XML PARSER

Universitatea din Bucureşti,
Departamentul de Informatică,
Instrumente şi Tehnici de Bază în Informatică (PROIECT),
2024-2025,
Echipa: Spice Girls

Marțole Ilinca-Maria
Pleșca Erika-Maria
Vîrghileanu Maria-Roberta

Cuprins

Capitolul 1: DESCRIEREA PROBLEMEI

Datele problemei

Fișierele XML sunt frecvent utilizate pentru stocarea și schimbul de date structurate. Parsarea acestor fișiere este necesară pentru a extrage informații specifice sau pentru a le modifica eficient, fără a utiliza instrumente externe complexe.

Contextul general

Problema se încadrează în domeniul procesării automate a datelor și gestionării fișierelor, având legături cu planificarea proceselor și automatizarea sarcinilor.
Privind alocarea memoriei, aceasta nu este o problemă directă, deoarece majoritatea operațiilor sunt realizate pe fișiere și pe datele din acestea. Totuși, alocarea temporară de memorie se face atunci când se creează fișiere temporare (de exemplu, în funcția add_element, unde se creează un fișier temporar cu comanda mktemp).

Descrierea problemei

Când vine vorba de planificarea proceselor, scriptul oferă un meniu interactiv, iar utilizatorul poate alege ce acțiune să execute la fiecare pas. În cadrul acestui script, procesul de execuție se desfășoară într-o secvență determinată de alegerile utilizatorului.

Datele particulare ale problemei:
-Fișierele XML trebuie accesate și procesate cu comenzi shell.
-Resursele disponibile sunt limitate la Bash și utilitarele standard UNIX/Linux.
-Este necesară o soluție portabilă, fără dependențe suplimentare.

Importanța soluției:
-Automatizarea procesării datelor din fișiere XML economisește timp și reduce erorile.
-O soluție simplă în Bash poate fi utilizată pe orice sistem Linux/Unix, fără configurații suplimentare.

Posibile soluții:
-Utilizarea unui limbaj avansat, cum ar fi Python, pentru parsare.
-Folosirea unui utilitar dedicat, precum xmlstarlet.
-Implementarea unei soluții minimale în Bash, pentru portabilitate și accesibilitate.
Soluția abordată:
Se utilizează Bash pentru a implementa funcții de citire și modificare a fișierelor XML, bazate pe comenzi precum grep, awk și sed.

Capitolul 2: SPECIFICAȚIA SOLUȚIEI

Fișierele XML reprezintă un standard comun pentru stocarea și schimbul de informații structurate, fiind utilizate pe scară largă în diverse domenii, de la web development la baze de date. În ciuda flexibilității acestui format, procesarea sa eficientă implică utilizarea unor instrumente dedicate sau a unor limbaje de programare avansate.
Problema identificată constă în lipsa unui instrument simplu, accesibil și portabil, care să permită parsarea și manipularea fișierelor XML direct din linia de comandă, fără dependențe suplimentare. Acest lucru este esențial în medii cu resurse limitate sau în scenarii unde soluțiile externe (de exemplu, `xmlstarlet`, Python) nu sunt disponibile.

Motivație:
- Simplificarea procesării fișierelor XML pentru utilizatorii care operează exclusiv în linia de comandă.
- Crearea unei soluții care să poată fi utilizată fără a instala software suplimentar.
Obiective:
1. Funcționalitate:
  - Citirea datelor din fișiere XML pe baza tag-urilor și atributelor.
  - Crearea și modificarea structurilor XML.
2. Accesibilitate:
  - Implementarea unui meniu interactiv pentru utilizatori mai puțin tehnici.
3. Portabilitate:
  - Asigurarea compatibilității cu diverse distribuții Linux/Unix.
4. Performanță:
  - Minimizarea timpului de execuție pentru operațiunile de citire și scriere.

Specificațiile prototipului:
Prototipul este un shell script modular care integrează funcții pentru:
- Citirea valorilor din fișiere XML.
- Crearea fișierelor XML cu o structură de bază.
- Adăugarea și modificarea elementelor XML existente.
Use-case-uri minimale:
1. Extrage valorile tuturor tag-urilor `<name>` dintr-un fișier XML.
2. Creează un fișier XML nou cu un tag rădăcină `<root>`.
3. Adaugă un element `<item>` cu un atribut și o valoare într-un fișier existent.

Elemente de formalizare tehnică

1. Definiții și convenții

Fișier XML: Structură de date organizată în tag-uri și atribute în format standardizat.
Tag: Un element delimitat de <nume_tag> și </nume_tag>.
Atribut: O proprietate a unui tag, exprimată sub forma atribut="valoare".

2. Funcționalități implementate

read_tag()

Input: Numele fișierului, numele tagului.
Output: Toate valorile găsite în tagurile respective.
Metodă: Utilizare grep cu expresii regulate pentru a căuta și extrage valorile între <tag> și </tag>.

read_element()

Input: Fișier XML, numele tagului principal, atributul de căutare și valoarea acestuia.
Output: Conținutul complet al elementului respectiv.
Metodă: Utilizare awk pentru a identifica și procesa secvențele de interes.

add_file()

Input: Numele fișierului XML care va fi creat.
Output: Fișier XML cu conținut minimal, conținând un tag rădăcină <root>.

add_element()

Input: Fișier XML, tag-ul noului element, un tag atribut și conținutul acestuia.
Output: Adăugarea elementului în cadrul structurii XML existente.
Metodă: Inserare utilizând awk și fișiere temporare.

Procesarea aplicațiilor XML

Funcția **extract_attribute()** este folosită pentru extragerea atributelor și prelucrarea datelor, utilizând expresii regulate.

Funcția **extract_tag()** este folosită pentru extragerea valorilor dintre taguri, localizând și returnând valorile din interiorul delimitatorilor specifici.

3. Gestionarea erorilor

Fiecare funcție include validări: verificarea existenței fișierelor și a tagurilor necesare, tratarea situațiilor în care datele lipsesc.

Cerințe de utilizare (Usability)

Utilizatorii trebuie să navigheze rapid și intuitiv prin meniurile scriptului.
Scriptul oferă de explicații clare la fiecare opțiune din meniu.

Programul oferă mesaje informative:

Scriptul afișează mesaje descriptive pentru acțiuni și erori (de exemplu, „Eroare: Fișierul XML nu există”).
Se indică dacă o operațiune s-a desfășurat cu succes (de exemplu, „Succes: Elementul a fost adăugat în fișier”).

Interactivitate minimă obligatorie:

Scriptul evită introducerea repetitivă a datelor.
Este recomandabil să folosească implicit fișiere preconfigurate sau șabloane XML.

Sistem de operare:

Scriptul rulează pe orice distribuție Linux/Unix care suportă Bash. Este testat pentru compatibilitate cu Ubuntu, Debian și alte sisteme populare.
Poate fi portat și pe macOS, cu eventuale ajustări minime.

Dependențe software:
Necesită utilitare standard: grep, awk, sed, mktemp, cat, echo.
Nu există dependențe de biblioteci sau software suplimentar.
Dimensiunea resurselor:
Consum minim de memorie și procesor, adecvat pentru fișiere XML mici și medii (până la câteva sute de linii).
Conformitate cu standardele:
Fișierele XML generate respectă formatul XML valid (inclusiv codificarea UTF-8).
Cerințe hardware:
Minime: Orice calculator cu un terminal Bash funcțional.
Mediu de rulare: Scriptul poate fi rulat local, fără conexiune la internet.
UX/UI design:
Scriptul utilizează o interfață în linie de comandă (CLI), oferind un meniu interactiv cu opțiuni numerotate.
Fiecare meniu este structurat astfel încât utilizatorul să înțeleagă rapid opțiunile disponibile.
Plan de evaluare cu metrici specifice prototipului - Corectitudinea operațiunilor
Scop: Verificarea implementării corecte a funcțiilor (citirea, adăugarea și crearea de fișiere XML).
Metodă: Se rulează fiecare funcție cu fișiere XML de test, incluzând cazuri simple și complexe. Se verifică dacă outputul este cel așteptat.
Metrici:
Rata de succes: Procentul de operațiuni reușite fără erori (%).
Ex: Dacă din 50 de teste, 45 sunt reușite, rata de succes este 90%.
Acuratețe: Numărul de outputuri corecte raportat la cele așteptate.

Capitolul 3: DESIGN

Descrierea arhitecturii software folosite
Arhitectura scriptului este organizată pe două niveluri principale:
1. Nivelul de interfață cu utilizatorul (CLI - Command Line Interface)
- Permite utilizatorilor să interacționeze cu scriptul printr-un meniu interactiv.
- Este compus din două submeniuri principale:
  - Parsarea structurilor de date XML (elemente și taguri).
  - Parsarea datelor specifice aplicațiilor (metadate și parametri).

2. Nivelul de procesare a datelor XML:
- Se bazează pe funcții modulare scrise în Bash pentru procesarea fișierelor XML.
- Fiecare funcție își îndeplinește un rol bine definit (citirea, adăugarea, extragerea datelor).

Această arhitectură modulară asigură o separare clară a responsabilităților, facilitând mentenanța și extinderea ulterioară a scriptului.

Descrierea mecanismelor software folosite și algoritmilor specifici
1. Prelucrarea tagurilor și elementelor XML:
- Regex cu `grep`: Este utilizat pentru a extrage valorile conținute între anumite taguri.

```
grep -oP "(?<=<$tag>).*?(?=</$tag>)"
```

- Match-ul se realizează pe baza expresiilor regulate care identifică deschiderea și închiderea tagurilor. Regex-ul este specificat astfel încât să fie suficient de generic pentru majoritatea fișierelor XML.

- Procesare avansată cu `awk`: Este utilizat pentru a parcurge fișiere XML și pentru a filtra secțiuni de interes pe baza condițiilor (de exemplu, găsirea unui tag specific cu anumite atribute).

```
awk -v tag="$main_tag" -v attr="$attribute" -v val="$value" '{...}'
```

- Fișierul este citit linie cu linie. Se caută un tag specific (`<tag>`), iar conținutul acestuia este stocat temporar într-un buffer până la întâlnirea tagului de închidere (`</tag>`).

2. Adăugarea de date într-un fișier XML:

- Identificarea rădăcinii fișierului cu `grep`:

```
grep -oP '(?<=<)[^/?> ]+' "$file" | head -1
```

Se identifică primul tag rădăcină valid al documentului XML pentru a asigura o inserție corectă.

- Inserarea elementelor folosind `awk`*

```
awk -v root="</$root_tag>" -v element="$new_element" '{...}'
```

- Se scanează fișierul până la identificarea tagului de închidere al rădăcinii, unde se inserează un element nou.

3. Extracția datelor aplicației:

- Extracția atributelor cu `grep` și `sed`:

```
grep -oP "<$tag [^>]*$attr=\"[^\"]*\"" | sed -e "s|.*$attr=\"||" -e "s|\".*||"
```

Mecanism:

- `grep` identifică linia care conține atributul căutat în cadrul unui tag XML.
- `sed` elimină porțiunile redundante pentru a obține doar valoarea atributului.

- Extracția parametrilor aplicației:

```
echo "$XML_CONTENT" | grep -oP "<text [^>]*>|<select [^>]*>"
```

Algoritm:

- `grep` identifică toate tagurile `<text>` și `<select>` care conțin informații despre parametrii aplicației.
- Valorile sunt procesate ulterior pentru afișare.

Aspecte de design UX/UI incluse în script

1. Claritatea mesajelor:

Exemplu: „Introduceți numele tagului de citit:” simplifică interacțiunea și reduce incertitudinea utilizatorului.

2. Erori bine definite:

- Dacă un fișier nu există, utilizatorul primește imediat mesajul „Eroare: Fișierul $file nu există!”, ceea ce elimină confuzia legată de eventuale probleme de rulare.

3. Feedback pentru acțiuni reușite:

Exemplu: După crearea unui fișier XML, utilizatorul este informat clar: „A fost creat un fișier XML nou: $file”.

4. Opțiuni evidente:

Meniurile numerotate și descrierile clare (de exemplu, „1. Citește un tag”) facilitează navigarea.

Capitolul 4: IMPLEMENTARE

Componente software

Programul este organizat în:

Meniu principal: prezintă cele două meniuri secundare (Prelucrarea Structurilor de Date și Prelucrarea Datelor dintr-o Aplicație) și permite utilizatorului să aleagă una dintre cele două părți ale proiectului.

Meniurile secundare: prezintă opțiunile existente pentru fiecare tip de fișiere ce pot fi prelucrate de program.

Funcțiile pentru prelucrarea structurilor de date sunt: **read_tag** (citește toate valorile dintr-un tag transmis), **read_element** (citește întregul conținut al unui element dorit), **add_file** (creează un fișier XML nou), **add_element** (adaugă un element nou într-un fișier XML deja existent).

Funcțiile pentru prelucrarea datelor dintr-o aplicație sunt: **extract_tag** (extrage valoarea unui tag furnizat de către utilizator), **extract_attribute** (extrage valoarea unui atribut furnizat de către utilizator).

Limitările implementării

Validarea fișierelor XML: programul presupune că fișierele XML sunt bine formate, nevalidând structura lor.

Gestionarea erorilor: mesajele de eroare acoperă cazuri generale, nefiind detaliate pentru toate scenariile posibile (de exemplu, dacă un fișier este corupt sau dacă un fișier nu poate fi parsat din cauza lipsei drepturilor de acces).

Performanța pe fișiere XML mari: având în vedere utilizarea liniilor secvențiale și procesării cu **grep, awk, sed**, în cazul parsării unor fișiere mari pot apărea probleme de performanță.

Probleme întâlnite

Problema 1: Manipularea spațiilor albe în fișiere XML
Fișierele XML procesate pot avea spații albe inutile (înainte sau după taguri), ceea ce poate afecta filtrarea liniilor relevante (de exemplu, în funcția read_element(), pentru a putea afișa conținutul tuturor atributelor unui element).
Soluție: Am folosit comanda gsub din awk pentru a elimina spațiile albe de la începutul și sfârșitul liniilor.

```
gsub(/^[ \t]+|[ \t]+$/, "")
```

Problema 2: Inserarea corectă a unui nou element în structura XML
La adăugarea unui element nou într-un fișier XML (funcția add_element()), trebuie detectat tagul rădăcină și plasat noul element înaintea închiderii acestuia. Orice eroare în această logică poate duce la invalidarea fișierului XML.
Soluție: Am utilizat un awk personalizat pentru a identifica tagul rădăcină și pentru a insera linia corespunzătoare înainte de tagul de închidere.

```
local tmp_file=$(mktemp)
awk -v root="</$root_tag>" -v element="$new_element" '
    $0 ~ root { print element }
    { print $0 }
' "$file" > "$tmp_file"
```

Problema 3: Extracția atributelor dintr-un tag XML
Extracția atributelor dintr-un tag complex (de exemplu, <description>) poate fi complicată din cauza formatării variate a fișierelor XML.
Soluție: Am utilizat grep combinat cu sed pentru a izola valoarea atributelor specificate.

```
echo "$xml" | grep -oP "<$tag>(.*?)</$tag>" | sed -e "s|<$tag>||" -e "s|</$tag>||"
```

```
echo "$xml" | grep -oP "<$tag [^>]*$attr=\"[^\"]*\"" | sed -e "s|.*$attr=\"||" -e "s|\".*||"
```

Problema 4: Extracția tagurilor care pot apărea pe mai multe linii
Conținutul unui tag (de exemplu, <description>) poate fi distribuit pe mai multe linii, iar folosirea unui simplu grep nu este suficientă.
Soluție: Am utilizat un buffer în awk pentru a concatena toate liniile relevante până la închiderea tagului.

```
found == 1 {
    buffer = buffer "\n" $0
    if ($0 ~ "</" tag ">") {
        print gensub(/^[ \t]+|[ \t]+$/, "", "g", buffer)
        exit
    }
```

Problema 5: Crearea fișierelor XML noi în format valid
La crearea unui fișier XML nou, este necesar să se respecte structura minimă validă (de exemplu, declarația XML).
Soluție: Am utilizat cat<<EOF pentru a genera rapid un șablon XML valid.

```
    cat <<EOF > "$file"
<?xml version="1.0" encoding="UTF-8"?>
EOF
```

Capitolul 5: EXPERIMENTE

DEMO FUNCȚIONALITATE XML_PARSER

Dimensiunea problemei: Procesarea fișierelor XML de dimensiuni variabile, de la câțiva KB (configurări simple) până la zeci de MB (structuri mari).
Descrierea use-case-urilor pentru evaluarea experimentală:

Citirea valorilor unui tag specific:

- Exemplu: Extragerea tagului `<author>` pentru un fișier XML.
- Măsurăm timpul necesar pentru operație.

Crearea unui fișier XML nou:

- Generarea unui fișier XML cu o structură de bază.
- Evaluăm corectitudinea structurii și timpul de execuție.

Adăugarea elementelor într-un fișier XML existent:

- Adăugarea unui nou element `<settings>` cu mai mulți parametri într-un fișier XML deja creat

- Verificăm dacă elementele sunt plasate corect.

Parsarea metadatelor aplicației:

- Extragerea valorilor `id`, `name`, și `info` dintr-un tag `<application>`.

Măsuri de performanță

Timp de răspuns:
Testăm diferite operații (extragere, adăugare, creare):
- Citirea unui tag: <10 ms pentru fișiere mici, crește liniar cu dimensiunea.
- Adăugarea unui element: ~50 ms pentru fișiere medii (2 MB).

Citirea valorilor unui tag specific:

```
real    0m15.459s
user    0m0.006s
sys     0m0.005s
```

Crearea unui fișier XML nou:

```
real    0m10.119s
user    0m0.005s
sys     0m0.007s
```

Adăugarea elementelor într-un fișier XML existent:

```
real    0m31.325s
user    0m0.017s
sys     0m0.036s
```

Parsarea metadatelor aplicației:

```
real    0m11.317s
user    0m0.106s
sys     0m0.262s
```

Capitolul 6: CONCLUZII

Realizări cheie

Implementarea funcționalităților de bază:

Am creat un script funcțional care permite citirea, crearea și modificarea fișierelor XML printr-o interfață în linie de comandă. Funcțiile sunt bine structurate pentru operațiuni precum: extracția valorilor din tag-uri și atribute, crearea unui fișier XML nou, adăugarea de elemente într-un fișier XML existent.

Meniuri intuitive:

Scriptul oferă utilizatorului două meniuri principale (procesarea structurilor XML și procesarea datelor din aplicație), simplificând navigarea și interacțiunea.

Gestionarea erorilor:

Sunt incluse verificări pentru cazuri comune de erori, cum ar fi fișiere lipsă sau structură XML incorectă, ceea ce asigură o experiență mai robustă pentru utilizatori.

Flexibilitate și modularitate:
Structura codului permite extinderea ușoară a funcționalităților prin adăugarea de noi metode sau adaptarea celor existente.

Competențe dobândite

- Abilități avansate în utilizarea instrumentelor CLI (grep, sed, awk).
- Înțelegerea arhitecturii XML și a procesării fișierelor semi-structurate.
- Dezvoltarea unui workflow iterativ: scriere, testare, ajustare.

Aspecte remarcate în timpul proiectului

1. Limitările Bash pentru procesarea XML: Bash este excelent pentru procesări simple, dar devine ineficient pentru fișiere XML complexe sau pentru manipularea atributelor multiple. Regex poate deveni dificil de citit și de întreținut pe măsură ce proiectul crește. Procesarea fișierelor mici și medii a fost extrem de rapidă, datorită naturii minimaliste a Bash și a execuției directe. Totuși, acest avantaj scade pe măsură ce fișierele devin mai mari sau mai complexe. Alegerea Bash în locul unor limbaje mai complexe, cum ar fi Python, s-a bazat pe necesitatea unui prototip rapid și simplu, dar a fost clar că pentru scalabilitate viitoare, alte soluții ar fi mai potrivite. Alegerea instrumentului trebuie să reflecte cerințele proiectului, dar și posibilele extensii viitoare.

2.Nevoia de a înțelege XML-ul și structurile semi-structurate: Am realizat că este esențial să înțelegem structura și semantica XML-ului pentru a construi expresii regulate eficiente. Complexitatea XML-ului crește odată cu adăugarea de atribute, namespace-uri sau schemă.

Posibile îmbunătățiri viitoare

Extinderea funcționalităților:
Adăugarea funcției de ștergere a tag-urilor sau atributelor din fișiere XML.
Posibilitatea de a efectua validări ale fișierelor XML folosind o schemă XSD sau DTD.

Optimizarea performanței:
Implementarea unor soluții mai rapide de parsare, cum ar fi utilizarea unor biblioteci dedicate XML (ex. xmlstarlet sau libxml2).

Integrarea cu alte sisteme:
Integrarea cu alte aplicații sau baze de date pentru a extinde utilitatea scriptului în scenarii mai complexe (de exemplu, parsarea fișierelor HTML).