CastDoc

Danis Marius-Paul

Facultatea de Informatica, Iasi, Romania

1 Introducere

Pentru aplicatia **CastDoc** se doreste implementarea unui server si a unui client, ce va transforma documente dintr-un tip dat in altul. Clientul va transmite tipul documentului initial si tipul documentului final, iar in cazul in care aceasta conversie este posibila, se va transmite documentul initial si se va primi documentul rezultat. In ceea ce priveste serverul, se va utiliza un fisier de configurare in care se vor specifica transformarile suportate, precum si utilitarele (functiile) care realizeaza aceste transformari. Mai mult, serverul va oferi, de asemenea, functionalitatea de caching (verificarea numelui fisierului si a hash-ului continut). Pe baza acestei functionalitati, serverul va putea fie sa realizeze conversia pe baza a ce este retinut in cache, fie sa adauge in cache numele fisierului ce se doreste a fi convertit si hash-ul acestuia.

2 Tehnologii utilizate

In ceea ce priveste tehnologiile utilizate in realizarea aplicatiei Cast Doc, mentionam folosirea protocolului TCP[1] (transmiterea de mesaje intre server si clienti). Utilizarea acestuia este necesara pentru realizarea adecvata a conversiilor de fisiere, fiind un protocol orientat conexiune. Este preferata folosirea sa in locul protocolului UDP intrucat nu este atat de importanta viteza de transmitere a fisierelor. Mai mult decat atat, utilizarea protocolului TCP asigura in primul rand integritatea fisierelor transmise.

3 Arhitectura aplicatiei

3.1 Structura programului

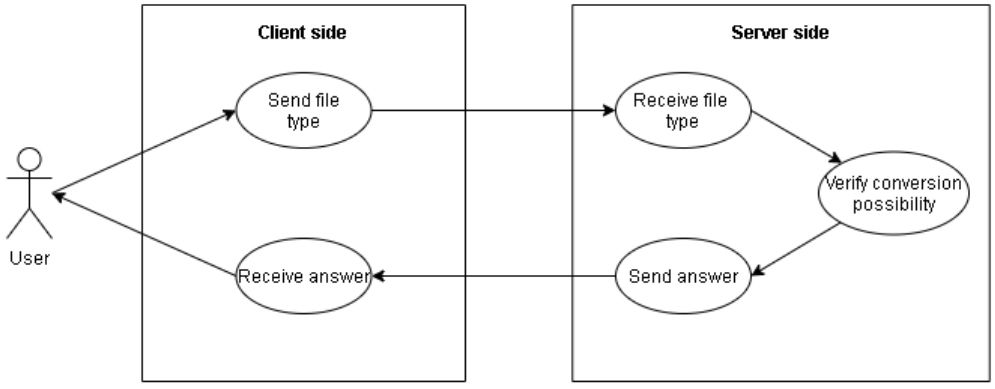

Fig. 1. Clientul trimite tipul fisierului catre server si primeste un raspuns daca este suportata conversia

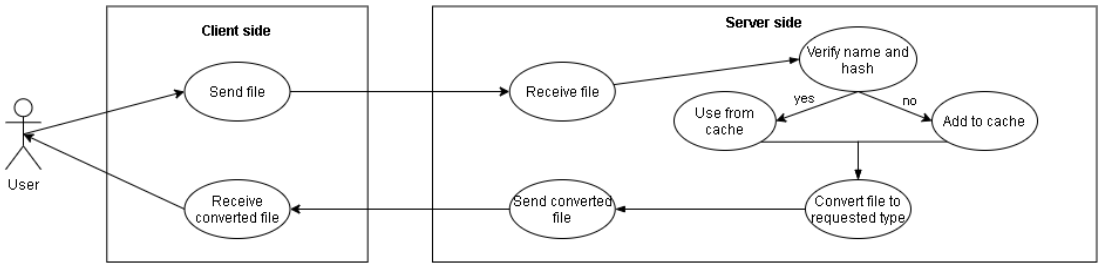

Fig. 2. Clientul trimite fisierul catre server dupa ce acesta primeste confirmare pentru conversie,verifica cache-ul, realizeaza conversia si trimite fisierul convertit catre client

Clientul va trimite intai formatul dorit pentru conversie, iar dupa primirea unui mesaj de confirmare din partea serverului, va transmite catre server fisierul pe care doreste sa il converteasca. La final, dupa ce conversia a fost realizata, clientul va primi de la server fisierul nou convertit.
Serverul va primi intai formatul dorit pentru conversie, va verifica daca acea conversie este posibila, rezultat ce va fi transmis inapoi clientului printr-un mesaj ce va cere fie alegerea unei alte conversii, fie trimiterea fisierului. Dupa primirea fisierului, prin functionalitatea de caching, serverul va verifica numele fisierului si a hash ului continut. Daca acestea se regasesc in baza de date, le va prelua de acolo si va realiza conversia, iar daca nu se regasesc, le va adauga in baza de date si va realiza conversia. La final, acesta va trimite inapoi catre client fisierul convertit.

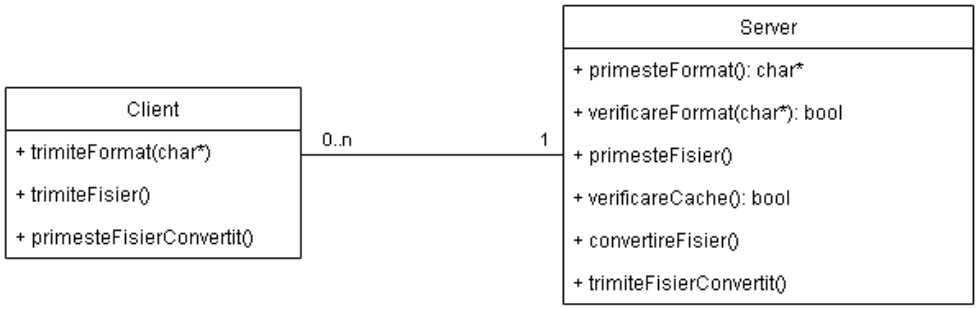

Fig. 3. Diagrama de clasa UML cu modulele principale, Client si Server

3.2 Configurare si Cache

Pentru stocarea conversiilor posibile (tipul de intrare, tipul de iesire si comanda executata pentru conversie), precum si pentru realizarea functionalitatii de caching(stocarea numelui si a hash-ului) vom utiliza fisiere de `JSON` (pentru fisierele *config.json* si *cache.json*)

4 Detalii de implementare

Clientul va trimite intai formatul initial al fisierului de intrare, imediat dupa care va trimite formatul dorit pentru fisierul de iesire.

```
/* citirea mesajului */
    bzero (msg1, 100);
    printf ("[client]Introduceti tipul fisierului de intrare: ");
    fflush (stdout);
    read (0, msg1, 100);
    msg1[strlen(msg1)-1] = 0;
    msglen1 = strlen(msg1);

    bzero(msg2,100);
    printf ("[client]Introduceti tipul fisierului de iesire: ");
    fflush (stdout);
    read (0,msg2,100);
    msg2[strlen(msg2)-1] = 0;
    msglen2 = strlen(msg2);
```

Serverul va primi apoi formatul initial si cel final, dupa care le va verifica cu cele existente in fisierul *config.json* si va determina daca transformarea dorita este posibila. Aceasta pas se va realiza prin intermediul functiei *conversionsResponse*, in care se va realiza de asemenea primirea fisierului, convertirea acestuia si trimiterea fisierului convertit inapoi catre client:

```
   printf("[server]Formatul de intrare receptionat este: %s\n", msg1);

        printf("[server]Formatul de iesire receptionat este: %s\n", msg2);

        string format_in = msg1;
        string format_out = msg2;

        printf("%s,%s\n",format_in.c_str(),format_out.c_str());
        for(int i=0; i<config.size();i++)
        {
            if(config[i]["tip_intrare"] == format_in && config[i]["tip_iesire"] == format_out)
            {
                conversion = true;
                index_conversie = i;
                break;
            }
        }
     bzero(msgrasp, 100);
        if (conversion) {
            strcpy(msgrasp, "DA");
            ok = 0;
        } else {
            strcpy(msgrasp, "NU");
        }
```

Daca verificarea a fost realizata cu succes, clientul poate sa trimita fisierul catre server prin functia *fileSender* ce va primi ca parametrii file-descriptorul serverului si path-ul fisierului ce urmeaza a fi trimis spre conversie. Serverul va primi acest fisier prin intermediul functiei *fileReceiver* ce va primi ca parametru file-descriptorul clientului si path-ul fisierului.

```
int fileSender(int serverfd, char* fpath)
{
    int filedescriptor;
    size_t bytes;
```

```
    char buffer[1025];
    size_t fsize;
    struct stat fstat;

    if (stat(fpath, &fstat) < 0)
    {
        fprintf(stderr, "Error at fstat, cause is: %s", strerror(errno));
        exit(EXIT_FAILURE);
    }

    fsize = (size_t)fstat.st_size;
    if(fileWrite(serverfd, &fsize, sizeof(fsize)) == -1)
        return -1;

    if((filedescriptor = open(fpath, O_RDONLY)) == -1)
        return -1;

    ssize_t unsent = fsize;
    while (((bytes = read(filedescriptor,buffer,1025)) > 0) && (unsent > 0))
    {
        if(fileWrite(serverfd,buffer,bytes) == -1)
            return -1;
        unsent = unsent - bytes;
    }
    close(filedescriptor);
    return 0;
}

int fileReceiver(int clientfd, char* fpath)
{
    char* buffer = (char*)malloc(1025);
    size_t fsize;

    void* pointer = &fsize;
    if (fileRead(clientfd, pointer) == -1)
        return -1;
    int filedescriptor;
    if ((filedescriptor = open(fpath, O_RDWR | O_CREAT | O_TRUNC, 0600)) == -1)
        return -1;

    ssize_t unsent = fsize;
    ssize_t length;

    pointer = buffer;
    while ((unsent > 0) and ((length = fileRead(clientfd,pointer)) >= 0)){
        buffer = (char*) pointer;
        write(filedescriptor, buffer, length);
        unsent = unsent - length;
        pointer = buffer;
    }
    if(length < 0){
        free(buffer);
        return -1;
    }
    free(buffer);
    close(filedescriptor);
    return 0;
```

```
}
```

La nivelul serverului, acesta va realiza operatia de conversie a fisierului primit de la client prin intermediul functiei *Conversion* ce va primi ca si parametrii comanda de executie specifica utilitarului folosit pentru conversie, dimensiunea comenzii, numele fisierului de intrare si numele fisierului de iesire:

```
void Conversion(char* command, size_t command_len, char* fname_in, char* fname_out) {

    char command_user[100];
    strcpy(command_user,command);

    //command_user[command_len - 1] = 0;
    printf("%s\n", command_user);

    char *cmd_argv[15];
    int cmd_argc = 0;
    for (int i = 0; i < 15; i++)
        cmd_argv[i] = static_cast<char *>(malloc(50 * sizeof(char)));
    char *cmd_arg = strtok(command_user, " ");
    while (cmd_arg != NULL) {
        strcpy(cmd_argv[cmd_argc], cmd_arg);
        if(strcmp(cmd_argv[cmd_argc],"in") == 0)strcpy(cmd_argv[cmd_argc], fname_in);if(strcmp(cmd_argv[cmd_argc],"
        {
            strcpy(cmd_argv[cmd_argc],fname_out);
        }
        cmd_argc++;
        cmd_arg = strtok(NULL, " ");
    }
    cmd_argv[cmd_argc] = cmd_arg;

    execvp(cmd_argv[0], cmd_argv);
    exit(0);
}
```

In ceea ce priveste functionalitatea de caching, determinarea hash-ului se va realiza prin intermediul functiei *getFileHashCode*:

```
int getFileHashCode(char* fname,unsigned char* out_pointer)
{
    unsigned char buffer[8192];
    FILE *fd;
    SHA_CTX sha_c;
    int error;
    fd = fopen(fname, "rb");
    if (fd == NULL) {
        return -1;
    }
    SHA1_Init(&sha_c);
    for (;;) {
        size_t length;

        length = fread(buffer, 1, sizeof buffer, fd);
        if (length == 0)
            break;
        SHA1_Update(&sha_c, buffer, length);
    }
    error = ferror(fd);
    fclose(fd);
```

```
22      if (error) {
23          /* some I/O error was encountered; report the error */
24          return -1;
25      }
26      SHA1_Final(out_pointer, &sha_c);
27      return 0;
28  }
```

5 Concluzii

Dupa cum se poate vedea mai sus, programul nu este, in acest stadiu, unul complex. El mai poate fi imbunatatit astfel:

- prin adaugarea unei interfete grafice la nivelul clientului pentru o utilizare mai intuitiva si placuta (utilizatorul nu trebuie sa tasteze comenzile din terminal, ci poate urma pasii necesari din cadrul interfetei);
- prin implementarea unei functionalitati de caching si a unui comportament blocant in cadrul utilizarii cache-ului.
- prin implementarea optiunii de a converti mai multe fisiere in acelasi timp pentru un format de convertire

References

1. Resurse referitoare la protocolul TCP si exemple de utilizare – `https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php`
2. Resurse referitoare la nlohmann/json - JSON for Modern C++ – `https://github.com/nlohmann/json`
3. Resurse referitoare la *hash* – `https://en.wikipedia.org/wiki/Hash_table`