisée par l'environnement de bureau GNOME. Le choix s'était d'abord porté sur une interface graphique en Java mais le but du projet était aussi de découvrir de nouvelles notions, et si le langage Java nous était déjà connu, GTK non.

Le logiciel a été développé sous Linux (Ubuntu). Si nous nous étions engagés à fournir une version alternative fonctionnant sous Windows, nous n'avons malheureusement pas eu le temps de la développer avant la date buttoir et nous le regrettons.

Toutes les informations sur les prérequis logiciels nécessaires au bon fonctionnement de notre projet peuvent être trouvées dans le manuel d'installation et d'utilisation (partie 1).

1.4 Objectifs

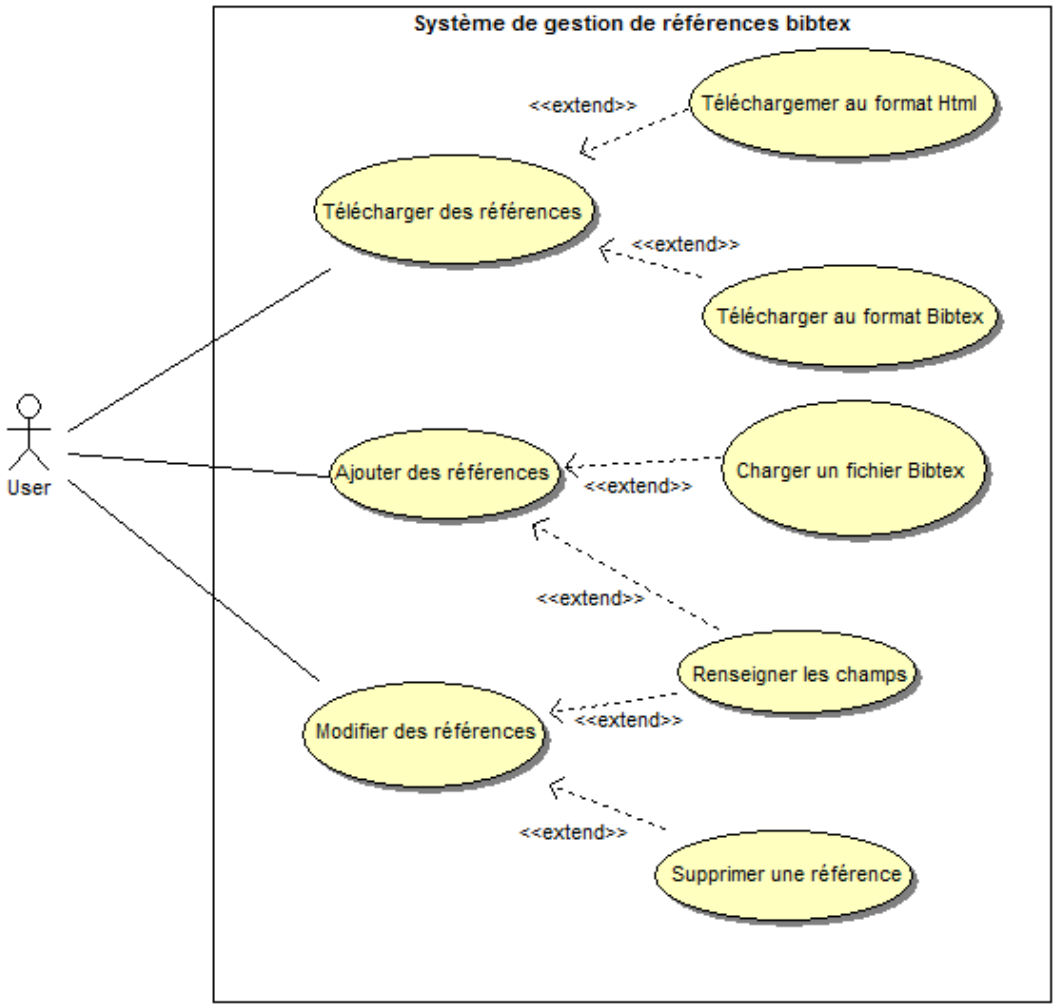
Notre but va être de développer un logiciel permettant à un utilisateur de gérer ses références BibTeX sur un serveur. En effet, les listes de références bibliographiques peuvent, à la longue, devenir de plus en plus volumineuses et il devient alors important de disposer d'un outil permettant de les gérer.

Le but de ce logiciel sera d'accompagner l'utilisateur dans la gestion de ses références bibliographiques.

Le stockage sur un serveur permettra aussi de conserver les références sur internet, afin de les avoir accessibles partout, de les partager avec d'autres utilisateurs et éventuellement d'enrichir la base existante.

Le format BibTeX n'est pas visuellement pertinent pour un utilisateur souhaitant consulter ses références bibliographiques. Aussi sera-t-il possible de générer des fichiers HTML avec ses références, pour une consultation plus agréable et efficace.

Les fonctionnalités nécessaires seront donc l'ajout de références (une par une ou par fichiers), la modification, la suppression, la recherche et téléchargement de références (au format bibtex et/ou au format html). Vous trouverez page suivante le schéma associé, issu de l'analyse préliminaire du projet.


Système de gestion de références bibtex
Téléchargemer au format Html
<<extend>>
Télécharger des références
<<extend>>
Télécharger au format Bibtex
User
Ajouter des références
<<extend>>
Charger un fichier Bibtex
<<extend>>
Renseigner les champs
Modifier des références
<<extend>>
<<extend>>
Supprimer une référence

2 Conception Architecturale

Notre projet utilise une architecture client/serveur. Le serveur fait appel à un exécutable Flex/Bison permettant d'analyser un fichier de références bibtex et de transformer ces références au format HTML.
On peut donc considérer trois sur-modules principaux : Client, Serveur et Flex/Bison.

2.1 Client et Serveur : des modules "communs"

Les modules client et serveur communiquent lors de toutes les opérations de manipulation de fichiers bibtex. Ainsi les sous-modules "Ajouter Référence" sont présents et intimement liés et sur le client et sur le serveur. Chacun de ces sous-modules se retrouve dans un fichier à part. La liste de ces sous-modules est la suivante :

- "Ajouter Références" fichiers `bibtexo_client_add_reference` et `bibtexo_server_add_reference` : ajout de références
- "Modifier Références" fichiers `bibtexo_client_modify_reference` et `bibtexo_server_modify_reference` : modification de références
- "Supprimer Références" fichiers `bibtexo_client_delete_reference` et `bibtexo_server_delete_reference` : suppression de références
- "Téléchargement Références"/"Envoi références" fichiers `bibtexo_client_download_reference` et `bibtexo_server_send_reference` s'occuperont de l'envoi et de la réception de listes de fichiers bibetx/html.

2.2 Modules Client

Le client possède également ses propres sous-modules.

Le module "Main" fichier bibtexo_client_main initialisera la fenêtre principale du logiciel, l'invite de connexion et gérera la fermeture du logiciel (en erreur ou de manière standard).
Le module "Connexion" fichiers bibtexo_client_connexion s'occupera de la connexion du client au serveur.
Le module "A propos" fichier bibtexo_client_apropos gérera l'affichage de la fenêtre du même nom.

2.3 Modules Serveur

Le serveur possède également ses propres sous-modules.

Le module "Main" fichier bibtexo_server_main initialisera le serveur et contient la fonction de service qui traitera les requêtes du client en faisant appel aux fonctions adéquates.
Le module "Connexion" fichiers bibtexo_server_connexion s'occupera de la connexion des clients au serveur.
Le module "Outils" fichier bibtexo_server_tool contient des fonctions utilisées dans les autres modules de manière récurrentes. Elles ont été regroupées dans ce fichier.

2.4 Modules Flex/Bison

Le sur-module "Flex/Bison" comprend deux modules, le module "Analyse Lexicale" et le module "Analyse Syntaxique", réalisant respectivement l'analyse lexicale et l'analyse syntaxique d'un fichier bibtex.

3 Conception détaillée

Pour une meilleure compréhension, le code a été très largement et precisément commenté, n'hésitez pas à vous y reporter.
Le code et les commentaires ont été réalisés intégralement en anglais, c'est la langue la plus répandue parmi les programmeurs du monde, permettant alors de partager son travail et d'en faire profiter le plus de personnes possibles.

3.1 Module Client : sous-modules

3.1.1 Module Main

Ce module correspond au fichier bibtexo_client_main. Il va initialiser la fenêtre principale du logiciel et lancer la boucle événementielle de GTK+. A ce moment, seuls les modules "Connexion" et "Apropos" sont accessibles. Les boutons permettant l'accès aux autres modules sont verrouillés tant que la connexion avec le serveur n'a pas été établie. La fonction main fait appel à la fonction `retrieve_distant_connexion_data` du module "Connexion" pour récupérer les derniers paramètres de connexion réussie à un serveur distant (si il n'y en pas, un texte par défaut sera choisi) et les placer dans les champs "adresse" et "port" du bloc de connexion à un serveur distant. Cliquer sur la croix en haut de la fenêtre ou sur le bouton "Quitter" fera quitter le programme de manière standard. La croix force la fermeture : fonction `on_destroy_main_no_confirm`, tandis que par le bouton une confirmation sera préalablement demandée : fonction `on_destroy_main_confirm`. Lorsque l'on fait appel à ces fonctions (et que l'on confirme la fermeture dans le cas du bouton "Quitter"), une requête de fermeture de connexion est envoyée au serveur, la socket est fermée, la boucle événementielle arrêtée et le programme quitte.

Lorsque qu'une fonction est appellée suite à la sélection d'un bouton, GTK autorise deux arguments pour cette fonction : le bouton en lui même (type `GtkWidget`) et un argument de données `p_data`. Nos fonctions ont souvent besoin de manipuler de nombreuses variables, aussi a-t-on décidé de manipuler une structure d'arguments contenant toutes ces variables. L'argument envoyé à chaque fonction est un pointeur sur cette structure. La structure principale possède des variables de type structure correspondant aux variables nécessaires à chaque module ; ceci pour une meilleure compréhension (une structure pour les arguments du module "Modifier" , une pour "Ajouter" etc...). Ainsi on peut utiliser les propriétés de la boucle événementielle de GTK sans perdre aucune donnée qui soit nécessaire. Voir le fichier `bibtexo_client_main.h` pour plus de détails sur cette structure d'arguments.

Le programme démarre par une analyse des arguments par `get_opt_long`, les options possibles étant `--help` (affiche une aide sur la sortie standard et quitte normalement) et `--version` (affiche la version du logiciel sur la sortie standard et quitte normalement).

Cliquer sur le bouton "A propos" appellera la fonction `display_apropos` du module "A propos".

Cliquer sur le bouton "Connecter" appellera la fonction `connexion` du module "Connexion".

3.1.2 Module Connexion

Ce module correspond au fichier `bibtexo_client_connexion`. Il peut être appelé par sa fonction `connexion` en cliquant sur le bouton "Connecter". Suivant le choix de l'utilisateur entre connexion distante ou locale, la fonction va lire les champs associés et essayer de se connecter au serveur spécifié. Si l'adresse ou le nom du serveur est inconnu un message d'erreur sera affiché et la fonction quitte normalement. Sinon, la fonction fait appel à la fonction `create_socket` en lui donnant le type de la socket à créer. Cette fonction va créer, lier et renvoyer la socket. Puis, la première fonction fait un appel à `connect` pour se connecter au serveur.
Si la connexion échoue, un message est spécifié à l'utilisateur et la fonction quitte normalement. Sinon, un message de succès est affiché. Si la connexion était une connexion à un serveur distant la fonction fait appel à la fonction `save_distant_connexion_data` avec comme paramètres l'adresse ou nom du serveur et le port. Cette fonction écrira dans le fichier `/distant_connexion/distant.txt` ces informations afin qu'elles soient réutilisées lors du prochain lancement du logiciel. Pour finir, la fonction fait appel à la fonction `connexion_successful`.

La fonction `connexion_successful` remplace le contenu central de la fenêtre principale (supprimant ainsi l'invite de connexion) et déverrouille les boutons du menu, toutes les fonctionnalités sont alors accessibles.

La fonction `exit_on_error` s'occupera de la fermeture du programme en cas d'erreur critique. Tous les appels systèmes sont testés et si l'un d'eux échoue la fonction fera appel à cette fonction en lui transmettant une chaîne de caractères reprenant le code de l'erreur. Le code de l'erreur comprendra l'appel système ayant échoué, la fonction et le fichier dans lesquels il se trouve. Après affichage de l'erreur dans une fenêtre, une requête de fermeture de connexion est envoyée au serveur, la socket est fermée, la boucle événementielle arrêtée et le programme quitte.

3.1.3 Module A propos

Ce module correspond au fichier `bibtexo_client_apropos`. Il est appelé lorsque l'utilisateur clique sur le bouton "A propos" du menu de la fenêtre principale ; la fonction `display_apropos` est alors lancée.
Cette fonction va vérifier la variable `Apropos_display` de la structure `apropos`. Si la variable vaut TRUE, la fenêtre a déjà été ouverte et elle est déiconifiée et la fonction quitte, sinon elle est créée. Ceci permet d'éviter l'ouverture simultanée de plusieurs fenêtres "A propos". On peut quitter cette fenêtre en cliquant sur la croix en haut à droite ou en cliquant sur le bouton "Ok". Un appel est alors fait à la fonction `on_close_apropos`, qui va mettre la précédente variable à FALSE. Cette fenêtre comprend deux autres boutons "Licence" et "Auteurs".

Cliquer sur le bouton "Licence" appellera la fonction `display_licence`. Cette fonction va vérifier la variable `Licence_display` de la structure `licence`. Si la variable vaut TRUE, la fenêtre a déjà été ouverte et elle est déiconifiée et la fonction quitte, sinon elle est créée. Ceci permet d'éviter l'ouverture simultanée de plusieurs fenêtres "Licence". Cette fenêtre contient des informations sur la licence du logiciel. On peut quitter cette fenêtre en cliquant sur la croix en haut à droite ou en cliquant sur le bouton "Ok". Un appel est alors fait à la fonction `on_close_licence`, qui va mettre la précédente variable à FALSE.

Cliquer sur le bouton "Auteurs" appellera la fonction `display_authors`. Cette fonction va vérifier la variable `Authors_display` de la structure `autors`. Si la variable vaut TRUE, la fenêtre a déjà été ouverte et elle est déiconifiée et la fonction quitte, sinon elle est créée. Ceci permet d'éviter l'ouverture simultanée de plusieurs fenêtres "Auteurs". Cette fenêtre contient des informations sur les auteurs du projet. On peut quitter cette fenêtre en cliquant sur la croix en haut à droite ou en cliquant sur le bouton "Ok". Un appel est alors fait à la fonction `on_close_autors`, qui va mettre

la précédente variable à FALSE.

Les trois variables précédentes sont initialisées à FALSE, dans la fonction "main" du module du même nom.

3.2 Module Serveur : sous-modules

3.2.1 Module Main

Ce module correspond au fichier `bibtexo_server_main`.
La fonction `main` démarre par une analyse des arguments en utilisant `get_opt_long`, les options possibles étant `--help` (affiche une aide sur la sortie standard et quitte normalement) et `--version` (affiche la version du logiciel sur la sortie standard et quitte normalement).

Si aucune de ces options n'est spécifiée, la fonction `main` initalise la liste chainée de références. Les références sont représentées sous la forme d'une structure :

```
  linked list of references
struct Reference
{
      Type type;
      char *key;
      struct Field field;
      struct Reference *next;
};

structure for the different types of a reference
enum Type
{
ARTICLE, BOOK, BOOKLET, CONFERENCE, INBOOK,
INCOLLECTION, INPROCEEDINGS, MANUAL, MASTERTHESIS,
MISC, PHDTHESIS, PROCEEDINGS, TECHREPORT, UNPUBLISHED
};

structure for the list of fields of a reference
typedef struct Field Field;
struct Field
{
char *author;
char *title;
char *address;
char *annote;
char *booktitle;
char *chapter;
char *editor;
char *howpublished;
char *institution;
char *journal;
char *month;
char *note;
```

```
char *number;
char *organization;
char *pages;
char *publisher;
char *school;
char *series;
char *type;
char *volume;
char *year;
char *edition;
};
```

La fonction `main` crée alors la socket par appel à la fonction `create_socket` du module serveur "Connexion" et place le serveur en attente de connexion d'un client. Lorsque l'appel à `accept` réussi, un thread est créé pour gérer le client, le thread va exécuter la fonction de service `service` avec comme argument la socket `socket_service`. Le choix d'un serveur multi-thread plutot qu'un serveur multi-processus (par appels à `fork`) s'est imposé de lui même. En effet, par appel à `fork`, la liste de références aurait été dupliquée par chaque processus, les modifications apportées par un client n'auraient été visible que par lui même. Pour réaliser un véritable logiciel communautaire, il fallait donc bien utiliser une liste chaînée de référence partagée en tant que variable globale dans un serveur multi-thread.

La fonction `service` va commencer par créer les noms des trois fichiers `file_to_save`, `bibtex_to_send` et `html_to_send` dont nous aurons besoin suivant les requêtes de l'utilisateur. Ces fichiers seront créés dans le dossier "temporary_files" et doivent être uniques pour chaque client. Comment s'assurer que ces fichiers seront uniques ? Il a été décidé d'ajouter le numéro de la socket utilisée pour communiquer avec le client. En effet cette socket est nécessairement unique pour chaque client !

La fonction service rentre alors dans une boucle qui va lire les requêtes de l'utilisateur et faire appel aux fonctions adéquates tant que les variables `end` et `error` valent FALSE. Il y a deux manières de quitter cette boucle. La manière standard est de recevoir la requête "quitter" du client qui met la variable `end` à TRUE. L'autre manière correspond aux cas d'erreurs critiques. Comme sur le client, tout appel système est testé et si il échoue, la fonction dans laquelle il se trouvait retourne -1 et affiche un message d'erreur sur la sortie d'erreur standard comprenant l'appel ayant échoué, le type de l'erreur, le fichier et la fonction où se trouvait l'appel. Tout retour de fonction est alors testé et si il renvoie -1 la fonction retournera -1 de proche en proche, jusqu'à revenir dans la fonction `service` qui mettra la variable `error` à TRUE.
Dans tous les cas, la fonction quittera alors la boucle et terminera le thread après avoir clos la socket. Il aurait été ennuyeux de quitter le programme `bibtexo_server`, car cela aurait détruit la liste chaînée des référence, handicapant tous les utilisateus pour l'erreur d'un seul. Cette solution, plus pragmatique nous permet d'offrir un meilleur service aux utilisateurs.

La liste des requêtes que le serveur peut recevoir est la suivante (les requêtes se présentent sous forme de numéros) :

- requête 1 : ajouter une référence manuellement, appel à la fonction `add_reference_manually_type` du module "Ajouter Référence"
- requête 2 : vérifier l'unicité d'une clef, appel à la fonction `single_key` du module "Tool"
- requête 3 : ajouter un fichier de références, appel à la fonction `add_file_bibtex` du module "Ajouter Référence"
- requête 4 : connaître le type d'une référence, appel à la fonction `get_type_reference` du module "Modifier Référence"
- requête 5 : modifier une référence, appel à la fonction `modify_reference_get_data` du module "Modifier Référence"
- requête 6 : connaître les champs d'une référence donnée, appel à la fonction `send_data_reference` du module "Modifier Référence"

- requête 7 : supprimer une référence, appel à la fonction `delete_reference` du module "Supprimer Référence"
- requête 8 : téléchargement de références au format bibtex, appel aux fonctions `search_reference` et `send_file_bibtex` du module "Send Référence"
- requête 9 : téléchargement de références au format HTML, appel aux fonctions `search_reference` et `send_file_html` du module "Send Référence"
- requête 10 : se déconnecter, la variable `end` est mise à TRUE

3.2.2 Module Connexion

Ce module correspond au fichier `bibtexo_server_connexion`. La fonction `create_socket` va créer la socket du serveur, la lier, écouter dessus et la retourner.

3.2.3 Module Tool

Ce module correspond au fichier `bibtexo_server_tool` et contient des fonctions de type "boîte à outils" pour le serveur.

La fonction `init_reference` permet d'initialiser tous les champs d'une référence.

La fonction `get_type_reference` permet de retourner le type de la référence ayant la chaine de caractère passée en argument pour clef. On parcourt la liste de références et quand le champ clef est égal à la chaine on retourne le type de la référence.

La fonction `get_char_from_type` retourne un type de référence passé en argument sous forme de chaine de caractères.

La fonction `uppercase` renvoie la chaine de caractères passée en argument avec tous les caractères mis en majuscules.

La fonction `single_key` retourne 0 si une référence a la chaine de caractères passée en argument pour clef, 1 sinon. Elle nous permet de vérifier l'unicité d'une clef.

3.3 Modules "communs"

3.3.1 Module Ajouter Références

Ces modules correspondent aux fichiers `bibtexo_client_add_reference` et `bibtexo_server_add_reference`. Dans la partie client, cliquer sur le bouton "Ajouter" du menu principal appellera la fonction `add_reference_choice` du module client "Ajouter Références". Cette fonction ouvre une fenêtre de dialogue "Mode d'ajout" invitant l'utilisateur à choisir entre ajouter des références une par une ou ajouter un fichier bibtex. Si l'utilisateur choisit le premier mode, un appel est fait à la fonction `add_reference_manually`, pour l'autre c'est la fonction `add_file`. Enfin, cliquer sur le bouton "Annuler" ou la croix de la fenêtre fait terminer la fonction sans autres effets.

La fonction `add_reference_manually` va se charger de détruire l'affichage central de la fenêtre principale, puis y placer une liste déroulante de type de références et un bouton "Valider", bouton qui appellera la fonction `add_reference_manually_type`.

La fonction`add_reference_manually_type` va rechercher le type de références que l'utilisateur a sélectionné dans la précédente liste déroulante par appel à la fonction `combo_box_active_get_text`. Si l'utilisateur n'a rien choisi, la fonction quitte sans effet. Sinon, selon le type de références choisi, la fonction va ajouter au centre de la fenêtre centrale les champs obligatoires et optionnels correspondant (avec un texte par défaut), un champ clef et un bouton "Envoyer". Tous les champs présents à l'écran et le bouton sont sauvés dans la structure `add_entry` de la liste d'arguments (sauf le champ "Aucun" de la liste des champs obligatoires de la référence MISC, purement décoratif). Cliquer sur le bouton "Valider" appellera la fonction `add_reference_manually_check`.

La fonction `add_reference_manually_check` va se charger d'opérer une vérification sur le contenu des champs (obligatoires, optionnels et clef) que l'utilisateur a entré. L'envoi des références n'aura lieu que si et seulement si il n'y a aucune erreur. Le contenu des champs est analysé (grâce au contenu de la structure `add_entry`, on peut appeler le texte contenu dans les champs sauvegardés). Selon le type de champ choisi, le contenu des champs obligatoires doit différer du contenu par défaut et ne pas être vide. Pour chaque champ où cela est le cas, on incrémente la variable `nbError`, permettant de compter les erreurs et on ajoute le type de champ au tableau de char `tabError`, tous deux sauvegardés dans la structure `add_error`. Le champ clef subit le même traitement avec une opération supplémentaire, on vérifie que la clef n'existe pas déjà sur le serveur en lui envoyant la requête 2 ; si elle existe déjà c'est un nouveau type d'erreur qui incrémente `nbError` et la variable `exist_key` de la structure `add_error` est mise à TRUE.
Si il y a au moins une erreur, la fonction fait appel à la fonction `display_error` ; sinon, elle fait appel à la fonction `send_entry_manual_X` où X est le type de la référence.

La fonction `display_error` va parcourir les éléments de la structure `add_error` et créer, à partir de ces éléments, une fenêtre d'erreur listant les champs non remplis et précisant le cas échéant si la clef que l'utilisateur a choisie existait déjà sur le serveur.

Les fonctions `send_entry_manual_X` vont se charger d'envoyer les données de la nouvelle référence au serveur pour qu'il les ajoute à sa liste de références. On envoie tout d'abord la requête associée (1) au serveur, puis le type de la référence qui est lu dans la fonction `add_reference_manually_type` du module "Ajouter Référence" du serveur. Selon le type de la référence, cette fonction fera appel à la fonction `add_entry_manual_X` où X est le type de la référence. Le serveur crée une nouvelle structure `référence` et attend de la compléter avec les informations du client. Le client va rechercher le contenu des champs (grâce à la structure `add_rentry`) et, suivant le type de la référence, envoyer le contenu des champs obligatoires au serveur. Puis, les champs optionnels ne sont envoyés et lus que si la taille de leur contenu est supérieure à 0. Une fois que l'envoi est terminé, le serveur ajoute cette nouvelle référence à sa liste. Pour le client, un message de succès est affiché et le bouton "Envoyer" est verrouillé en attendant que le client revalide un type de référence où il sera recréé.

La fonction `add_file` du module client "Ajouter Référence" va se charger de détruire l'affichage central de la fenêtre principale puis y placer un bouton "Parcourir" appelant la fonction `add_file_selection` et un bouton "Envoyer" appelant la fonction `send_file_to_server`. Ce dernier est sauvegardé dans la structure `add_file` et est verrouillé tant qu'on n'a pas cliqué sur le bouton "Parcourir".

La fonction `add_file_selection` va ouvrir une fenêtre de sélection de fichier avec deux boutons "Valider" et "Annuler". Le bouton "Annuler" détruit cette fenêtre et la fonction stoppe. Le bouton "Valider" appelle la fonction `add_file_path`. Le bouton "Envoyer" de la fenêtre précédente est déverrouilé.

La fonction `add_file_path` va nous permettre de récupérer le chemin du fichier choisi par l'utilisateur qui sera stocké dans la variable `file_to_send`. Un problème s'est posé, car la fonction permettant de récupérer le chemin du fichier ne peut posséder d'arguments de données. Aussi la variable `file_to_send` a-t-elle du être utilisée comme une variable globale.

La fonction `send_file_to_server` va envoyer une requête au serveur pour lui demander l'envoi d'un fichier bibtex (3), puis, envoyer le contenu du fichier (si le fichier ne peut être ouvert, un message d'erreur apparait et la fonction quitte sans autres effets). La fonction `add_file_bibtex` du module serveur "Ajouter référence", appelée depuis la fonction `service`, va lire le contenu du fichier bibtex et l'enregistrer dans le fichier `bibtex_to_save`. Une fois l'envoi et la réception terminés, le client se met en attente du serveur et la fonction `add_file_bibtex` du serveur fait appel à la fonction `read_file_bibtex`.

La fonction `read_file_bibtex` va analyser le fichier venant d'être créé. Pourquoi ne pas avoir analysé le fichier depuis le client et envoyé les références reconnues ? Car cela aurait répété de manière exponentielle des appels à `read` et `write`, alourdissant les fonctions et risquant à terme de créer des erreurs. La fonction va donc lire le fichier et ajouter au fur et à mesure les références qu'elle rencontre, reconnaissant la structure des champs, des types de références, des clefs... (se reporter au code commenté pour les détails précis de l'algorithme). Pour chaque clef lue, un appel à `single_key` du module "Tool" est fait et si la clef existe déjà une variable `ref_valid` est mise à FALSE. Dans ce cas, la référence n'est pas ajoutée et la clef est ajoutée à une liste chainée de chaine de caratères, représentant toutes les références existant déjà sur le serveur. A la fin de l'analyse du fichier, cette liste est envoyée au client, c'est la réponse qu'il attendait. Si la liste a une taille de 0, le client annonce que l'intégralité du fichier a été ajoutée. Sinon, il affiche à l'utilisateur la liste de ces références.

3.3.2 Module Modifier Références

Ces modules correspondent aux fichiers `bibtexo_client_modify_reference` et `bibtexo_server_modify_reference`. Dans la partie client, cliquer sur le bouton "Modifier" du menu principal appellera la fonction `modify_reference` du module client "Modifier Références". Cette fonction va se charger de détruire l'affichage central de la fenêtre principale, puis y placer un champ "Clef" et un bouton "Valider", bouton qui appellera la fonction `modify_reference_check_key`. Le champ est sauvé dans la structure `modify_ref` et la variable `delete` de la structure `delete_reference` est mise à FALSE (elle nous permet de faire la différence entre modifier et supprimer une référence).

La fonction `modify_reference_check_key` va utiliser la structure `modify_ref` pour obtenir le contenu du champ "Clef" et le sauver dans la variable `previous_key`. Si le champ a été laissé vide ou avec le texte par défaut, un message d'erreur spécifique est affiché et la fonction quitte sans autres effets. Puis la fonction envoie une requête au serveur pour savoir si la clef existe (requête 2). Le serveur fait appel à la fonction `single_key` de son module "Tool" et renvoie la réponse au client. Si la clef n'existe pas, la référence n'existe pas et la fonction quitte sans autres effets (il est évidemment impossible de modifier une référence qui n'existe pas). Si la clef existe, une nouvelle requête est envoyée au serveur, une requête pour connaître le type de la référence (requête 4). Le serveu fait successivement appel à `get_type_reference` et `get_char_from_type` du module serveur "Tool" et finalement renvoie au client le type de la référence, que celui-ci sauve dans la structure `modify_ref`. Le client fait alors appel à la fonction `modify_reference_get_data` et envoie au serveur la requête 5 "connaître les champs d'une référence". Le serveur, recevant cette requête, va faire appel à la fonction `send_data_reference` du module serveur "Modifier référence". La fonction `modify_reference_get_data` du client envoie alors la clef de la référence. la fonction `send_data_reference` du serveur parcourt la liste de références, compare les clefs et, quand la référence est trouvée, envoie le contenu des champs de la référence. Si la taille d'un champ vaut 0, seule la taille est envoyée pour signifier que la référence est vide. Le client sauve le contenu de ces champs dans la structure `modify_reference`. Une fois l'envoi terminé les deux fonctions quittent.

Depuis la fonction client `modify_reference_check_key`, après l'appel à `modify_reference_get_data`, un appel est fait à `modify_reference_display`. Cette fonction va détruire le centre de la fenêtre principale et, selon le type de références choisi, ajouter les champs obligatoires et optionnels correspondant (avec le texte reçu dans `modify_reference_get_data` et sauvé dans la structure `modify_reference`) ainsi qu'un champ clef qui sera lui aussi sauvegardé. La variable `delete` de la structure `delete_reference` étant à FALSE, un bouton "Modifier" est ajouté à la fenêtre et sauvé dans la structure `modify_reference`. L'utilisateur peut alors modifier les champs comme il l'entend. Cliquer sur le bouton "Modifier" appellera la fonction `modify_reference_check_data`.

Cette fonction aura un comportement similaire à la fonction `add_reference_manually_check` du module client "Ajouter Référence". A la différence près que la verification de l'unicité de la clef ne se fera que si l'utilisateur a remplacé la clef initiale. Si il ne l'a pas fait, la clef est forcément unique et disponible vu qu'elle existait déjà avant. Si plus d'une erreur est détecté, la fonction fait appel à la fonction `display_error` du module "Ajouter Référence". Sinon, la fonction fait appel à la fonction `modify_reference_send_modification`.

Cette fonction commence par envoyer une requête au seveur "modifier une référence existante" (requête 6). Le serveur, recevant cette requête va faire appel à la fonction `modify_reference_get_data`. Le client envoie la clef au serveur, qui va parcourir la liste des références jusqu'à trouver celle ayant le champ clef pour clef. Une fois ceci fait, le client envoie la taille du contenu des champs au serveur, suivi de leur contenu, une taille de 0 signifiant que le champ est vide, aucun contenu n'est alors ni lu, ni envoyé. Une fois l'envoi et la réception terminés, un message de succès est affiché au client, le bouton "Modifier" est verrouillé (tant que l'utilisateur n'aura pas choisi une nouvelle référence valide à modifier) et les deux fonctions quittent.

3.3.3 Module Supprimer Références

Ces modules correspondent aux fichiers `bibtexo_client_delete_reference` et `bibtexo_server_delete_reference`. Dans la partie client, cliquer sur le bouton "Supprimer" du menu principal appellera la fonction `delete_reference` du module client "Supprimer Références". Cette fonction a le même fonctionnement que la fonction `modify_reference` du module client "Modifier Référence", à la différence près que la variable `delete` de la structure `delete_reference` est mise à TRUE. On utilise les mêmes fonctions que dans ce précédent module pour continuer.

Lors de l'appel qui sera fait à `modify_reference_display`, du fait de la valeur TRUE de la précédente variable, ce n'est pas un bouton "Modifier" qui sera ajouté à la fenêtre, mais un bouton "Supprimer". Les champs sont affichés comme dans le module "Modifier référence", mais ici non pas pour être modifiés, mais pour que l'utilisateur vérifie la référence avant de la supprimer. Le champ clef qui était ajouté précédemment n'a ici pas lieu d'être et n'est pas ajouté. Le bouton "Supprimer" est sauvegardé dans la structure `delete_ref`. Cliquer sur ce bouton appellera la fonction `delete_reference_check`.

Cette fonction proposera à l'utilisateur de confirmer sa volonté de supprimer la référence avec une boite de dialogue. Si l'utilisateur clique sur le bouton "Annuler", la fonction quitte sans autres effets et la référence ne sera ainsi pas supprimée. Si il clique sur "Oui", la fonction fait appel à la fonction `delete_reference_send_key` du module client "Supprimer Référence".

Cette fonction va envoyer la requête "supprimer une référence" au serveur (requête 7). Cela aura pour effet que le serveur fasse appel à la fonction `delete_reference` du module serveur "Supprimer Référence". Le client envoie alors la clef au serveur (la clef a été conservée dans la structure `modify_ref`). Le serveur va lire cette clef et tenter de supprimer la référence de sa liste chainée. Si la liste ne comporte qu'une référence, elle est détruite et la liste de référence vaut `NULL`. Sinon, si la référence à supprimer est la première de la liste, la référence est détruite et la liste commence

dorénavant au second élément de la liste. Sinon, la liste est parcourue jusqu'à trouver la référence recherchée. A ce moment, la référence précédent la référence à supprimer aura son élément `next` poitant sur la référence suivant celle à supprimer. La référence à supprimer est alors supprimée. Du côté du client, un message de succès est affiché et le bouton "Supprimer" est verrouillé tant que l'utilisateur n'aura pas choisi une nouvelle référence valide à supprimer.

3.3.4 Modules Télécharger Références et Envoyer références

Ces modules correspondent aux fichiers `bibtexo_client_download_reference` et `bibtexo_server_send_reference`. Dans la partie client, cliquer sur le bouton "Télécharger" du menu principal appellera la fonction `download_reference` du module client "Télécharger Références". Cette fonction va se charger de détruire l'affichage central de la fenêtre principale, puis y placer trois champs "Clef", "Auteurs" et "Titre", ainsi que trois boutons "HTML", "BibTeX" et "Télécharger", ce dernier étant initialement verrouillé. Les trois champs ainsi que le bouton "Télécharger" sont sauvegardés dans la structure `download_ref`. Ces trois champs permettent de spécifier des critères de recherche de références. Cliquer sur le bouton "HTML" appellera la fonction `download_file_html`, cliquer sur le bouton "BibTeX" appellera la fonction `download_file_bibtex`.

Ces deux fonctions sont quasiment identiques. Elles vont ouvrir une fenêtre de sélection de fichiers avec comme nom de fichier par défaut `bibtex_saving.bib` pour l'une, et `bibtex_saving.html` pour l'autre. Cliquer sur le bouton "Annuler" de cette fenêtre la fermera sans autres effets. Cliquer sur le bouton "Valider" appellera la fonction `download_file_name`. Suivant le type de bouton cliqué, la variable `bibtex` de la structure `download_ref` est mise à TRUE ou FALSE. Le précédent bouton "Télécharger" est déverrouillé; cliquer sur ce bouton appellera la fonction `download_file_check_name`.

La fonction `download_file_name` va nous permettre de récupérer le chemin du fichier choisi par l'utilisateur dans la variable globale `file_to_save`. Il nous a fallu utiliser une variable globale car la fonction `download_file_name` ne pouvait prendre notre liste d'arguments comme argument. La variable `file_to_save` est découpée de manière à nous permettre de séprarer le nom du fichier et le nom du répertoire dans lequel il se trouve. On utilise alors une structure `dirent` pour explorer le contenu du répertoire en question et comparer les noms des fichiers avec celui choisi par l'utilisateur. Si deux noms sont identiques, la variable `file_already_exist` est mise à TRUE, sinon à FALSE. Pour les mêmes raisons que précédemment, cette variable est globale.

La fonction `download_file_check_name` va essayer d'ouvrir le fichier représenté par le chemin `file_to_save`. Si l'ouverture échoue, un message d'erreur est affiché et la fonction quitte sans autres effets. Si la variable `file_already_exist` vaut TRUE, une boite de dialogue est affichée, demandant à l'utilisateur si il souhaite écraser le fichier. Si il répond "Non", la fonction quitte sans autres effets. Sinon, ou si le fichier doit être créé, la fonction appelle la fonction `download_file_receive_data`.

La fonction `download_file_receive_data` va verrouiller le bouton "Télécharger", puis, si la variable `bibtex` de la structure `download_ref` vaut TRUE, envoyer au serveur une requête "télécharger un fichier bitex" (requête 8), sinon la requête sera "télécharger un fichier html" (requête 9).

Lorsqu'il reçoit la requête 8, le serveur fait appel à la fonction `search_reference` du module serveur "Envoyer Références", qui va lire le contenu des champs "Clef", "Auteurs" et "Titre" envoyés par le client. Si un champ est vide, il doit être ignoré. Pour cela, le serveur construit un mode de recherche suivant les tailles des contenus :

- mode 0, toutes les références serotn téléchargées
- mode 1, seules les références dont le champ clef contient le champ clef envoyé depuis le client seront téléchargées
- mode 2, seules les références dont le champ auteur contient le champ auteur envoyé depuis le client seront téléchargées
- mode 3, combine les modes 1 et 2
- mode 4, seules les références dont le champ titre contient le champ titre envoyé depuis le client seront téléchargées
- mode 5, combine les modes 1 et 4
- mode 6, combine les modes 2 et 4
- mode 7, combine les modes 1, 2 et 4

Seuls les champs clefs sont comparés caractères par caractères ; pour les autres, c'est le résultat de ces champs à qui on a appliqué la fonction `uppercase` du module serveur "Tool" qui sont comparés. Ainsi les deux chaines sont toutes deux en majsucule, ce qui nous élimine toute différence liée à la casse. La fonction `search_reference` va donc parcourir la liste des références et, suivant le mode de recherche, vérifier si la référence en cours est valide. Si oui, elle est écrite au format bibtex dans le fichier `bibtex_to_send` passé en paramètre et la variable `number_reference` permettant de compter le nombre de références valides est incrémentée de 1. Sinon, on passe à la suivante. Une fois la liste de références parcourue, la fonction retourne le nombre de références ajoutées. La fonction `service` du module serveur "Main" va lire ce nombre, et l'envoyer au client. Côté client, ce nombre est lu et si il vaut 0, la recherche n'a renvoyé aucun résultat, le client en est informé par un message et la fonction quitte sans autres effets. Sinon, le serveur envoie le fichier `bibtex_to_send` au client par la fonction `send_file_bibtex`. Le client sauve ce fichier dans le fichier `file_to_save`, reverrouille le bouton "Télécharger" et les deux fonctions quittent.

Si la requête 9 est envoyée au serveur, le déroulement sera identique, à la différence près que c'est la fonction `send_file_html` qui sera appellée pour envoyer le fichier `html_to_send` au client. Ce fichier sera créé en utilisant le module "Flex et Bison". On fait appel à `fork` pour créer un nouveau processus. Le processus fils va utiliser `open` pour ouvrir le fichier `bibtex_to_send` et sauvegarder le file descriptor associé. Puis, avec un appel à `dup2`, l'entrée standard (stdin) est remplacée par le précédent file descriptor. Cette opération est nécessaire pour que l'exécutable `bibtex_to_html` analyse le fichier `bibtex_to_send` et produise le fichier `html_to_send`. On utilise `execlp` pour faire appel à l'exécutable. Le processus père attend la fin du processus fils par un appel à `wait` et commence alors l'envoi au client du fichier `html_to_send`. Le client sauve ce fichier dans le fichier `file_to_save`, reverrouille le bouton "Télécharger" et les deux fonctions quittent.

3.4 Module Flex et Bison

3.4.1 Présentation

Ce module s'occupe de l'analyse d'un fichier de références bibtex et de la production associée d'un fichier de référence au format HTML. Le fichier flex est `bitexo_lexical.l`, le fichier bison est `bitexo_syntax.y`.

Comme son man l'indique, Flex est un analyseur lexical très répandu, il prend en entrée un stream et est capable de reconnaître des séquences de bits pour ensuite les remplacer par d'autres séquences de bits, définies par l'utilisateur à l'aide d'un patron. Utilisé dans la création de compilateurs et de traducteurs, nous l'utiliserons pour reconnaître les unités lexicales d'une liste de références au format bibtex.

Yacc est un utilitaire d'unix, bison un produit GNU (de même pour lex et flex), leur fonctionnements est à quelques fonctionnalités près semblable. YACC signifie "Yet Another Compiler Compiler" (ce qui n'est pas tout à fait exact en réalité, yacc/bison seul ne permet pas la création d'un compilateur). L'appellation Bison est un jeu de mot se basant sur l'homophonie yacc/yack. Dans le fichier Bison seront rentrées des règles de production, ce fichier nous produira un fichier C qui, une fois compilé, sera capable de faire l'analyse syntaxique d'une liste de références bibtex.
L'analyse syntaxique se fait en relation avec l'analyse lexicale ; l'analyse lexicale reconnaît des token appartenant au langage qu'elle transmet à l'analyse syntaxique qui détermine via les règles de production si la/les phrases sont correctes et appartiennent bien au langage. L'analyse se fera via une table d'analyse construite par Bison, qui utilisera tour à tour des décalages et réductions (shift et reduce).

La construction du module s'est révélée être plus simple que prévu. En effet, les fichiers analysés sont des fichiers produits par le serveur à partir de la liste chainée de références. Il nous est donc possible de prévoir la forme exacte qu'auront ces fichiers à analyser, en restreignant les libertés lexicales et syntaxiques, sans pour autant que cela constitue la moindre gêne pour l'utilisateur. Par exemple, les dispositions des virgules, sauts de lignes, types de références sont formatés d'une manière qui sera la même pour chaque fichier. Evidemment, si notre but avait été de produire un analyseur capable de traiter n'importe quel type de fichier bibtex valide, nous aurions du procéder différemment.

3.4.2 Token lexicaux

```
"@ARTICLE"  return ARTICLE;
"@BOOK"  return BOOK;
"@BOOKLET"  return BOOKLET;
"@CONFERENCE"  return CONFERENCE;
"@INBOOK"  return INBOOK;
"@INCOLLECTION"  return INCOLLECTION;
"@INPROCEEDINGS" return INPROCEEDINGS;
"@MANUAL"  return MANUAL;
"@MASTERTHESIS"  return MASTERTHESIS;
"@MISC"  return MISC;
"@PHDTHESIS"  return PHDTHESIS;
"@PROCEEDINGS"  return PROCEEDINGS;
"@TECHREPORT"  return TECHREPORT;
"@UNPUBLISHED"  return UNPUBLISHED;

"author = "  return AUTHOR;
"address = "  return ADDRESS;
"chapter = "  return CHAPTER;
"title = "  return TITLE;
"journal = "  return JOURNAL;
"volume = "  return VOLUME;
"pages = "  return PAGES;
"month = "  return MONTH;
"year = "  return YEAR;
"booktitle = "  return BOOKTITLE;
"publisher = "  return PUBLISHER;
"institution = "  return INSTITUTION;
"edition = "  return EDITION;
"editor = "  return EDITOR;
"howpublished = "  return HOWPUBLISHED;
```

```
"note = "  return NOTE;
"school = "  return SCHOOL;
"number = "  return NUMBER;
"organization = "  return ORGANIZATION;
"series = "  return SERIES;
"type = "  return TYPE;

"\n}\n"  return END_REFERENCE;
",\n"  return END_FIELD;

"\n"


"{"[^}]+",\n"  {key_array[key_l] = malloc(sizeof(char)* strlen(yytext) +1);
strcpy(key_array[key_l], yytext);
key_l ++;
return KEY;}


 "{".*"}"  {text_array[text_l] = malloc(sizeof(char)* strlen(yytext) +1);
strcpy(text_array[text_l], yytext);
text_l ++;
return TEXT;}
```

Les deux tableaux `text_array` et `key_array` seront déclarés en externe dans le fichier d'analyse syntaxique et nous permettront de récupérer les textes des champs et des clefs.

3.4.3 Régles syntaxiques

```
P :
                          Reference_List {Règle P 1}

Reference_List :
                          Reference_List Reference {Règle Refrence_List 2}
                          Reference {Règle Refrence_List 3}

Reference :
                          Reference_Type KEY Field_List END_REFERENCE  {Règle Refrence 4}
                          Reference_Type KEY END_REFERENCE {Règle Refrence 5}

Field_List :
                          Field_List Field {Règle Field_List 6}
                          Field  {Règle Field_List 7}

Field :
                          Field_Type TEXT END_FIELD  {Règle Field 8}
                          Field_Type TEXT {Règle Field 9}


Field_Type :
                          AUTHOR {Règle Field_Type 10}
                          ADDRESS {Règle Field_Type 11}
                          CHAPTER {Règle Field_Type 12}
```

```
                        TITLE {Règle Field_Type 13}
                        JOURNAL {Règle Field_Type 14}
                        VOLUME {Règle Field_Type 15}
                        PAGES {Règle Field_Type 16}
                        MONTH {Règle Field_Type 17}
                        YEAR {Règle Field_Type 18}
                        BOOKTITLE {Règle Field_Type 19}
                        PUBLISHER {Règle Field_Type 20}
                        INSTITUTION {Règle Field_Type 21}
                        EDITION {Règle Field_Type 22}
                        EDITOR {Règle Field_Type 23}
                        HOWPUBLISHED {Règle Field_Type 24}
                        NOTE {Règle Field_Type 25}
                        SCHOOL {Règle Field_Type 26}
                        NUMBER {Règle Field_Type 27}
                        ORGANIZATION {Règle Field_Type 28}
                        SERIES {Règle Field_Type 29}
                        TYPE {Règle Field_Type 30}

Reference_Type :
                        ARTICLE {Règle reference_Type 31}
                        BOOK {Règle reference_Type 32}
                        BOOKLET {Règle reference_Type 33}
                        CONFERENCE {Règle reference_Type 34}
                        INBOOK {Règle reference_Type 35}
                        INCOLLECTION {Règle reference_Type 36}
                        INPROCEEDINGS {Règle reference_Type 37}
                        MANUAL {Règle reference_Type 38}
                        MASTERTHESIS {Règle reference_Type 39}
                        MISC {Règle reference_Type 40}
                        PHDTHESIS {Règle reference_Type 41}
                        PROCEEDINGS {Règle reference_Type 42}
                        TECHREPORT {Règle reference_Type 43}
                        UNPUBLISHED {Règle reference_Type 44}
```

On notera également la présence de fonctions de traitement, permettant de retirer certains caractères tels que '', '' ou ',' des extrémités des champs des références et ce afin de garantir l'obtention d'un fichier HTML lisible.

3.4.4 HTML valide

Les standards de validité du HTML sont des normes décidées par le World Wide Web Consortium (W3C). Ces normes sont extrêmement importantes. Si un code HTML n'est pas valide, ce sera le rôle du navigateur de le réparer. Et chaque navigateur réparera le fichier à sa manière, entrainant le risque que, d'un navigateur à l'autre, la page s'affiche différemment. Produire un code HTML valide en accélérera l'affichage et permettra un meilleur référencement par les robots des moteurs de recherche (ne buttant pas sur des portions de codes inconnues). Enfin, il facilite l'accessibilité au Web des personnes mal voyantes en améliorant la lecture des pages.

Il est possible de tester la validité HTML et CSS d'une page web en se rendant sur les pages suivantes, proposant des validateurs de code en ligne : `http://validator.w3.org/` et `http://jigsaw.w3.org/css-validator/`.

Les fichiers, que nous créons, se soumettent avec succès à ces deux tests, nous pouvons donc leur ajouter les deux logos rendant compte de cet état de fait !

4 Manuel

Le manuel d'installation et d'utilisation a été rédigé à part dans le fichier `manuel.pdf` qui vous a été remis. N'hésitez pas à le consulter !

5 Planning détaillé

Le travail préliminaire réalisé lors du précédent rendu a été d'une grande aide dans la réalisation du logiciel et a ainsi permis d'entrer dans la réalisation du code avec rapidité et efficacité. Cela démontre encore une fois toute l'importance du travail préliminaire de réflexion et d'analyse nécessaire à la conduite de tout projet. Le planning prévisionnel a été le suivant :

- Découverte de GTK, tests, petits programmes, module client "A propos" : deux jours
- modules client et serveur "Main" et "Connexion" : deux jours
- modules client et serveur "Ajouter référence" : trois jours
- modules client et serveur "Modifier référence" : un jour
- modules client et serveur "Supprimer référence" : un jour
- module "Flex et Bison" : un jour
- modules client et serveur "Télécharger Références" et "Send Référence" : un jour
- Phase de tests générale, phase de test à l'Institut Galilée, vérification des commentaires et des tests des appels systèmes : trois jours
- Rédaction du manuel : deux jours
- Rédaction du rapport de développement : trois jours

Ce planning prévisionnel a été suivi avec succès !

6 Validation

Les tests suivants sont issus du rapport préléminaire.

Sur le Client :

- Création de la socket, liaison, le client parvient à se connecter au serveur : OK
- Messages d'erreurs alertant le client en cas de serveur introuvable ou déconnexion du serveur : OK
- Toutes les requêtes sont disponibles, le serveur répond de manière adéquate aux requêtes du client et propose le service associé : OK
- Modification de référence : OK
- Suppression d'une référence : OK
- Ajout d'une référence : OK
- Chargement d'un fichier au format BibTeX : OK
- Téléchargement d'une liste de références au format BibTeX : OK
- Téléchargement d'une liste de références au format html : OK
- Lors de l'ajout d'une référence, via l'interface graphique, celle-ci doit respecter les champs obligatoires suivant le type de fichier ; si des champs sont restés vides dans la fenêtre d'ajout, la validation ne pourra se faire : OK
- La demande de modification d'une référence inexistante renvoie un message d'erreur approprié : OK
- La demande de suppression d'une référence inexistante renvoie un message d'erreur approprié OK
- L'ajout d'une référence échoue si la clef est redondante : OK

Sur le Serveur :

- Création de la socket, liaison, mise en écoute, création de la socket de service pour gérer le client : OK
- Message d'erreur si la connexion avec le client est perdue, fermeture de la socket, terminaison du processus, libération des sémaphores, destruction des fichiers créés et non téléchargés : OUI et NON, on a manqué de temps pour l'ajout des sémaphores et la destruction des fichiers créés
- Le serveur réagit de la manière attendue aux sollicitations du client (demande de service) - voir liste ci-dessus : OK
- Le serveur peut stocker différents fichiers de références envoyés par différents clients et ajouter les références à une même liste chainée : OK
- Les modifications apportées par le client sont intégrées correctement : OK
- En cas de demande de modification, le serveur trouve la référence avec la clef associée, sinon un message d'erreur est envoyé : OK
- La suppression d'une référence est effective si la clef mentionnée est la bonne, sinon un message d'erreur est envoyé : OK
- A la déconnexion du client, tous les fichiers temporaires créés par ce client sont détruits (création d'une liste de références sur critères, fichiers au format HTML etc). Seuls les fichiers au format BibTeX ayant été chargés doivent rester. Tous les sémaphores associés sont libérés : NON

Pour l'analyse lexicale et syntaxique :

- L'analyse lexicale se déroule correctement, sortie d'un document contenant les différents token lexicaux : OUI et NON une modification de flex nous a empêchés de tester l'analyse lexicale seule, les tests n'ont pu être réalisés que sur l'analyse lexicale et syntaxique conjointe
- Les token lexicaux sont tous et correctement identifiés : OK
- Un document lexicalement non valide génère un message d'erreur : OK (mais dans le cadre du logiciel, les cas d'erreurs sont normalement impossibles)
- Les règles de grammaire sont correctement appliquées : OK
- Un document syntaxiquement non valide génère un message d'erreur : OK (même remarque que précédemment)
- La production de code HTML se déroule conformément aux attentes, le fichier obtenu est clair, lisible et valide : OK
- La production de code HTML à partir d'un document erroné (erreurs lexicales, erreurs syntaxiques) échoue : OK (même remarque que précédemment)

L'installation du client multiplateforme fonctionne : NON

Même si nous n'avons pu satisfaire toutes les fonctionnalités, un nombre substantiel à été porté à terme, ce qui est plutôt positif.

7 Fonctionnalités non réalisées et problèmes restant

Nous avons malheureusement manqué de temps pour réaliser toutes les fonctionnalités initialement prévues.

Nous avions eu l'intention de proposer une version Windows du client mais il s'est avéré que nos connaissances de la programmation en C sous windows étaient trop restreintes pour avoir le temps de développer cette version dans les temps impartis. Même si l'on a une légère préférence pour Linux plutôt que Windows, il reste néamoins un OS extrêmement utilisé de part le monde et approfondir nos connaissances dessus ne pourrait être que bénéfique.

L'implantation des sémaphores sur le serveur n'a pas non pu être réalisée. C'est vraiment dommage tant le problème s'y prétait particulièrement bien. Notre ressource critique aurait été la liste chainée de références qu'il aurait fallu protéger de plusieurs lecteurs/écrivains (les threads pouvant endosser ces fonctions à tour de rôle). L'écriture aurait bloqué toute autre écriture ou lecture tandis que des lectures concurrentes auraient pu avoir lieu.

Nous n'avons pas nettoyé les fichiers temporaires créés sur le serveur à la déconnexion d'un client. Cela aurait été pourtant rapide avec `unlink`, les noms des fichiers étant uniques pour chaque client, mais ces fichiers sont généralement réutilisés par les clients suivant, aussi on a préféré s'occuper de fonctionnalités plus importantes, celle-ci s'étant retrouvée n'être pas vraiment primordiale.

A notre grand regret subsistent encore quelques erreurs dans le programme.

Dans les fenêtres de sélection de fichier, des boutons raccourcis sont présents de manière automatique. Si il est possible de retirer les boutons permettant la création ou suppression de fichiers, il n'est pas possible de supprimer ces raccourcis. Hors, sur Ubuntu, un des raccourcis proposé est le dossier "Desktop" qui correspond en réalité au dossier "Bureau". Si l'on utilise ce bouton de raccourci pour sélectionner un fichier, tout appel à `fopen` plantera irrémédiablement, le dossier "Desktop" n'existant pas. Peut être est-ce dû à la version française de Ubuntu ? Ou au fait que Debian sur lequel est basé Ubuntu possède lui un dossier "Desktop" ?

Nous avions mis un point d'honneur à utiliser au maximum l'allocation dynamique pour tirer le meilleur parti possible de la gestion de la mémoire offerte par le C, en utilisant des appels à `malloc`, `calloc` et `realloc`. Malheureusement, pendant la phase finale de tests, de nombreux appels à `realloc` faisaient planter le logiciel sans qu'il ne s'agisse nécessairement des mêmes à chaque fois, même en cas d'utilisation similaire du logiciel et ce malgré que tous ces appels soient testés. Des recherches et de nombreux tests ont été effectués pour comprendre la nature du problème sans que des solutions ne puissent être trouvées (au mieux avons nous appris que de multiples appels à realloc n'étaient pas optimisés, mais cela ne semblait pas avoir de rapport direct avec notre souci). Nous avons donc été contraints de remplacer ces appels à `realloc` par des appels à `calloc`. Cette solution de secours, qui n'a été choisie que pour pouvoir présenter un résultat final fonctionnel n'est évidemment pas satisfaisante et nous déçoit énormément car elle ne correspond pas du tout à notre volonté initiale.

Présenter ses échecs est une étape tout aussi importante que présenter ses succès. Ils doivent être acceptés et exploités pour nous permettre de nous fixer de nouveaux objectifs et de continuellement se perfectionner et progresser.

8 Fichier de test : exemple.bib

```
@UNPUBLISHED{PKGT:1,
author = {P. Kenneth and G. E. Taylor},
title = {Solution of variational problems with bounded control variables by means of the genera
note = {presented at the Symp.\ on Recent Advances in Optimization Techniques, Carnegie Inst.\
month = {April},
year = {1965}
}

@TECHREPORT{user-behavior-tr,
author = {Konstantin Pussep and Sebastian Kaune and Christof Leng and Aleksandra Kovacevic and
title = {Impact of User Behavior Modeling on Evaluation of Peer-to-Peer Systems},
institution = {Technische Universit{\"a}t Darmstadt},
year = {2008},
number = {KOM-TR-2008-07},
month = {November}
}

@PROCEEDINGS{YAN-83,
title = {Proc. Fifteenth Annual},
year = {1983},
editor = {Wizard V. Oz and Mihalis Yannakakis},
publisher = {Academic Press},
organization = {ACM},
address = {Boston},
month = {mar}
}

@PHDTHESIS{Knuth63,
author = {Knuth, Donald E.},
title = {Finite semifields and projective planes},
school = {California Institute of Technology},
year = {1963}
}

@MISC{cowan1991label,
author = {Cowan, R.G. and Donovan, R.J. and McKillip, B.G. and others},
title = {Label assembly with removable booklet},
howpublished = {Google Patents},
month = {feb # "~12"},
year = {1991},
note = {US Patent 4,991,878}
}

@MASTERTHESIS{e1-207a-f00,
author = {Max Rydahl Andersen and Claus Nyhus Christensen and Kristian Lykkegaard Sorensen},
title = {Internal documentation in an Elucidative environment},
school = {Aalborg University},
year = {2000},
month = {June},
note = {Available from http://dopu.cs.auc.dk}
}
```

```
@MANUAL{MAN-86,
title = {The Definitive Computer Manual},
author = {Larry Manmaker},
organization = {Chips-R-Us},
address = {Silicon Valley},
edition = {Silver},
month = {apr-may},
year = {1986}
}

@INPROCEEDINGS{pati930MP,
author = {Y. Pati and R. Rezaiifar and P. Krishnaprasad},
title = {Orthogonal Matching Pursuit: Recursive Function Approximaton with Applications },
booktitle = {27 th Annual Asilomar Conference on Signals, Systems, and Computers},
year = {93},
pages = {40-44},
publisher = {IEEE}
}

@INCOLLECTION{LIN-77,
author = {Daniel D. Lincoll},
title = {Semigroups of Recurrences},
booktitle = {High Speed Computer and Algorithm Organization},
year = {1977},
editor = {David J. Lipcoll and D. H. Lawrie and A. H. Sameh},
pages = {179--183},
publisher = {Academic Press},
address = {New York},
month = {sep}
}

@INBOOK{Kristensen83,
author = {B. B. Kristensen and Ole L. Madsen and B. Mouller-Pedersen and K. Nygaard},
title = {Integrated Interactive Computing Systems},
chapter = {Syntax-directed program modularization},
publisher = {North-Holland, Amsterdam},
year = {1983},
editor = {P. Degano and E. Sandewall},
pages = {207-219}
}

@CONFERENCE{Duessman:09,
author = {Duessmann, H. and Wenus, J. and Kalamatianos, D. and Paul, P. and Wellstead, P. and H
title = {Real-Time automated live cell image acquisition and analysis using confocal laser scan
chapter = {12-2},
publisher = {Neuroscience 2009},
year = {2009}
}

@BOOKLET{JK-TPCA,
title = {The Programming of Computer Art},
author = {Jill C. Knvth},
howpublished = {Vernier Art Center},
address = {Stanford, California},
month = {feb},
```

```
year = {1988}
}

@BOOK{Mallatbook,
author = {S. Mallat},
title = {A Wavelet Tour of Signal Processing},
publisher = {Academic Press},
year = {1998},
address = {Boston}
}

@ARTICLE{Temlyakov,
author = {V. Temlyakov},
title = {Nonlinear methods of approximation},
journal = {FOCM},
year = {2008},
volume = {3},
number = {1},
pages = {33-107},
month = {Feb.}
}
```

9 Bibliographie

De nombreux sites nous ont servi de support au cours de la réalisation de ce projet :

- cours de GTK sur le site developpez.com : http ://gtk.developpez.com/cours/gtk2/
- article sur les listes simplement chainées sur le site developpez.com : http ://nicolasj.developpez.com/articles/listesimple/
- article sur les combos box sur le site dcobac.free.de : http ://dcobac.free.fr/spip/spip.php ?article48
- la documentaion gnome sur GTK : http ://library.gnome.org/devel/gtk/unstable/GtkWidget.html#GtkWidget.synopsis
- un tutoriel du site du zero sur les logiciels libres : http ://www.siteduzero.com/tutoriel-3-37075-faire-de-son-programme-un-logiciel-libre.html
- un article du site du site logram project sur la commande getopt http ://www.logram-project.org/news-2-10-1-getopt-et-le-parsage-de-la-ligne-de-commande-en-c.html
- de manière générale, tous les manuels de nos fonctions !