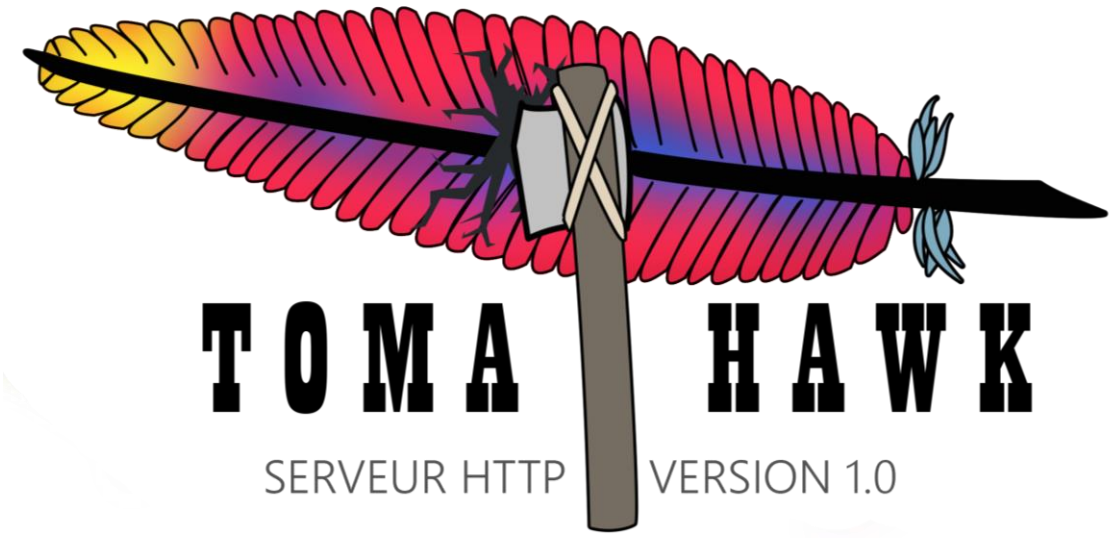
Romain VIDAL
Thomas JACUMIN
Angelo LORBER

3A - IR
GROUPE 11

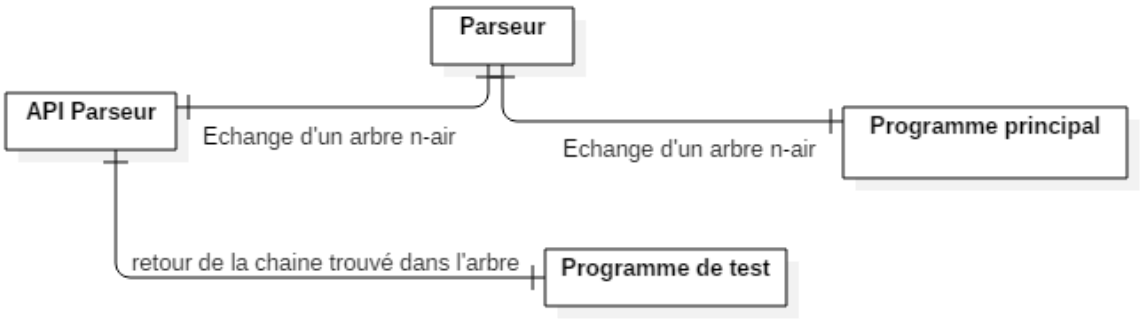
PROJET NE 302 – SERVEUR HTTP

PARTIE 2 : PARSEUR HTTP



1. ÉLABORATION DE L'ARCHITECTURE

1.1. ARCHITECTURE GLOBALE

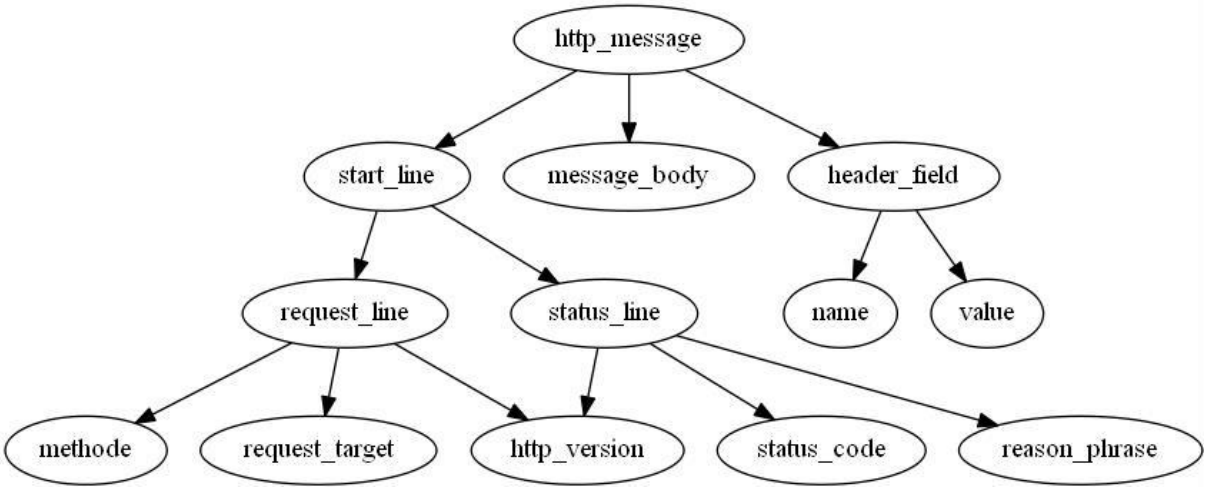


Notre programme principal utilisera un arbre qu'il parcourra pour y étudier les différents champs. Le parseur http étudiera donc une requête passé en paramètre par le programme principal et retournera une structure représentant la requête sous la forme d'un arbre.

L'API échangera de la même manière avec le parseur puis parcourra l'arbre en comparant chaque valeur avec la chaine demandée pour la retourner.

1.2. LE PARSEUR HTTP

Notre parseur va donc prendre en entrée une requête http et la « découper » selon le schéma ci-dessous



Il utilisera des automates pour déterminer l'exactitude de la requête (au niveau syntaxique seulement). Les automates que nous avons utilisés pour cela sont disponibles en annexe I.

2. MISE EN PLACE DU PROJET

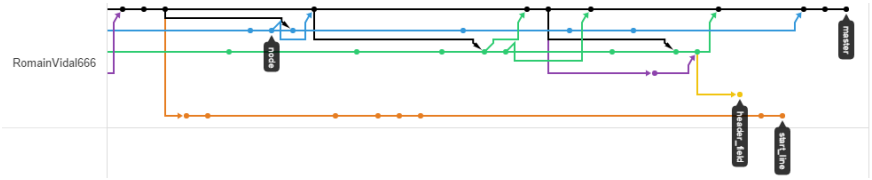
2.1. MISE EN PLACE D'UN GIT

Pour la gestion des versions nous avons choisi d'utiliser un git, hébergé sur Git Hub.

Il est disponible à l'adresse suivante :

https://github.com/RomainVidal666/Tomahawk.git

Nous avons eu quelque difficulté avec GitHub pour fusionner les différentes branches de notre projet comme le montre l'image ci-dessous :



2.2. REPARTITION DE LA CHARGE DE TRAVAIL

Pour la répartition du travail nous nous sommes retrouvées pour travailler en groupe, notamment pendant la phase de conception, il était indispensable pour nous de pouvoir échanger nos idées.

Pour le codage du parseur nous avons procédé de la même façon, chacun s'occupant d'une fonction. Cela nous a permis d'avancer efficacement et de ne pas rester bloqué sur des problèmes simples.

2.3. GESTION DE LA DOCUMENTATION

Pour gérer la documentation du code du projet, nous avons choisi d'utiliser Doxygen que nous avions déjà utilisé et qui s'est révélé très efficace.

Il permettra d'avoir une documentation du code correct et de la tenir à jour facilement.

3. IMPLEMENTATION

3.1. IMPLEMENTATIONS DES AUTOMATES

L'automate parcourt la chaine à l'aide d'un pointeur sur celle-ci, qu'il fait avancer à chaque lecture. Quand il se trouve sur un état accepteur la sous-chaine est sauvegardée dans l'arbre.

Nous avons écrit une fonction par automates, pour chacune de ces fonctions le comportement est le suivant :

- L'arriver sur un état accepteur termine la fonction.
- Une transition non reconnu termine la fonction.
- Une structure est créée à l'entrée de la fonction et est ajouté à l'arbre.
- Si l'état représente un sous-automate, on appelle la fonction correspondante.

3.2. IMPLEMENTATION DE L'ARBRE

L'arbre est défini sous la forme d'une liste chainée.

On a défini une structure avec comme attributs :

- Le nom
- un pointeur sur le début de la sous-chaine.
- Un pointeur sur la fin de la sous-chaine.
- Une liste pour stocker un pointeur vers les éléments suivants.

On utilise les fonctions standards de parcourt, d'ajout et de suppression dans un arbre.

3.3. IMPLEMENTATION DE L'API

L'API est assez simple il s'agit de passer la requête au parseur afin récupérer l'arbre puis de le parcourir afin de trouver le champ demandé, extraire la sous chaine et enfin, la renvoyer.

Le main ainsi que la fonction de callback se trouve dans le fichier main.c. Ce fichier doit inclure le fichier « api.h » qui se trouve dans le même répertoire.

```
#include "api.h"
```

La fonction de callback doit suivre le prototype suivant :

```
void callback (char *found, unsigned int len) ;
```

Et enfin le prototype du parseur et le suivant :

```
int parser (char *buf, unsigned int len, char *search, void (*callback) ()) ;
```

4. TESTS

4.1. TESTS FOURNIS

Nous avons effectué des tests sur le jeu de test fournit les requêtes suivantes ont passé le test avec succès :

- get1 à get 6.
- get11 et get12.

Les requêtes suivantes ne sont pas reconnues :

- get7 à get9 : FAUX car on trouve un SP après le field-content en fin de ligne.
- get10 : FAUX car version A.1

4.2. TESTS SUPLEMENTAIRES

Nous avons ensuite testé notre parseur avec d'autre requête, voici les résultats :

```
Le test est OK pour les requêtes suivantes :
```

- ```
  POST /fichier.ext HTTP/1.1\n\rHost: www.site.com\n\rConnection: Close\n\rContent-type:
  application/x-www-form-urlencoded\n\rContent-Length:
  33\n\r\n\rvariable=valeur&variable2=valeur2
  ```

- ```
  HTTP/1.1 200 OK\n\rDate: Thu, 11 Jan 2007 14:00:36 GMT\n\rServer: Apache/2.0.54
  (Debian GNU/Linux) DAV/2 SVN/1.1.4\n\rConnection: close\n\rTransfer-Encoding:
  chunked\n\rContent-Type: text/html; charset=ISO-8859-1\n\r\n\r
  ```

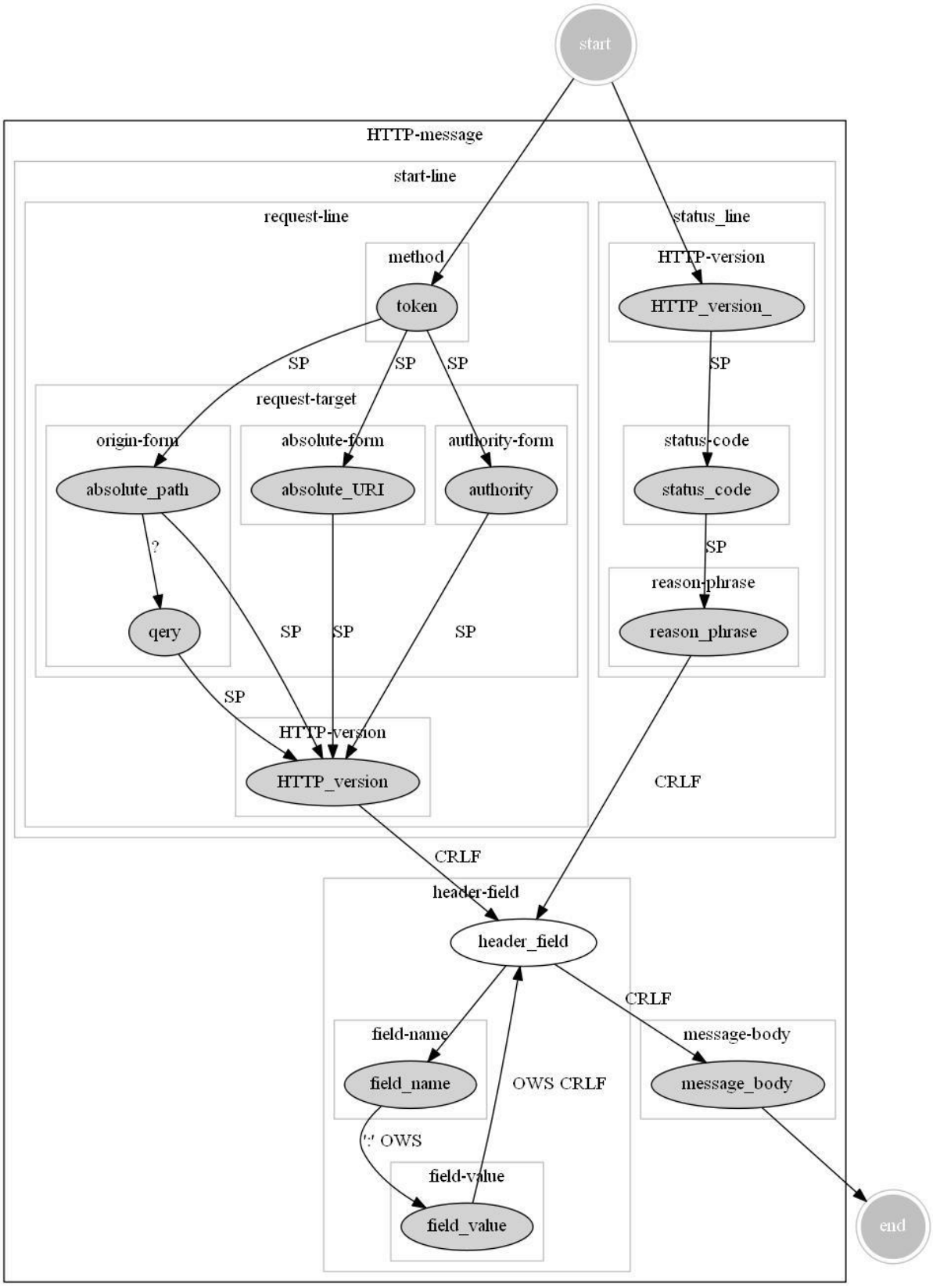
-

Le test est en revanche KO pour les requêtes suivantes (à cause de l'espace entre le 'field-name' et ':') :

- GET http://www.commentcamarche.net HTTP/1.0\n\rAccept : text/html\n\rIf-Modified-Since : Saturday, 15-January-2000 14:37:11 GMT\n\rUser-Agent : Mozilla/4.0 (compatible; MSIE 5.0; Windows 95)\n\r\n\r

ANNEXE

I. AUTOMATE



II. AUTOMATE DETAILLE