Academia Tehnică Militară

Document de descriere al design-ului software

Web Crawler

Realizatori:
Ghenea Claudiu
Hariga George
Florea Vlad
Neacșu Gabriel
Panțucu Flavius

BUCUREŞTI
2020

Cuprins

1. Scopul documentului

Acest document este conceput pentru a descrie procesul de dezvoltare a aplicației de tip Web Crawler. Documentul în cauză servește drept ghid practic în dezvoltarea soluției software.

2. Conținutul documentului

Documentul este structurat în trei secțiuni esențiale.

2.1. Modelul datelor

În cadrul acestui capitol sunt descrise principalele componente folosite, formatul fișierelor și argumentelor acceptate, precum și câteva exemple de utilizare ale aplicației.

2.2. Modelul arhitectural

Acest capitol este dedicat prezentării detaliate a claselor ce stau la baza aplicației dezvoltate.

Relația dintre clase este ilustrată prin intermediul unei diagrame UML.

2.3. Elementele de testare

Capitolul conține detalierea procesului de testare prin intermediul căruia se va verifica funcționalitatea întregului sistem.

3. Modelul datelor

3.1. Structuri de date globale

Elementele globale ale aplicației sunt reprezentate de clasele **Logger**, **Configurations** și **CrawlerThreadPool**, aceste clase fiind cele ce conțin date sau funcționalități necesare întregului program.

Logger-ul este clasa ce frunizează datele ce sunt necesare a fi observate de utilizator precum: mesajele de eroare, istoricul log-urilor de către utilitar, sitemap-ul furnizat pe baza activității de descărcare a utilitarului.

Clasa **Configuration** este primul pas în activitatea utilitarului, indiferent de funcționalitatea lansată de utilizator. Aceasta are rolul de a seta, fie pe baza unor date prestabilite, fie pe baza unui fișier de configurare furnizat de utilizator, componentele necesare activității utilitarului. În cadrul acestei clase, sunt setați parametrii precum:

- path-ul necesar descărcării datelor din cadrul funcției de crawl;
- nivelul de recursivitate al parcurgerii URL-urilor vizate;
- timpul maxim de răspuns la accesarea unui URL;
- setarea unui tip de fișiere din cadrul funcționalității de căutare în path-ul cu datele descărcate de utilitar.

În ceea ce privește ansamblul tuturor task-urilor realizate de utilitar, indiferent de funcționalitatea aleasă, acestea sunt regăsite în cadrul clasei **Crawl**. Pe baza acestei clase, numărul de thread-uri furnizate aplicației preiau task-urile necesare îndeplinirii funcționalității selectate.

3.2. Structuri de date de legătură

Elementele de legătură sunt reprezentate de obiectele de tipul Crawl, obiecte ce fac legătura între tipul de operație ce necesită a fi executată și Controller, respectiv CrawlerThreadPool.

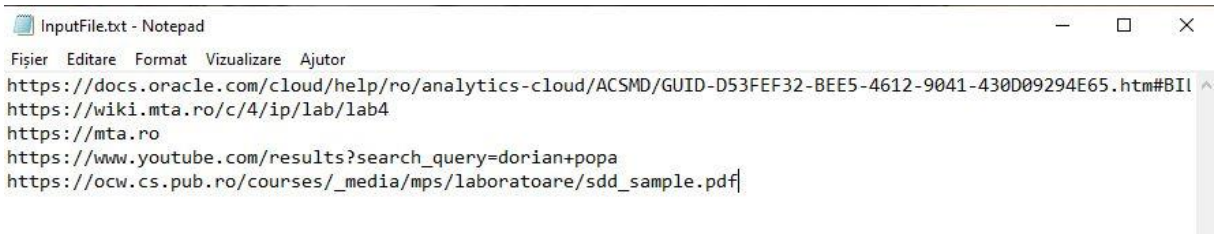
3.3. Structuri de date temporare

În cadrul rulării aplicației ca elemente temporare putem menționa obiectele de tipul **Task** (care există doat cât timp sun in coada de așteptare și când se execută), precum și obiectele de tipul **CrawlerException**

3.4. Formatul fișierelor utilizate

În ceea ce privește rularea aplicației, mai multe argumente pot fi transmise de utilizator:

1) **Fisierul de intrare** -> acest fisier este responsabil de furnizarea către aplicație a URL-urilor de pe care informația încearcă a fi accesată și descarcată. Acesta este singurul argument obligatoriu atunci când se dorește folosirea funcției de crawl. Fișierul de intrare trebuie să conțină câte o adresă URL pe fiecare linie.



2) **Fișierul de configurare** -> acest fișier poate lipsii la rularea aplicației, scenariu în care utilitarul va utiliza parametrii de configurare standard. Acest fișier conține elementele ce pot fi modificate de utilizator în ceea ce privește activitatea aplicației precum:

a) Locația de pe stația de lucru unde datele sunt descărcate;

b) Nivelul de recursivitate pentru parcurgerea URL-urilor initiale furnizate de fișierul de intrare;

c) Limita de dimensiune a datelor ce sunt descărcate [MB];

d) Setarea unor restricții de abordare a utilitarului în functie de funcționalitatea aleasă.

Un exemplu de fișier de configurare este prezentat în Figura(1):

Figura 1

3) **Modul de activitate al utilitarului** -> Acest parametru setează utilitarul de rula în una dintre cele 3 funcționalități ale sale: crawl, search, sitemap. În urma setării unei modalități de funcționare, utilitarul poate necesita și alți parametrii precum:

- tipul de date/ cuvinte cheie pe baza cărora se realizează funcția de search;
- adresa URL pe baza căreia se realizează sitemap-ul;
- cuvinte cheie pentru listarea conținutul descărcat.

Exemple de utilizare a utilitarului:

1. crawler crawl urlFile.txt -config confFile.json -> folosirea funcționalității de crawl din cadrul utilitarului pe baza datelor furnizate de fișierul de config "confFile.json";
2. crawler sitemap /sites/wiki.mta.ro -> ierarhizarea datelor descărcate din cadrul documentului rădăcină /sites/wiki.mta.ro.

4. Modelul arhitectural

4.1. Arhitectura sistemului

4.1.1. Modelele arhitecturale folosite

Soluția actuală a fost proiectată după modelul arhitectural **Model-view-controller**.

Componenta View oferă funcționalități de afișare a mesajelor către utilizator, fie în linia de comandă (standard output), fie într-un fișier specificat de acesta.

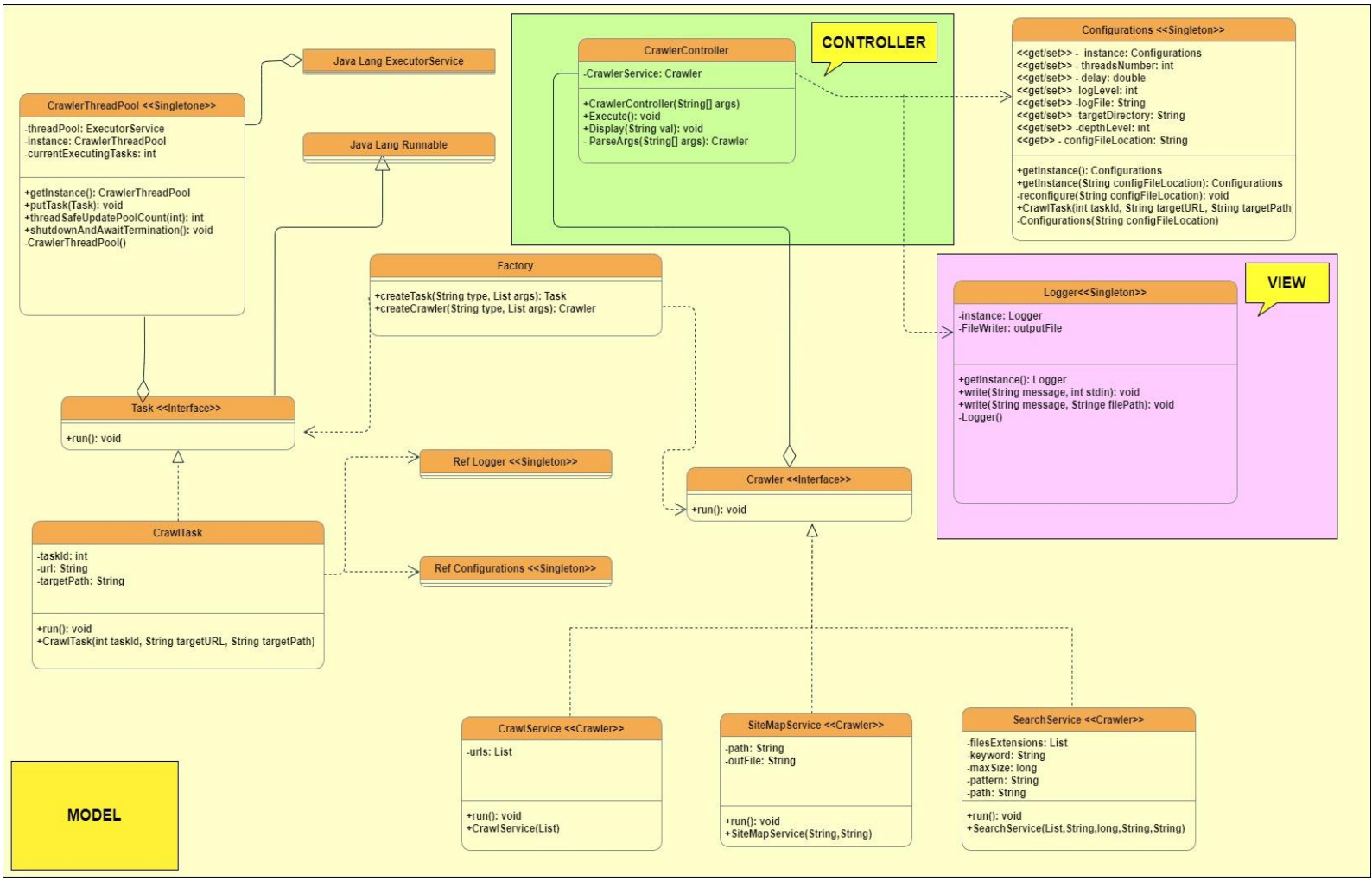
Componenta Model este responsabilă de toată arhitectura internă a aplicației, aceasta procesează datele, inițializează thread-urile, salvează fișierele, controlează afișarea și gestionează erorile apărute.

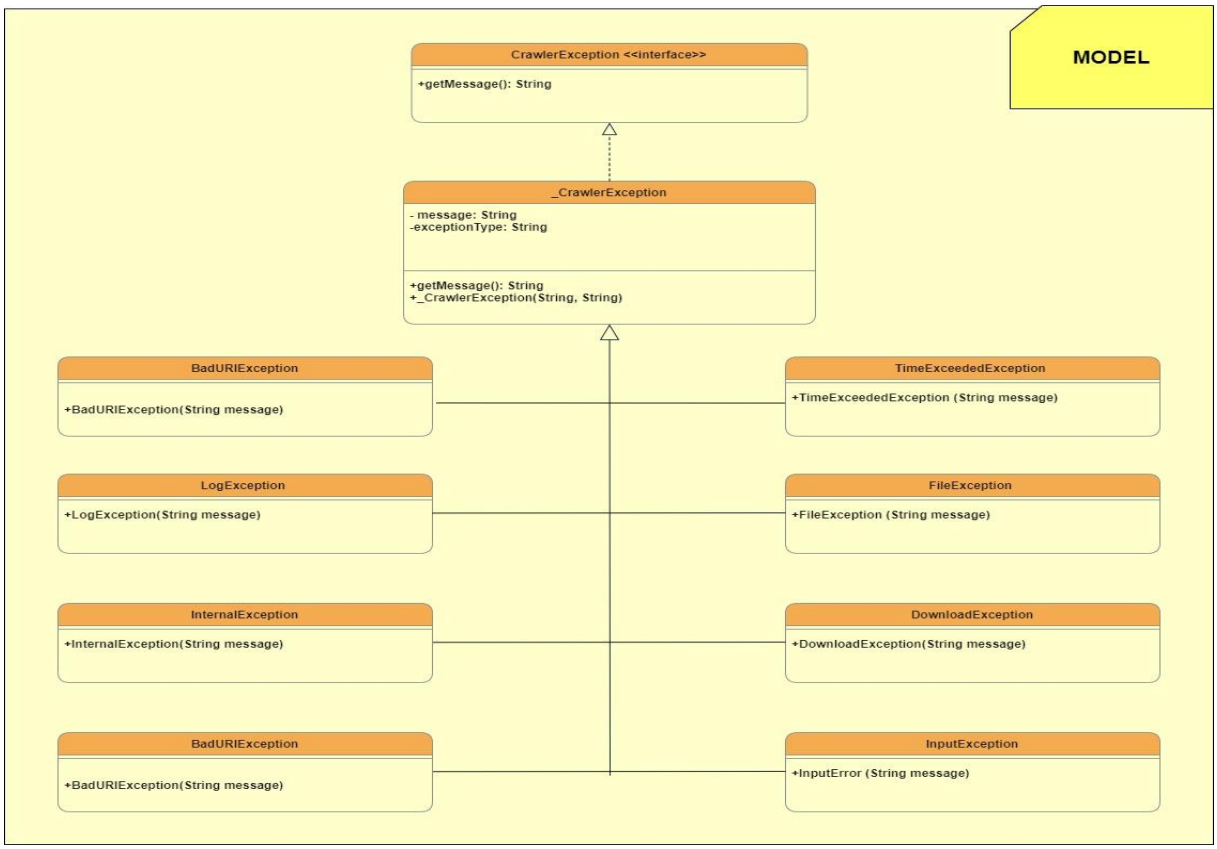
Ultima componentă, dar și cea mai importantă este componenta controller, aceasta este responsabilă de gestionarea comenzilor primite de la utilizator și inițializarea componentelor model corespunzătoare pentru satisfacerea cerințelor. Aplicația oferă utilizatorului posibilitatea de a descărca un site web, de a căuta anumite fișiere din cadrul site-urilor deja descărcate după o serie de criterii și posibilitatea de a afișa un site descărcat într-o formă arborescentă.

4.1.2. Diagrama de clase

Diagrama de clase de mai jos prezintă componentele aplicației și relațiile dintre acestea.

Am evidențiat marile părți componente ale aplicației.





4.2. Descrierea componentelor

Aplicația este alcătuită din următoarele clase interconectate.

- **CrawlerController** (Controller) – parsează argumentele primite de la utilzator în linia de comandă, inițializează clasele globale (Singleton) Logger și Configurations și execută funcționalitatea specificată de utilizator.
    - **Membrii:**
        - **crawlerService** – este un obiect de tipul Crawler ce reprezintă tipul de operație ce trebuie executată (crawl, search sau site map).
    - **Metode**:
        - **CrawlerController** – este constructorul clasei.
        - **Execute** – este metoda apelată de funcția main pentru a începe execuția operației cerute de către utilizator.
        - **Display** – este metoda ce poate fii apelată pentru a afișa informații către utilizator.
        - **parseArgs** – este metoda privată ce este apelată din constructor, va instanția clasele Configurations și Logger

și va returna un obiect de tipul Crawler prin intermediul clasei Factory.

- **Crawler (**Model) – aceasta este o interfață pentru cele trei funcționalități oferite pană la momentul actual de aplicație.
    - **Metode:**
        - **run –** este metoda virtuală pură pentru execuția diferitelor funcții.
- **CrawlerService** : **Crawler** (Model) - aceasta este o clasa ce implementează interfața **Crawler** ce va fi responsabilă de inițializarea clasei **CrawlerThreadPool** (Singleton) și de descărcarea site-urilor web solicitate de utilizator.
    - **Membrii**:
        - **urls** – este o listă de URL-uri furnizate de către utilizator ce trebuie descărcate.
    - **Metode**:
        - **run** – metodă ce va inițializa thread pool-ul, va începe descărcarea paginilor, specificând locația pentru descărcare și va aștepta ca toate task-urile din coada pool-ului să se finalizeze.
- **SearchService : Crawler** (Model) – asemănător clasei CrawlerService, implementează interfața Crawler și este responsabilă pentru căutarea de fișiere într-un anumit site deja descărcat după anumite criterii.
    - **Membrii:**
        - **extensions –** este o listă de extensii de interes specificate de utilizator.
        - **keyword –** este un cuvânt cheie specificat de utilizator, se vor afisă toate fișierele ce conțin cuvântul respectiv, sau conținutul daca aceste este un fișier.
        - **maxSize –** este o dimensiune maximă specificată de utilizator, dacă un fișier depășește această dimensiune, el nu va fi afișat.
        - **pattern –** este un sir de caractere ce conține o expresie regulată specificată de utilizator.
    - **Metode:**
        - **run –** metoda ce va executa căutarea fișierelor ce se privesc criteriilor specificate.
- **SiteMapService : Crawler** (Model) – clasă ce implementează interfața Crawler, se va ocupa cu afișarea unui site deja descărcat în formă arborescentă.

  - **Membrii:**
    - **path –** calea către fișierul rădăcină al site-ului deja aflat în sistemul de fișiere local.
    - **outFile** – calea fișierului de afișare în cazul în care este specificat de către utilizator.
  - **Metode:**
    - **run –** metodă ce va executa căutarea resursivă în directorul dat și va returna forma arborescentă.
- **CrawlerThreadPool** <Singleton> (Model) – această clasă se va ocupa de gestionarea thread pool-ului la nivelul programului, va lucra cu task-uri, va oferi mecanisme de oprire a tuturor thread-urilor și de punere în coada de execuție unor task-uri noi, pe lângă aceste funcționalități va oferi metode specifice pattern-ului de programare singleton.
  - **Membrii:**
    - **threadPool –** obiect de tip **ExecutorService** ce va reprezenta thread pool-ul.
    - **instance –** membru specific singleton.
    - **currentExecutingTasks –** membru thread safe ce va memora numarul de task-uri ce sunt in execuție și in coada de așteptare.
  - **Metode:**
    - **getInstance** – specific singleton.
    - **CrawlerThreadPool –** constructor ce va folosi instanța clasei Configurations pentru inițializare.
    - **putTask** – funcție prin intermediul căreia se aduga un nou task coadei de așteptare.
    - **threadSafeUpdatePoolCount –** funcție thread safe ce va modifica si returna **currentExecutingTasks** (1 pentru incrementare, 0 pentru verificare si -1 pentru decrementare).
    - **shutdownAndAwaitTermination –** functie blocantă ce va opri forțat toate thread-urile, implicit task-urile aflate în execuție.
- **Factory** (Mode) – această clasă se va ocupa de generarea de obiecte după anumite criterii specificate date ca parametru.
  - **Metode:**
    - **createTask** – va returna un obiect de tipul Task.
    - **createCrawler –** va returna un obiecvt de tipul Crawler.

- **Configurations** <Singleton> (Model) – această clasă globală va conține toate informațiile de configurare fie găsite într-un fișier dat de către utilizator, fie un fișier standard, toți membrii acestei clase sunt specificați in structura fișierului de configurare.
    - **Metode:**
        - **get** – metode de get pentru fiecare membru.
        - **getInstance** – specific singleton, poate primi opțional calea unui fișier de configurare, este necesar pentru prima inițializare în cazul în care un fișier este specificat de către utilziator, se poate reapela acastă funcție cu un alt fișier de configurare caz în care se va realiza reinițializarea membrilor.
        - **reconfigure** – funcția ce se va ocupa de inițializarea/reinițializarea membrilor.
- **Task** (Model) – aceasta este interfața ce extinde clasa Runnable și va fi folosită pentru crearea de task-uri specializate pentru a fi rulate în cadrul thread pool.
    - **Metode:**
        - **run –** este metoda virtuală pură pentru execuția diferitelor task-uri.
- **CrawlTask : Task** (Model) – clasa ce implementează interfața task, această clasă se va ocupa de descărcarea site-urilor, analiza referințelor și pornirea recursivă de alte task-uri ce vor descărca referințele găsite, pe lângă acest lucru se vor înlocui link-urile cu calea locală și se vor salva corespunzător.
    - **Membrii**:
        - **taskId –** identificatorul task-ului curent, folosit și pentru detecția adâncimii.
        - **url –** url-ul ce urmează să fie descărcat.
        - **targetPath** – calea către directorul țintă.
    - **Metode:**
        - **CrawlTask –** constructorul clasei.
        - **run –** metoda ce va realiza funcționalitatea clasei.
- **Logger** <Singleton> (View) – această clasă globală este responsabilă de toate afișările efectuate de program, fie într-un fișer specificat, fie pe ieșirea standard.
    - **Membrii:**
        - **instance** – specific singleton.
        - **outputFile –** în cazul în care un fișier de ieșire este specificat, acest membru va fi inițializat.

    - **Metode:**
        - **getInstance** – specific singleton.
        - **write –** funcție thread safe pentru afișare.
        - **Logger –** constructor ce se va folosi de clasa Configurations.
- **CrawlerException** (Model) – aceasta este interfața ce va fi folosită pentru semnalarea erorilor apărute în timpul execuției programului.
    - **Metode:**
        - **getMessage –** este metoda virtuală pură pentru returnarea unui șir de caractere ce va conține informații despre tipul erorii, locul apariției precum și un mesaj special.
- **_CrawlerException : CrawlerException** (Model) – este o clasă abstractă ce implementează interfața **CrawlerException** și oferă o implementare mai simplă pentru celelalte clase de excepție ce vor fi derivate din aceasta.
    - **Membrii:**
        - **message –** mesajul de eroare specific
        - **exceptionType –** tipul erorii
    - **Metode:**
        - **getMessage –** va returna un mesaj specific de eroare
        - **_CrawlerException –** constructorul clasei

Celelalte clase de excepție sunt derivate din clasa **_CrawlerException** conform diagramei de mai sus, acestea au doar un constructor ce va transmite clasei de baza (**_CrawlerException**) tipul erorii precum și mesajul acesteia.

4.3. Restricții de implementare

Modulele aplicației (clasele acesteia) trebuie să respecte următoarele restricții de implementare:

- Comentarea codului va fi conform standardului “Java Code Conventions**”.**
- Numele claselor, membrilor, metodelor, precum și variabilele vor respecta convențiile detaliate în documentul “Coding Conventions”.
- În cadrul proiectului se va folosi JAVA SDK 11.

4.4. Interacțiunea dintre componente

Atunci când aplicația este pornită de către utilizator, se va lansa în execuție clasa Main din modului **WebCrawler** care ulterior va instanția obiectul **CrawlerController**, obiect ce va prelua controlul asupra execuției și managementul programului, acesta reprezentând partea de controller din modelul MVC, mai departe va inițializa obiectele globale **Logger**, **Configurations**, va stabili pe baza input-ului de la utilizator tipul de **Crawler** ce trebuie generat cu ajutorul clasei **Factory** și al metodei **ParseArgs**, apoi va fi executată funcția **run** din cadrul obiectului returnat.

În acest moment sigurul caz particular este cazul **CrawlService**, generat de comanda (crawl), acesta va inițializa și obiectul global **CrawlerThreadPool** care va executa pe un număr prestabilit (din fișierul de configurare) de thread-uri task-urile ce vor fi inițializate.

În toate cazurile după execuția funcției de **run,** specifică obiectelor de tip **Crawl,** controlul execuției se va întoarce cărte modulul **CrawlerController** și, ulterior, modulului **WebCrawler** (funcția Main) programul finalizându-se.

În oricare moment al execuției programului, controlul poate fi cedat componentei **Logger** pentru afișarea de mesaje.

5. Elemente de testare

5.1.Teste vizate

Pentru testarea aplicației va fi folosită o suită de teste dezvoltate în acest scop.

5.1.1. Testarea funcționalității de parsare a fișierului de configurare

Se va testa posibilitatea abordării diferitelor scenarii de configurare prin intermediul fișierului specificat. Se va verifica funcționalitatea aplicației de a extrage parametrii de interes din cadrul fișierului. În lipsa acestuia se va aborda un scenariu prestabilit.

5.1.2. Testarea serviciilor oferite

Aceste teste sunt dezvoltate cu scopul de a determina comportamentul fiecărui serviciu oferit de utilitar:

- Extragerea informațiilor dorite din cadrul paginilor Web furnizate de utilizator;
- Posibilitatea navigării recursive în cadrul domeniului Web specificat;
- Determinarea corectă a scheletului elementelor descărcate – site map;
- Capabilitatea localizării informațiilor de interes dorite – search;
- Înlocuirea corectă a dependințelor dintre paginile descărcate.

5.1.3. Testarea mecanismelor de toleranță la erori

Testarea mecanismelor de tratare a erorilor se va face prin utilizarea unei suite de scenarii care vizează comportamentul aplicației.

Scenariile posibile sunt următoarele:

- Furnizarea de fișiere de configurare și inițializare greșit structurate;
- Încercarea accesării unor URI- uri eronate;
- Rularea aplicației cu un număr de fire de execuție ce depășește limita maximă suportată de stația de lucru;
- Erori ce pot apărea în urma interacțiunii dintre sistemul de operare și aplicația în cauză:
    - Dimensiunea datelor ce se doresc a fi descărcate exced capacitatea de stocare;
    - Existența și permisiunile asupra fișierelor și zonelor de date cu care lucrează aplicația;
    - Lipsa unei conexiuni stabile la internet de pe stația de lucru.
- Situațiile ce pot apărea în urma utilizării utilitarului cu datele de configurare prestabilite:
    - Determinarea locației de stocare a datelor;
    - Stabilirea delay-ul maxim admis în funcție de resursele mașinii de calcul.
    - Determinarea nivelului maxim de recursivitate.

5.1.4. Testarea interoperabilității componentelor utilitarului

Testarea unor cicluri de utilizare complete a aplicației software. Aceste teste urmăresc plierea tuturor componentelor pe diferitele scenarii ce pot apărea. Un astfel de exemplu îl constituie actualizarea automată a site map-ului în urma descărcării datelor.