Razvoj parsera za komunikacijsku matricu s fokusom na AutoSAR XML format unutar generatora testnog okruženja

Romanić, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Razvoj parsera za komunikacijsku matricu s fokusom na AUTOSAR XML format unutar generatora testnog okruženja

Diplomski rad

Ivan Romanić

Osijek, 2021.

SADRŽAJ

1.UVOD **3**
2.PARSER ZA KOMUNIKACIJSKU MATRICU **5**
2.1. AUTOSAR XML **5**
2.2. Komunikacijska matrica **7**
2.3. Arhitektura parsera za komunikacijsku matricu **8**
2.4. Postojeće rješenja parsera za komunikacijsku matricu **9**
3. PREDLOŽENO RJEŠENJE PARSERA ZA KOMUNIKACIJSKU MATRICU **11**
3.1. C++ programski jezik **11**
3.2. *Tinyxml2* biblioteka **12**
3.3. Predloženo rješenje za parser za komunikacijsku matricu **13**
3.4. Razmjena podataka između parsera i baze podataka **25**
4. TESTIRANJE PREDLOŽENOG RJEŠENJA PARSERA KOMUNIKACIJSKE MATRICE **27**
4.1. Opis testova **27**
4.2. Rezultati ispravnosti zapisivanja podataka u bazu podataka **28**
4.3. Rezultati brzine rada parsera za komunikacijsku matricu **29**
5. ZAKLJUČAK **32**
LITERATURA **32**
SAŽETAK **35**
ABSTRACT **36**
ŽIVOTOPIS **37**

1. UVOD

Moderna vozila sadrže veliki broj elektroničkih upravljačkih jedinica (engl. *Engine Control Unit - ECU*) koje međusobno razmjenjuju podatke. Njih je potrebno testirati. Da bi se automatsko testiranje u sektoru automobilske industrije provodilo učinkovitije, potrebno je imati ispravno i učinkovito implementiran generator testnog okruženja (engl. *Test Environment Generator - TEG*). Generator testnog okruženja je softver koji ubrzava proces testiranja upravljačkih jedinica. Jedan od glavnih dijelova generatora testnog okruženja je i taj zadužen za parsiranje datoteka koje opisuju komunikaciju unutar sustava. Zadatak ovog rada je razviti parser za komunikacijsku matricu s naglaskom na AUTOSAR XML (ARXML) tip datoteka unutar generatora testnog okruženja. Već postoji parser napisan u Python2 i ovaj rad treba pokazati je li moguće razviti parser u C++ programskom jeziku.

U generatoru testnog okruženja postoji dio za stvaranje modela podataka, skupa osnovnih vrijednosti koji trebaju biti prepisani ili nadopunjeni s podacima iz datoteka koje parser uzima kao ulaz. Dio za stvaranje modela pohranjuje u bazu podataka osnovne podatke koji se nadopunjuju s podacima za koje je zadužen parser za komunikacijsku matricu. On parsira datoteke koje sadrže podatke koji se nadopunjuju na model podataka. Te datoteke su komunikacijske matrice. Parser ima zadaću izvući informacije iz datoteka formata ARXML i predati ih dijelu TEG-a koji je zadužen za upisivanje u bazu podataka. Datoteke se mogu shvatiti kao ulaz u parser, a podaci isparsirani u smislene cjeline kao izlaz parsera. Isparsirani podaci se predaju bazi podataka.

Parser za komunikacijsku matricu treba učitati potrebne datoteke te ih procesirati s ciljem prepoznavanja softverskih komponenata te njihovih elemenata. Elemente je potrebno spremiti u strukturu podataka koja je kompatibilna s originalnom strukturom iz datoteka koje se parsiraju. Nakon spremanja u strukturu parser sprema podatke u bazu podataka. Mora postojati bliska komunikacija parsera s dijelom koji radi s bazom podataka. Parser ne može komunicirati s bazom podataka bez korištenja funkcija svojstvenih bazi podataka, pomoću kojih pretražuje bazu podataka za mjestom gdje se mogu spremiti podaci koji su parsirani iz ulaznih datoteka. Parser se poziva od baze podataka.

Datoteke koje se parsiraju su u ARXML formatu, pa stoga parser mora imati mogućnost rada s tim formatom da bi sve potrebne podatke ispravno pronašao i pohranio. Za ARXML format se mogu koristiti alati koji već postoje za XML format, te su već optimizirani i testirani za rad s tim formatom. U ovom radu se koristi alat za parsiranje XML formata naziva "tinyxml2". "tinyxml2" je nadograda "tinyxml" alata. Jednostavan je za korištenje kao i "tinyxml", ali je brži i učinkovitiji. Parser za komunikacijsku matricu je kao i ostali dijelovi testnog okruženja razvijen u C++, za razliku od prijašnje verzije koja je pisana u Pythonu2, što daje novoj verziji određene prednosti nad prijašnjom, te koristi objektno orijentiranu paradigmu koji C++ programski jezik podržava.

Nakon implementacije potrebno je i testirati rad parsera, što je učinjeno na dvije razine: testiranje ispravnosti parsiranih podataka te testiranje performansi. Testiranje ispravnosti parsiranih podataka se odnosi na provjeru sadržavaju li sve cjeline parsiranih podataka sve podatke koje sadržavaju i u AUTOSAR XML dokumentu, te spremaju li se te cjeline podataka ispravno u bazu podataka. Testiranje performansi se odnosi na mjerenje brzine parsiranja te je li brzina kojom parser radi prihvatljiva za rad generator testnog okruženja.

Rad je strukturiran na sljedeći način. U drugom poglavlju je opisan opseg trenutnog rješenja problema parsera za komunikacijsku matricu, te postojeća rješenja. U trećem poglavlju se nalazi predloženo rješenje za problem. U četvrtom poglavlju se nalazi testiranje predloženog rješenja za problem.

2. PARSER ZA KOMUNIKACIJSKU MATRICU

Ovo poglavlje opisuje opseg problema razvoja parsera za komunikacijsku matricu te trenutno rješenje parsera koje ovaj rad za cilj ima razviti u C++. Parser za komunikacijsku matricu je dio generatora testnog okruženja, koji se temelji na AUTomotive Open System ARchitecture (AUTOSAR) okruženju i čija je zadaća testiranje automobilskih komunikacijskih sustava.

AUTOSAR se sastoji od 3 sloja:

1. mikro upravljač - sloj koji direktno pristupa sklopovlju

2. RTE (engl. *runtime environment*) - posrednički sloj između mikro upravljača i aplikacijskog sloja

3. aplikacijski sloj - najviši sloj, sadrži softverske komponente specifične za određene aplikacije, čija je svrha izvršenje određenih zadataka

Svrha AUTOSAR-a je stvaranje otvorenog industrijskog standarda za automotiv arhitekturu softvera između dobavljača i proizvođača automobilskih dijelova. Standard se sastoji od skupa specifikacija koje opisuju komponente softverske arhitekture, te definiraju njihova sučelja [1].

Generator testnog kruženja se nalazi na 2. sloju AUTOSAR-a, u posredničkom sloju. Pošto je parser za komunikacijsku matricu dio generatora testnog okruženja, on se isto nužno nalazi na tom sloju.

2.1. AUTOSAR XML

Ulaz u parser za komunikacijsku matricu čine ARXML datoteke. To su vrste XML datoteka koje se koriste specifično u automotiv razvojnom okruženju za spremanje podataka o specifičnim komunikacijskim ulazima i izlazima koji se koriste, paketima koji se šalju i softverskim komponentama koje ih obrađuju. ARXML matrica je definicija XML jezika za razmjenu AUTOSAR modela i opisa [2].

Centralni podatkovni format za opis informacija o modeliranju cijelog automobilskog sustava je prikazana u ARXML matrici. Ta je matrica generirana od strane jedne AUTOSAR radne grupe iz jednog prikaza ujedinjenog jezika za modeliranje (eng. *Unified Modeling Language* - UML) AUTOSAR meta modela te to odražava u modelnim elementima dokumentiranim od strane AUTOSAR specifikacija [3].

ARXML datoteke predstavljaju konfiguraciju jednog ECU-a objašnjenu sve do primitivnih tipova podataka koji se koriste. Ona predstavlja specifične portove, softverske komponente i signale koji se koriste u pojedinom ECU u komunikaciji između dva ili više ECU-a. ARXML definira sve tipove podataka potrebne za specifičnu ECU komunikaciju, od osnovnih tipova (int, float, string) do onih složenijih (liste, objekti). Koristeći osnovne tipove, ARXML gradi podatkovni model trenutne komunikacije u obliku signala i pripadajućih portova kojima se šalju. Također sadrži i popis softverskih komponenti u koje se generira kôd testnog okruženja [4].

ARXML datoteke su veoma složene XML datoteke. Složenost varira ovisno o ECU-u, te potrebama složenosti softverske podrške ECU-a. Veličina datoteke može imati raspon od nekoliko tisuća kilobajta do nekoliko stotina megabajta, što izravno utječe na brzinu rada parsera, s obzirom na to da što više podataka sadrži jedna datoteka, to će više vremena trebati parseru da obradi tu datoteku. Obrađivanje tih datoteka je temeljni problem parsera za komunikacijsku matricu.

Kao u XML, ARXML je građe stabla. XML stablo počinje s korijenskim elementom s kojeg se granaju njegovi podelementi. Podelementi se često zovu elementi djeca, a elementi pod kojima se nalaze se gledaju kao njihovi roditelji. Svaki element počinje i završava s *tagom*. To je oznaka početka i kraja nekog elementa stabla te se nalazi u šiljastim zagradama. Na slici 2.1. se nalazi prikaz jedne jednostavne XML datoteke. Prve informacije u XML datoteci je verzija XML-a i kodiranje znakova. Korijenski element ili čvor u toj datoteci je *bookstore* i nalazi se između dva *tag*-a imena *bookstore*. Pod tim elementom se nalaze elementi djeca ili podčvorovi *book*, isto između svojih *tag*-ova. Svaki *book* element ima svoje posebne elemente kao što su *title*, *author*, *year* i *price* koji sadrže podatke u sebi.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="cooking">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
<book category="children">
<title lang="en">Harry Potter</title>
<author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
</book>
<book category="web">
<title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
<price>39.95</price>
</book>
</bookstore>
```

Sl. 2.1. Prikaz jednostavne XML datoteke

2.2. Komunikacijska matrica

Konkretna definicija određenog AUTOSAR modela nalazi se jednoj ARXML datoteci, koja se zove i matricom. ARXML matrica čini skup svih podataka softverskih komponenata automobila u jednoj ARXML datoteci. U modelu će na primjer biti utvrđeno da je softverska komponenta provedena kroz portove koji mogu podupirati komunikaciju tipa pošiljatelj-primatelj ili klijent-server. Matrica pokriva semantički dio prijenosa podataka kao što su semantička ograničenja. Ta ograničenja su od velike važnosti za prijenos podataka [5].

ARXML datoteka se sastoji od paketa podataka od kojih svaki predstavlja nakupinu određenih tipova softverskih komponenti automobila. Paketi mogu biti različite građe ovisno o datotekama. Odnos entiteta u paketima je opisan u AUTOSAR modelu.

ARXML datoteke komunikacijske matrice čine paketi s raznim tipovima podataka. Neki od tih podataka su:

- Cluster
- ECU
- Frame
- PDU (primary data unit)
- Signal
- Data Type

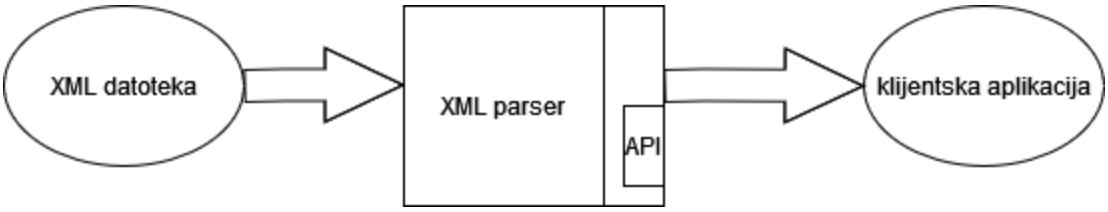
Osnovni tip podataka je PDU. To je kolekcija signala koja sadrži sučelja klijent-server i pošiljatelj-primatelj. Drugačiji slojevi u AUTOSAR-u imaju drugačije PDU-ove.

2.3. Arhitektura parsera za komunikacijsku matricu

Parser mora moći raditi s ARXML formatom kao što bi radio s XML formatom. Da bi se razvio ARXML parser, mora se razviti XML parser te ga primijeniti na ARXML standard i vrste podataka koji se podrazumijevaju pod tim standardom.

XML parser je softverski paket ili biblioteka koja daje sučelje klijentskim aplikacijama za rad s XML datotekom. XML parser čita XML datoteku i omogućuje ostalim dijelovima programa korištenje informacija koje su pohranjene u datoteci. XML parser vrši validaciju dokumenta te provjerava je li dokument dobro formatiran [6].

U slučaju ovog diplomskog rada, kao klijentska aplikacija bi se mogao shvatiti generator testnog okruženja u koji je parser integriran. Parser obrađuje ARXML datoteke i predstavlja odgovarajuće sučelje generatoru testnog okruženja za rad s ARXML datotekama. Na slici 2.2. prikazan je odnos XML datoteke, parsera i klijentske aplikacije.



Sl. 2.2. XML parser kao sučelje klijentskoj aplikaciji za obradu XML datoteka

U ovom radu se koristi DOM parser. DOM (engl. *Document Object Model*) je objekt koji ima strukturu stabla koji sadrži sve informacije iz XML dokumenta. DOM parser stvara unutarnju strukturu u memoriji iz koje klijentske aplikacije mogu dohvaćati podatke pozivanjem metoda tog objekta [7].

Funkcionalni XML parser će imati mogućnost rukovanja s ARXML datotekama, te njihovom unutarnjom strukturom. Imat će mogućnost učitavanja datoteka i mijenjanja njenog sadržaja. Imat će mogućnost prolaženja kroz ARXML strukturu podataka i dohvaćanja tih podataka. Bit će potrebno prilagoditi rad parsera ARXML strukturi. S obzirom na to da se radi o specifičnom načinu na koji su podaci složeni u datoteci, parser će pri svojoj implementaciji biti prilagođen radu s tim podacima.

Za parsiranje ARXML datoteka nijenužno razviti XML parser. Postoje brojni provjereni XML parseri čije se metode mogu koristiti. Potrebno je razviti način korištenja XML parsera i njegovih metoda da bi bio primjenjiv na komunikacijskoj matrici.

U tom slučaju parser za komunikacijsku matricu može igrati ulogu klijentske aplikacije koja koristi XML parser za parsiranje ARXML datoteka, a parser za komunikacijsku matricu je dio klijentske aplikacije koji je u ovom slučaju generator testnog okruženja.

2.4. Postojeće rješenja parsera za komunikacijsku matricu

Postojeće rješenje parsera za komunikacijsku matricu je napisano u programskom jeziku Python. Python ima module za rad sXML datotekama koje je potrebno dodati u glavni projekt. Postojeće rješenje rastavlja ARXML datoteku na cjeline pa svaku cjelinu posebno parsira.

Generator testnog okruženja se temelji na TOM (engl. *Test Object Model*), to je visoko složeni Python objekt koji se inicijalizira po predlošcima strukturno predodređenih klasa koje se sve referenciraju jedna na drugu i tvore temeljni objekt. TOM koji svojom hijerarhijom odražava hijerarhiju ARXML-a počinje najvišim objektom u hijerarhiji koji se naziva korijenskim objektom - TOM root. TOM *root* i lokacija ARXML dokumenta se predaju funkciji za raščlanjivanje koja prolazi kroz podatke ARXML-a i grananjem ih pohranjuje u određene podobjekte TOM-a. Parser je završio s radom kada se svi relevantni podaci prenesu u TOM *root*.

Parser za parsiranje koristi LXML modul za raščlanjivanje XML datoteka koji je nadogradnja standardnog XML modula u Pythonu. LXML modul donosi znatna ubrzanja u radu. LXML modul je alat koji se koristi za povezivanje C biblioteka libxml2 i libxslt u Pythonu. LXML povezuje brzinu i XML svojstva tih biblioteka s jednostavnošću Python API-a [8]. LXML je znatno brži od standardnog pythonovog XML modula, pa se zato preferira njegovo korištenje. LXML je pod BSD licencom, a libxml2 i libxslt su pod MIT licencom. To omogućuje slobodno korištenje tih tehnologija u bilo kojem projektu.

3. PREDLOŽENORJEŠENJEPARSERAZA KOMUNIKACIJSKU MATRICU

Cilj ovog diplomskog rada je razviti parser za komunikacijsku matricu u c++ jeziku. C++ varijanta parsera bi trebala imati određene prednosti nad postojećim rješenjem u Pythonu, uključujući brzinu izvođenja parsera, te mogućnost zaštite intelektualnog vlasništva. Postojeće rješenje je korišteno kao smjernica razvoja predloženog rješenja koje će se razvijati neovisno od arhitekture postojećeg rješenja, ali s određenim zahtjevima koji se moraju zadovoljiti kao što su ispravno parsiranje ARXML datoteka te predaja podataka iz datoteka u bazu podataka gdje ti podaci pohranjuju ili nadopunjavanja od strane model parsera.

3.1. C++ programski jezik

Parser za komunikacijsku matricu je u cijelosti napisan u C++ programskom jeziku. C++ je prevedeni jezik dok je prijašnje rješenje generatora testnog okruženja razvijeno u Python2 programskom jeziku koji je interpretirani jezik. Prevedeni jezici su prevedeni direktno u strojni jezik računala programom koji se naziva prevoditelj (engl. *compiler*). To rezultira veoma brzim kodom, osobito ako je prevoditelj efikasan u optimiziranju. Međutim, rezultirajući kod se kod pokretanja na drugom operacijskom sustavu mora ponovno prevesti, što usporava proces premještanja softvera na druge operacijske sustave.

Interpretirani jezici su čitani od strane programa koji se zove interpreter te su izvršeni od strane tog programa. Interpretirani jezici su prenosivi kao što je prenosiv njihov prenositelj. Interpretirani jezici su općenito sporiji od prevedenih jezika [9].

Prednost kod brzine izvođenja c++ nad Python2 je jedan od razloga korištenja c++ za razvoj generator testnog okruženja ili točnije u ovom radu specifično, parsera za komunikacijsku matricu. Druga velika prednost je mogućnost sakrivanja intelektualnog vlasništva pri predaji programa klijentima, pa se projekt napisan u c++ može komercijalizirati bez da se otkrije izvorni kod.

C++ omogućuje programiranje objektno orijentiranom paradigmom, koji se koristio pri razvoju parsera za komunikacijsku matricu. Parser je napisan kao klasa s funkcijama, te funkcionalnosti omogućuje objekt klase parsera koji se prethodno mora stvoriti.

Korištena je C++ 17 inačica koja sadrži *filesystem* biblioteku koja se koristi pri davanju ulaznih podataka parseru.

Okruženje u kojem je razvijen parser je Microsoft Visual Studio. To je razvojno okruženje koje ima podršku za c++ te je odabran zbog svoje efikasnosti i jednostavnosti korištenja.

3.2. *Tinyxml2* biblioteka

U predloženom rješenju za parser komunikacijske matrice je korišten tinyxml2, nadogradnja na *tinyxml* [10]. Kao alat za parsiranje XML ili ARXML dokumenta, tinyxml2 se odabrao zbog brojnih mogućnosti koje ima za rad nad datotekama xml formata u koje pripada i ARXML. Ima prednosti nad tinymxl-om jer koristi manje memorije, brži je, te koristi puno manje memorijskih alokacija. Radi se o alatu sa Zlib licencom. To je besplatna softverska licenca pa se može koristiti u open source ili komercijalnom kodu. Tinyxml2 može parsirati XML datoteku te na osnovu te datoteke izgraditi DOM koji je moguće čitati, mijenjati i snimiti.Korištenje DOM-a podrazumijeva parsiranje XML podataka u C++ objekte koji se mogu pretraživati i manipulirati, te zapisivati na disk ili standardni izlaz. Moguće je i kreirati XML datoteku sC++ objektima, te zapisivati ga u disk ili standardni izlaz.

Tinyxml2 je jednostavne građe, sastoji se od jedne datoteke zaglavlja, te jedne cpp datoteke. Za ispravan rad tinyxml2 alata je potrebno samo dodati te datoteke u projekt.

Rezultat parsiranja XML datoteke postaje objekt te se iz tog objekta mogu izvući čvorovi, te čvorovi tih čvorova. To omogućuje jednostavan i brzo pretraživanje kroz XML strukturu, te pronalaženje podataka koji su relevantni za određeni projekt.

U ovom rješenju korištene su funkcije tinyxml2 radi pohrane XML datoteke u DOM, te pretraživanje kroz dobiveni objekt i pronalaženje, te dohvaćanje podataka koji je potrebno dalje spremiti u bazu podataka.

Za pohranu datoteke u memoriju se koristi `LoadFile` funkcija, čiji je rezultat stablo čvorova. Za prolazak kroz ARXML datoteku su korištene `FirstChildElement` i `NextSiblingElement` funkcije, ovisno ide li se u dubinu ili širinu stabla. `FirstChildElement` se koristi za dohvaćanje prvog čvora djeteta nekog čvora dok se `NextSiblingElement` koristi za dohvaćanje idućeg čvora nakon trenutnog na istoj razini. Za dohvaćanje podataka u čvorovima koji imaju podatke su korištene `Name` i `GetText` funkcije. *`Name`* je potreban za dohvaćanje imena "taga", imena vrste podatka koji se nalazi unutar šiljaste zagrade (<>), a `GetText` za dohvaćanje vrijednosti podataka koji se nalaze izvan zagrada.

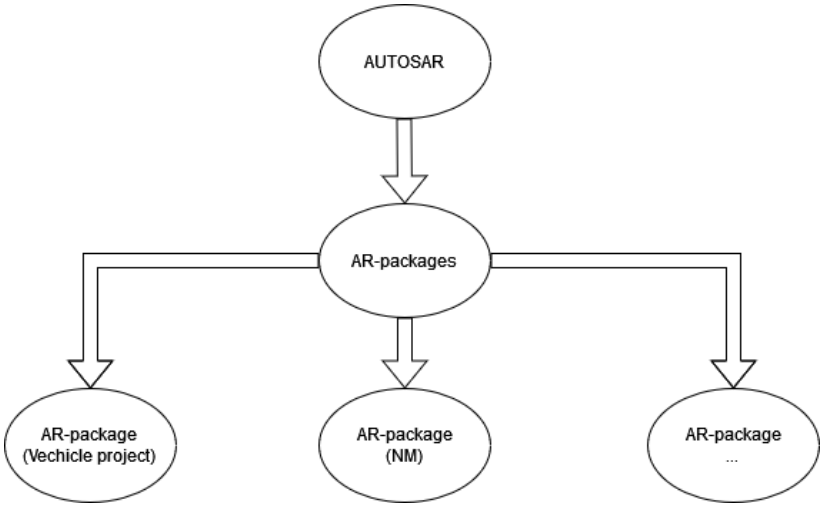
Tinyxml2 se pokazao kao dobro rješenje za pretraživanje po ARXML datoteci, te dohvaćanju podataka iz pojedinih čvorova. Pokazao se kao brz i efikasan alat, te pogodan za integrirati u rad Parsera za komunikacijsku matricu. Uz to licenca tinyxml2 dopušta korištenje u komercijalne svrhe, tako da ga to čini idealnim kandidatom za bilo kakve projekte kojima je potreban rad s XML ili ARXML datotekama.

3.3. Predloženo rješenje za parser za komunikacijsku matricu

ARXML datoteke koje se parsiraju su složene građe. Jedna takva datoteka se sastoji od jednog root čvora koji pod sobom sadrži više raznih paketa, od kojih svaki sadrži različite podatke. Svaki paket je drugačije arhitekture čvorova, pa postoje paketi jednostavne i paketi složenije arhitekture. Zbog te različitosti je potrebno koristiti drugačiji pristup za pojedine pakete iako je moguće više paketa parsirati na isti način.

Predloženo rješenje parsera koristi tinyxml2 funkcije radi pronalaženja podataka u složenoj strukturi ARXML datoteke. Koristi se objektno orijentirani pristup koji C++ podržava pa se rješenje parsera sastoji od jedne klase. Klasa sadrži samo funkcije bez vlastitih atributa, takozvana "utility class". Funkcije u klasi su uglavnom zadužene za parsiranje određenog dijela datoteke. Uglavnom je jedna funkcija zadužena za prolazak kroz sve podčvorove te pronalaženje i dohvaćanje podataka u sklopu jednog specifičnog paketa. Jedna funkcija može parsirati dva paketa ako dva paketa imaju istovjetnu građu. Male razlike u rasporedu čvorova između dva paketa čine ponovno korištenje funkcija za parsiranje teškim. Zbog toga je korištena veća

količina funkcija za pojedinačne pakete. Neki paketi u datotekama ne sadrže nikakve podatke osim imena paketa. Ti paketi su ignorirani, te nisu napravljene funkcije za njih jer je nepoznata arhitektura kakvu bi imali da su podaci prisutni. U dijagramu prikazanom na slici 3.1. je prikazana građa ARXML datoteke.

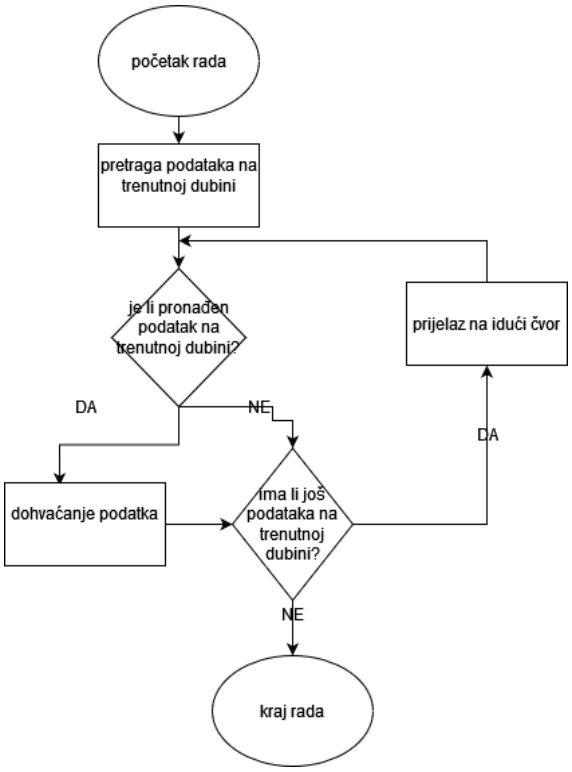


Sl. 3.1. Dijagram strukture jedne ARXML datoteke

AUTOSAR predstavlja korijenski čvor koji se dobije tinyxml2 `LoadFile` funkcijom. Svi čvorovi ispod čvora AUTOSAR koji su spojeni strelicom su njegova djeca. *AR-packages* predstavlja prvi čvor dijete čvora AUTOSAR, te sadrži sve pakete u ARXML datoteci, dok su prikazana 3 čvora ispod *AR-packagesa* prva djeca *AR-packagesa*. AR-package ima ime, te su ta imena na dijagramu napisana u zagradama ispod naziva *AR-package*. Primjeri imena na dijagramu su *Vehicle project*, *NM* itd. Strelicom je prikazana ovisnost čvorova tako što roditelj strelicom pokazuje na dijete. Glavna funkcija parsera, u kojoj se parsiraju sve ARXML datoteke, se poziva tako što se prvo definira *`XMLDocument`* tip podatka te se na tom tipu podatka poziva `LoadFile` funkcija s putanjom datoteke koja se parsira kao argumentom. S obzirom na to da se koristi C++ 17, korištene su putanje iz *filesystem* biblioteke koju podržava C++ 17. Parser kao ulaz prima proizvoljan broj putanja datoteka da bi se mogao parsirati proizvoljan broj datoteka, ovisno o tome koliko mu se predaje tamo gdje ga se poziva. Putanje se predaju funkciju u obliku

vektora putanja te ta funkcija u petlji, za svaku putanju pozove glavnu funkciju parsera. U glavnoj funkciji se poziva `LoadFile` funkcija koja kao argument prima putanju. Potrebno je putanju koja se kao argument predaje `LoadFile` funkciji pretvoriti u tip podataka pokazivač na `const char` jer `LoadFile` radi samo s tim tipom podataka. To se veoma jednostavno postiže pozivanjem funkcija pretvorba iz *filesystem* biblioteke na putanju koja se predaje `LoadFile` funkciji.

Opisanim procesom se dobije korijenski čvor preko kojeg se dalje dohvaćaju njegovi podčvorovi. Podčvor od bilo kojeg čvora se dohvaća `FirstChildElement` funkcijom. Tako se i dohvaća *AR-Packages* čvor koji je dijete AUTOSAR čvora, te se od *AR-Packages* čvora dalje dohvaćaju sva djeca *AR-Packages* čvora. Dijete AR-Packages čvora je *AR-Package* i svaki *AR-Package* ima svoje ime. U tom dijelu parsera gdje se dohvaćaju svi pojedinačni paketi, korištena je while petlja u kojoj se prelazi od paketa do paketa na istoj dubinskoj razini sa *NextSiblingElement* funkcijom. To se u petlji radi dok god postoji čvor iza čvora na kojem se petlja trenutno nalazi. Kada petlja dođe do zadnjeg čvora, petlja završava s radom, a i samim time parser jer je većina rada parsera sadržana u toj petlji. Na slici 3.2. je prikazan dijagram algoritma pretrage i dohvaćanja podataka na istoj dubinskoj razini. Taj algoritam se izvršava na svakom čvoru u ARXML datoteci. Završava kada parser pronađe kraj čvora i ponovno počinje s radom svaki put kada parser prijeđe na novi čvor.



Sl. 3.2. Dijagram rada algoritma za pretragu i dohvaćanje podataka

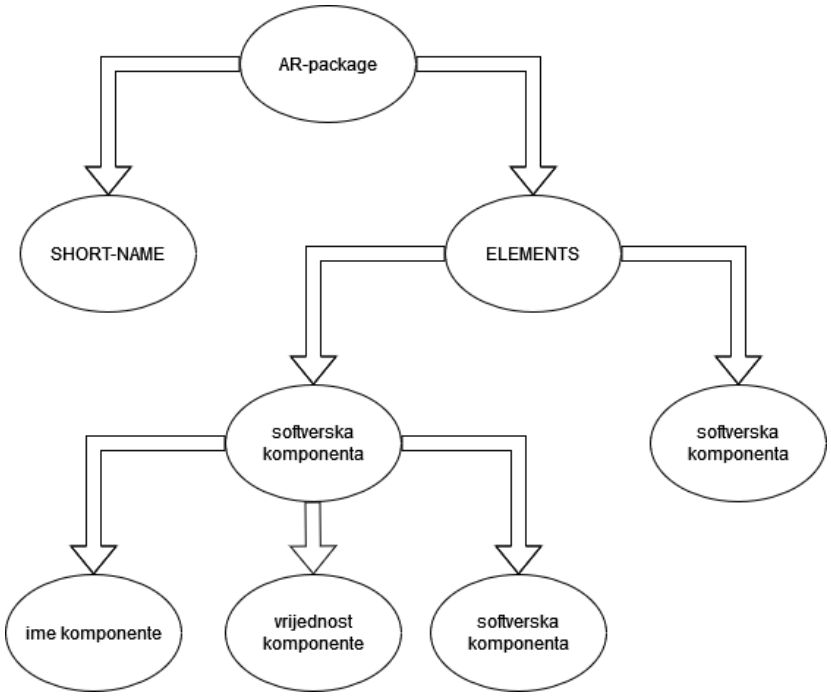
Na svakom čvoru petlje, se dohvaća prvi podčvor koji je u svakom slučaju dohvaćenog čvora ime paketa. Kada se dohvati ime, vrši se provjera postoji li funkcija za taj specifični paket, jer kako je prije spomenuto neki su paketi ignorirani zbog nedostatka podataka, a i samim time nepoznatosti njihove strukture. Ta provjera se vrši `if else` grananjem u kojem za svako ime paketa za koji postoji funkcija parsiranja u uvjetu za `else if` grananje vrši funkcija usporedbe znakovnih nizova. Usporedba znakovnih nizova se vrši između imena paketa na trenutnom čvoru, koji je dohvaćen od strane `GetText` funkcije te znakovnim nizom jednog od potencijalnih imena paketa koji sadrže funkcije za njihovo parsiranje. U slučaju da je rezultat provjere u `if` uvjetu jednak 0, pokreće se funkcija za parsiranje tog specifičnog paketa. Funkcija za usporedbu znakovnih nizova (engl. *string compare*) se koristi za usporedbu dva znakovna niza. Ako su dva znakovna niza jednaka onda usporedba vraća broj 0, inače vraća vrijednost različitu od 0. Funkcija uspoređuje znakovne nizove znak po znak koristeći ASCII vrijednost znakova. Uspoređivanje prestaje kada se dođe do kraj znakovnog niza ili se znakovi koji se uspoređuju ne podudaraju [11]. Usporedba znakovnih nizova je bitna za rad parsera jer se

pomoću nje vrši cjelokupno grananje pozivanja funkcija za parsiranje. Na slici 3.3. je prikazano korištenje funkcije za usporedbu dva znakovna niza. Funkcija se zove *strcmp*.

```
strcmp(“a”, “a”); // vraća 0 jer su ASCII vrijednosti od “a” i “a” iste
strcmp(“a”, “b”); // vraća -1 jer je ASCII vrijednost od  “a” manja od ASCII vrijednosti od “b”
strcmp(“a”, “c”); // vraća -1 jer je ASCII vrijednost od  “a” manja od ASCII vrijednosti od  “c”
strcmp(“z”, “d”); // vraća 1 jer je ASCII vrijednost od  “z” veća od ASCII vrijednosti od  “d”
strcmp(“abc”, “abe”); // vraća -1 jer je ASCII vrijednost od  “c” manja od ASCII vrijednosti od
“e”
strcmp(“apples”, “apple”); // vraća 1 jer je ASCII vrijednost od  “s” veća od ASCII vrijednosti
od  “\0
```

Sl. 3.3. Primjer rada `strcmp` funkcije

Pri grananju se pozivaju funkcije za parsiranje. Svaka funkcija parsira zasebni dio ARXML datoteke, ali sve funkcije rade na istom principu. Svaka funkcija mora proći kroz sve čvorove paketa kojeg parsira te dohvatiti, spremiti, te proslijediti podatke bazi podataka. Da bi se išta od toga moglo izvršiti, potrebno je proći kroz stablo. To se radi slično kao u dijelu parsera u kojem se pozivaju pojedinačne funkcije parsera, pozivanjem jedne `while` petlje koja obuhvaća cijeli paket. U svakom paketu se iza čvora imena, koji se naziva “SHORT-NAME”, na istoj razini kao i taj čvor nalazi čvor “ELEMENTS”. Taj čvor sadrži sve ostale podatke toga paketa. Na slici 3.4. je prikazan dijagram strukture jednog paketa u ARXML datoteci.

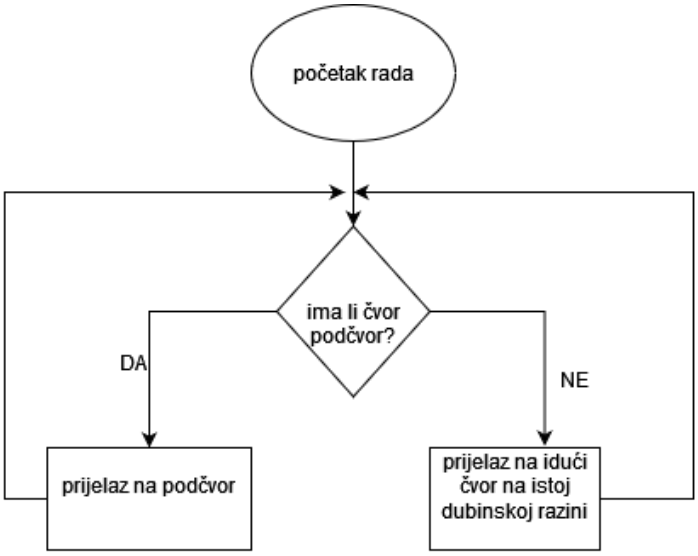


Sl. 3.4. Dijagram strukture paketa u ARXML datoteci

"ELEMENTS" čvor sadrži veliku količinu softverskih komponenti od kojih svaka sadrži ime ili nekakav identifikator koji je unikatan u kontekstu ARXML datoteke, određene vrijednosti koje mogu biti broj, string ili bool vrijednost, te često još i druge softverske komponente od kojih svaka sadrži vlastita imena, vrijednost i moguće softverske komponente. Sve softverske komponente te njihove članove s njihovim potkomponentama je potrebno proći radi dohvaćanja i spremanja podataka u njima.

Već je spomenuto korištenje `while` petlje koja obuhvaća cijeli paket. Ona se koristi kada je trenutno dohvaćeni čvor na "ELEMENTS" čvoru zbog spomenutog sadržavanja svih ostalih podataka pod "ELEMENTS" čvorom. While petlja prolazi kroz svaki čvor iste razine pod "ELEMENTS" čvorom sa `NextSiblingElement` funkcijom, te na svakom čvoru provjerava ima li taj čvor djece. Ako čvor nema djece onda je u većini slučajeva znak da se radi o čvoru koji sadrži podatke koji trebaju biti spremljeni te poslani bazi podataka. Postoji iznimka toga kada se radi o čvorovima sa znakom "\" na kraju čvora. Ti čvorovi ne sadržavaju podatke niti imaju djece pa je potrebno voditi računa o njima jer potencijalno mogu prekinuti rad parsera. Provjera

prisutnosti djece se vrši `if` grananjem gdje se pomoću `FirstChildElement` funkcije na trenutnom čvoru provjerava ima li taj čvor dijete. U slučaju da nema vrši se dohvaćanje i spremanje. U slučaju da ima dijete, stvara se novi čvor na djetetu tog čvora te se stvara nova while petlja koja obuhvaća sve čvorove iste razine pod čvorom za koji se ustvrdilo da ima dijete. Za svaki od tih čvorova se opet vrši provjera ima li djecu, te se opet u slučaju da nema djecu vrši dohvaćanje i spremanje, a u slučaju da ima djecu se opet stvara novi čvor na čvoru djeteta s pripadnom `while` petljom koja prolazi kroz sve čvorove iste razine. To se rekurzivno izvršava dok parser ne prođe kroz sve čvorove koji se nalaze u paketu, te tako pokrije sve moguće podatke koji se nalaze u tom specifičnom paketu ARXML datoteke. Na slici 3.5. je prikazan algoritam dubinskog prelaženja čvorova. Algoritam se izvršava u petlji dok ne naiđe na posljednji čvor u ARXML datoteci



Sl. 3.5. Dijagram rekurzivnog pretraživanja parsera po dubini

Paketi jednostavnije strukture će trebati manje ugniježđenih `while` petlji za prolazak kroz sve podatke dok će složenije strukture zahtijevati veću razinu ugniježđenosti radi dohvaćanja svih podataka. Uspješnost prolaska kroz cijeli paket se jednostavno provjeri tako što parser prođe kroz paket bez izbacivanja greške. Čim parser zapne na nekom čvoru, parser ne može dalje prelaziti po čvorovima i rad programa prestane, te se na terminalu izbacuje greška. Uspješnost prolaska parsera kroz paket se može utvrditi i ispisom, ali o tome će poslije biti riječ.

Uz prolazak kroz paket, potrebno je i dohvaćanje podataka pod čvorovima. Na slici 3.6. je prikazan čvor bez djece.

```
<IME>PODATAK</IME>
```

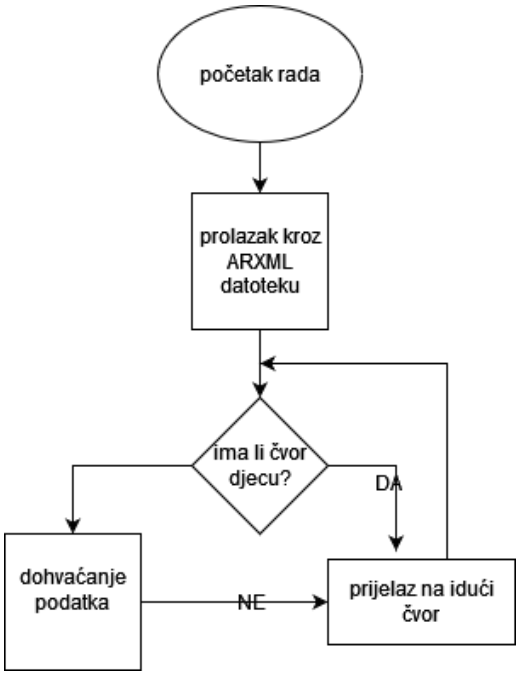
Sl. 3.6. ARXML podatak unutar sva *tag*-a

Takvi čvorovi imaju barem jedan čvor roditelj. Na slici 3.7. je prikazan čvor s podacima koji je ugniježđen pod drugim čvorom.

```
<ČVOR>
    <PODČVOR>
        <IME>podatak</IME>
    </PODČVOR>
</ČVOR>
```

Sl. 3.7. ARXML podatak unutar dva *tag*-a kao čvor dijete jednog čvora

Podatke između zagrada je potrebno dohvatiti i spremiti. Dohvaćanje postaje trivijalan zadatak ako je trenutni čvor s podatkom. Nakon izvršenja provjere prisutnosti djece čvora što je ujedno u većini slučajeva i provjera prisutnosti podataka poziva se tinyxml2 funkcija `GetText` na trenutnom čvoru. Dohvaćanje elemenata se može provjeriti ispisivanjem poziva `GetText` funkcije na trenutnom čvoru. Na slici 3.8. je prikazan algoritam dohvaćanja podataka iz datoteke. Podaci se dohvaćaju po algoritmu u petlji dok parser ne prijeđe cijelu datoteku.



Sl. 3.8. Algoritam dohvaćanja podataka

Nakon dohvaćanja podataka, potrebno ih je spremiti u odgovarajuće strukture podataka koje se pohranjuju u radnu memoriju računala, koje se mogu proslijediti bazi podataka na daljnju pohranu. Kao što je gore spomenuto, u datoteci se pod svakim paketom nalazi veća količina softverskih komponenata od kojih svaka ima ime i vrijednosti. Na slici 3.9. je prikazan jednostavni tip komponente bez dodatnih potkomponenata koje se nalaze u njoj.

```
<komponenta>
    <ime>podatak</ime>
    <vrijednost>podatak</vrijednost>
    <vrijednost>podatak</vrijednost>
</komponenta>
```

Sl. 3.9. prikaz jedne softverske komponente u ARXML datoteci

Sve podatke iz komponente je potrebno spremiti u strukturu podataka koja će činiti istu takvu cjelinu. Ta struktura podataka isto mora imati mogućnost dohvaćanja tih podataka na temelju njihove vrste, što u slici plavim fontom označava *tag*, koji predstavlja ime vrijednosti. Podatkovne strukture koje se koriste u parseru su stl mape. Mape su asocijativni kontejneri koji skladište elemente na mapiran način. Svaki element ima ključnu vrijednost i mapiranu vrijednost,

što je poznato kao *key-value* princip. Više mapiranih vrijednosti ne može imati isti ključ [12]. Tablica 3.1. predstavlja izgled građe jedne stl mape.

Tablica 3.1. Prikaz građe stl mape

Mapa1	
Ključ1	Vrijednost1
Ključ2	Vrijednost2
Ključ3	Vrijednost3

Mape se mogu koristiti uvrštavanjem standardne biblioteke za mape u C++ projekt. Nad mapama se mogu izvršavati određene funkcije. Za parser je bitna funkcija `pair insert`. Ta funkcija dodaje par vrijednosti u mapu. Jedna vrijednost je ključ, a druga vrijednost je vrijednost pod tim ključem.

```
map<int,int> m{{1,2} , {2,3} , {3,4} };

m.insert( pair<int,int> (4,5));
```

Sl. 3.10. Primjer korištenja pair insert funkcije

Na slici 3.10. se vidi kako prva linija koda definira mapu u kojoj su ključ i vrijednost pod ključem prirodni brojevi, te ima tri članova: vrijednost 2 s ključem 1, vrijednost 3 s ključem 2 i vrijednost 4 s ključem 3. U drugoj liniji koda se poziva funkcija *insert* na mapi, čime se dodaje još jedan član u mapu; vrijednost 5 s ključem 4. U slučaju podataka iz ARXML datoteke, tip podataka koji se navodi u `insert` funkciji će uvijek biti tipa string, neovisno je li podatak u ARXML datoteci prirodan broj, realan broj, znakovni niz ili logički tip podataka. Argumenti funkcije `insert` će na kraju biti dva podatka tipa znakovnog niza, jedan je predan kao funkcija

`Name` pozvana na trenutnom čvoru, a druga je funkcija `GetText` pozvana na trenutnom čvoru. Prvi podatak predstavlja "tag" ili vrijednost unutar zagrada, a drugi podatak predstavlja vrijednost između zagrada.

Tijekom prolaska kroz čvorove, parser sprema podatke u mape. Kada se dođe do posljednjeg elementa jedne specifične softverske komponente, spremanje podataka u jednu mapu je gotovo te se ta mapa, popunjena sa svim bitnim podacima predaje bazi podataka ili u multidimenzionalnu mapu. U multidimenzionalnoj mapi će biti spremljene sve mape koje nastaju prolaskom kroz stablo, svaka sa svojim jedinstvenim ključem. Multidimenzionalna mapa će biti pogodna za dohvaćanje svih njenih podataka kroz dvije petlje, te dobra struktura podataka za provjeravanje ispravnosti spremljenih podataka ispisom. Mapa se sprema u multidimenzionalnu mapu kao bilo koja druga vrijednost pod nekim ključem. Za ključ se uzima vrijednost u mapi koja je jedinstvena u toj mapi. Najpogodnija vrijednost za ključ neke mape u multidimenzionalnoj mapi je ime softverske komponente čije se vrijednosti spremaju u mapu, te koju ta mapa i predstavlja kao struktura podataka ili bilo koji jedinstveni identifikator.

```
map<int, map<int, int>> m;
```

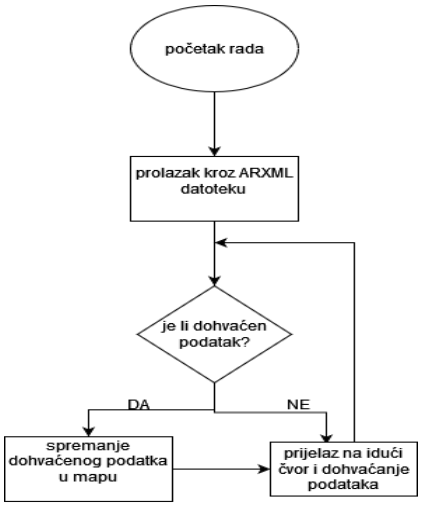
Sl. 3.11. Korištenje dvodimenzionalne mape u C++

Na slici 3.11. je prikazano definiranje multidimenzionalne maps s dvije dimenzije. Kada se stvori multidimenzionalna mapa, s mapama koje sadrže podatke iz ARXML datoteka, moguće je iz te multidimenzionalne mape slati pojedinačne mape u bazu podataka. Mogu se podaci slati i tako da se pojedinačne mape šalju u bazu podataka odmah nakon popunjavanja, bez spremanja u multidimenzionalnu mapu. Za obje stvari su potrebne dvije petlje, jedna radi dohvaćanja, pojedinih mapa, a druga radi dohvaćanja svakog elementa u mapi. To se može obaviti pomoću iteratora. U tom slučaju su potrebne funkcije `begin` i `end` koje pozivanjem na mapi daju početak i kraj mape. U tablici 3.2. je prikazana građa jedne dvodimenzionalne mape. U takve mape se spremaju sve pojedinačne mape s podacima i ključevima.

Tablica 3.2. Prikaz građe dvodimenzionalne mape

dvodimenzionalna mapa		
Ključ mape1	Mapa1	
	Ključ1	Vrijednost1
	Ključ2	Vrijednost2
	Ključ3	Vrijednost3
Ključ mape2	Mapa2	
	Ključ1	Vrijednost1
	Ključ2	Vrijednost2
	Ključ3	Vrijednost3

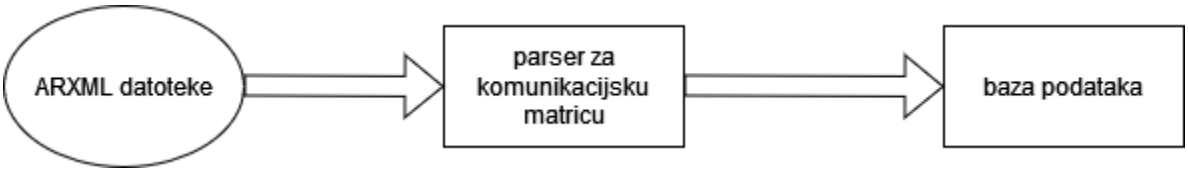
Cijeli proces prolaženja kroz ARXML datoteku, dohvaćanja podataka i spremanja u mape se može opisati jednim dijagramom. Na slici 3.12. je prikazan dijagram prolaska kroz datoteku, dohvaćanja i spremanja podataka. Podaci se dohvaćaju i spremaju u petlji dok parser nije došao do posljednjeg podatka u ARXML datoteci.



Sl. 3.12. Dijagram rada parsera za komunikacijsku matricu

3.4. Razmjena podataka između parsera i baze podataka

Podaci koje parser za komunikacijsku matricu izdvoji iz ARXML datoteka trebaju u bazi podataka zamijeniti njihove komplementarne podatke koji su tamo spremljeni od strane model parsera. Model parser u bazu podataka sprema osnovne podatke koje je potrebno nadopuniti ili prepisati. Potrebno je pronaći sve softverske komponente iz model parsera koje se imenom podudaraju sa softverskim komponentama iz parsera za komunikacijsku matricu. Nakon pronalaska je potrebno prepisati sve vrijednosti iz model parsera s vrijednostima koje je parser za komunikacijsku matricu prikupio te još nadopuniti vrijednosti model parsera s vrijednostima koje model parseru nedostaju. Na slici 3.13. je prikazan dijagram komunikacije parsera za komunikacijsku matricu s bazom podataka. ARXML datoteke su prikazane kao ulaz u parser za komunikacijsku matricu.



Sl. 3.13. Dijagram komunikacije parsera i baze podataka

Parser koristi funkcije za rad s bazom podataka kao što su funkcije za pretragu baze podataka i za spremanje podataka u bazu podataka. Radi se o dvjema funkcijama koje kao argument primaju mapu od parsera za komunikacijsku matricu, te pretražuju bazu podataka za mjestom gdje će smjestiti vrijednosti te mape, odnosno njima prepisati postojeće vrijednosti u bazi podataka. Navedene funkcije su zajedno sa bazom podataka razvijene u drugom radu. Jedna funkcija kao argument prima putanju do određene softverske komponente, koja obično završava njenim imenom, i mapu s podacima te softverske komponente. Druga funkcija prima samo ime softverske komponente, koje se tretira kao ključ za pronalazak komponente i mapu koja sadrži vrijednosti te softverske komponente. Funkcije prepišu na odgovarajuća mjesta u bazi postojeće podatke koje mogu, te dodaju podatke koje model parser nije tamo dodao. Parser za komunikacijsku matricu koristi funkciju koja za argument prima ključ, iako postoji podrška i za

drugu funkciju koja prima putanju, s obzirom na to da se u funkcijama za parsiranje paketa pamte putanje do pojedinih elemenata. To se radi tako što se preko tinyxml2 funkcije `Name` na svakom čvoru koji je potrebno prijeći na putu do pojedinog elementa spremi ime čvora u vektor. To se radi sve do vrijednosti imena pojedine softverske komponente koja predstavlja kraj putanje, te se sprema funkcijom `GetText`. Svi pojedini vektori koji sadržavaju putanje su spremljeni u dvodimenzionalni vektor koji se sastoji od tih vektora. Taj vektor sadržava sve putanje softverskih elemenata u ARXML datoteci i koristi se radi ispisa i provjere ispravnosti dohvaćenih putanja.

Dohvaćanje, spremanje podataka u mape i bazu podataka odvija se u glavnoj funkciji parsera. Uz glavnu funkciju parsera je potrebno prije pozivanja funkcije preko objekta od parsera pozvati funkciju za učitavanje baze podataka preko objekta za rad s bazom podataka, koja će pripremiti datoteku koja čini tu bazu. Datoteka sadrži podatke koje je model parser dohvatio i spremio u bazu podataka iz ARXML datoteka koji su bitni za njega. Datoteka baze podataka se radi ispravnog rada nalazi u određenom direktoriju na disku. Baza podataka ne zahtijeva internetsku vezu za rad, sve se obavlja interno na računalu. Nakon učitavanja i parsiranja, potrebno je pozvati funkciju za spremanje baze podataka preko istog objekta za rad s bazom podataka. Ta funkcija stvori novu datoteku u direktoriju za bazu podataka koja sadrži nadopunjene vrijednosti uz originalnu datoteku s vrijednostima model parsera. To omogućuje ručno uspoređivanje dviju datoteka s ciljem provjere pronalaženja te spremanja podataka u bazu. Spremanjem baze podataka završava rad parsera za komunikacijsku matricu. Parser za komunikacijsku matricu je još i potpuno integriran u rad baze podataka, što cijeli proces parsiranja i spremanja u bazu podataka čini jednostavnim i neprekinutim procesom.

4.TESTIRANJEPREDLOŽENOGRJEŠENJAPARSERAKOMUNIKACIJSKEMATRICE

S obzirom na to da se radi o novom rješenju parsera za komunikacijsku matricu s fokusom na ARXML, bilo je potrebno parser podvrgnuti testiranju. Potrebno je ispitati ispravnost rada parsera i njegovu brzinu izvođenja. To se postiže pomoću sljedeća dva testa:

- provjera ispravnosti podataka ispisanih u bazu podataka, i
- mjerenja vremena izvođenja rada parsera.

Sva testiranja za parser su obavljena na istom skupu podataka u obliku tri ARXML datoteke. Tablica 4.1. prikazuje točne specifikacije datoteka. Testovi su izvršeni na računalu koje ima procesor od Intela radnog takta od 3.5 GHz.

Tablica 4.1. specifikacije ulaznih ARXML datoteka

Ime datoteke	Memorijska veličina datoteke (MB)
TTADrive	0,16
TTADrive_Misc	0,33
TTADrive_Object	0,21

4.1. Opis testova

Provjera ispravnosti podataka koji su zapisani u bazu podataka odrađena je ručnom provjerom podataka koje je parser dohvatio iz ARXML datoteke te spremio u bazu podataka. Odrađene su i provjere ispravnosti prijelaza parsera kroz ARXML datoteke, dohvaćanja podataka, te spremanja podataka u privremene mape. Svi ti testovi su obavljeni jednostavnim ispisom na terminal te ručnom usporedbom podataka ispisanih na terminal s podacima iz ARXML datoteka. Napravljen je i pomoćni test dohvaćanja potrebnih podataka u bazi podataka, gdje će se smjestiti podaci iz komunikacijske matrice, ispisom riječi na terminal svaki put kada je pronađen element.

Provjera brzine rada parsera je izvršena mjerenjem vremena koje potrebno parseru da obavi svoj zadatak. Obavljena su tri eksperimenta:

- mjerenje vremena koje je potrebno parseru da spremi datoteku TTADrive u bazu podataka
- mjerenje vremena koje je potrebno parseru da spremi datoteku TTADrive_Misc u bazu podataka
- mjerenje vremena koje je potrebno parseru da spremi datoteku TTADrive_Object u bazu podataka

Nakon provedbe eksperimenta je izmjereno vrijeme predloženog rješenja parsera uspoređeno s vremenom izvođenja postojećeg rješenja parsera. Predloženo rješenje je razvijeno u Microsoft Visual Studiju u C++ programskom jeziku, pa je mjerenje vremena obavljeno automatski pri izvođenju samog programa, uz prikaz korištene memorije.

4.2. Rezultati ispravnosti zapisivanja podataka u bazu podataka

Kao što je objašnjeno u prijašnjem potpoglavlju, provjera ispravnosti podataka je odrađena ručnom usporedbom baze podataka prije i nakon spremanja podataka iz komunikacijske matrice. Nakon svakog spremanja podataka se izradi nova tekstualna datoteka s nadopunjenim podacima ispod tekstualne datoteke koja sadrži samo podatke od model parsera.

Podaci iz model parsera je potrebno prepisati i nadopuniti s podacima koje parser za komunikacijsku matricu dohvaća iz ARXML datoteke. To se provjerava ručnom usporedbom tekstualne datoteke koja sadrži damo podatke od model parsera, dakle datoteke prije spremanja podataka iz ARXML datoteka u bazu podataka s tekstualnom datotekom nastalom nakon spremanja podataka iz ARXML datoteka u bazu podataka.

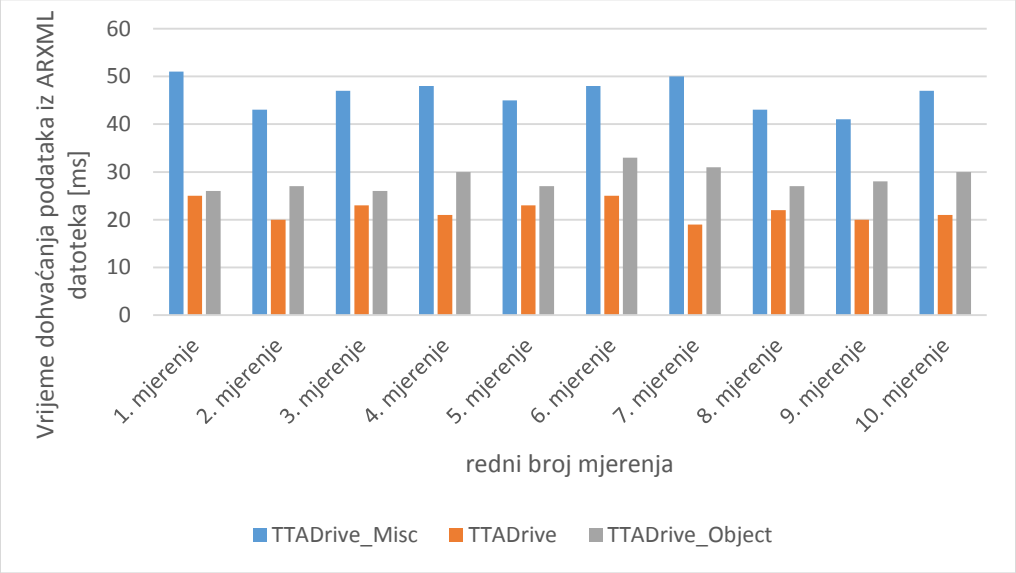
Ručnom usporedbom je utvrđeno da su podaci od model parsera uspješno prepisani i nadopunjeni s podacima iz ARXML datoteka koje je dohvatio parser za komunikacijsku matricu.

Parser prenosi podatke u bazu podataka Svi ti podaci su još ručno uspoređeni s podacima iz ARXML datoteka, te je utvrđeno da su podaci u bazi podataka zbilja podaci iz ARXML datoteke

koje je bilo potrebno spremiti u bazu podataka. Ovi testovi pokazuju da parser ispravno dohvaća podatke iz ARXML datoteka te ih ispravno prenosi u bazu podataka.

4.3. Rezultati brzine rada parsera za komunikacijsku matricu

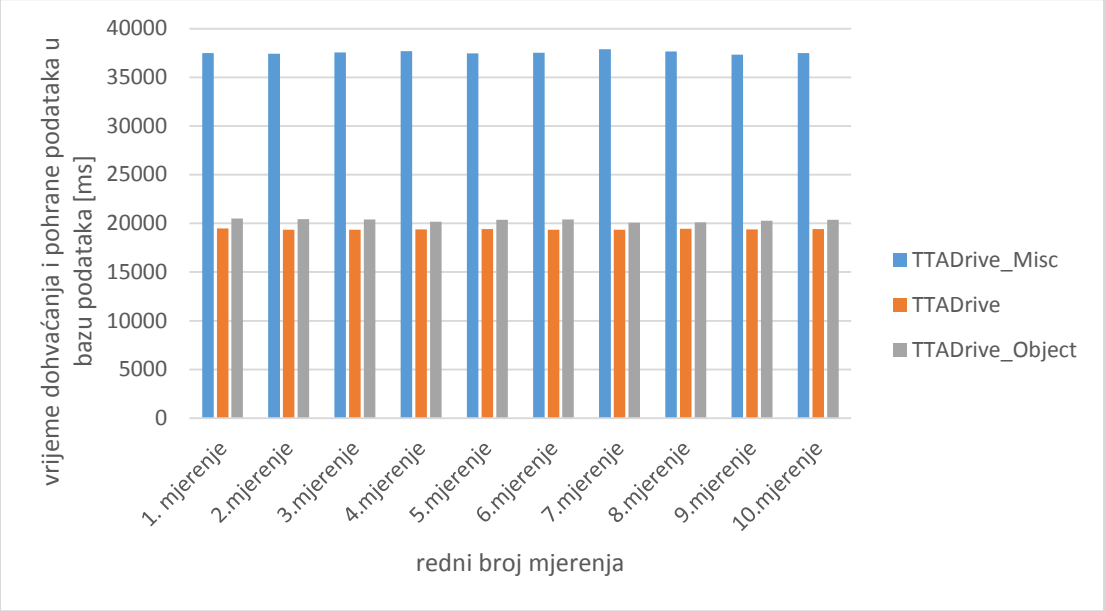
Kod testiranja brzine izvođenja parsera provedeno je dva tipa testa. Za svaku ulaznu datoteku je izvršeno 10 testova mjerenja za dohvaćanje i pohranu podataka u bazu podataka te usporedba istog procesa s prijašnjim rješenjem u Python2. Za svaku ulaznu datoteku je izvršeno i 10 testova mjerenja samo za dohvaćanje podataka iz ARXML datoteka bez spremanja u bazu podataka. To je prikazano u grafikonu na slici 4.1.



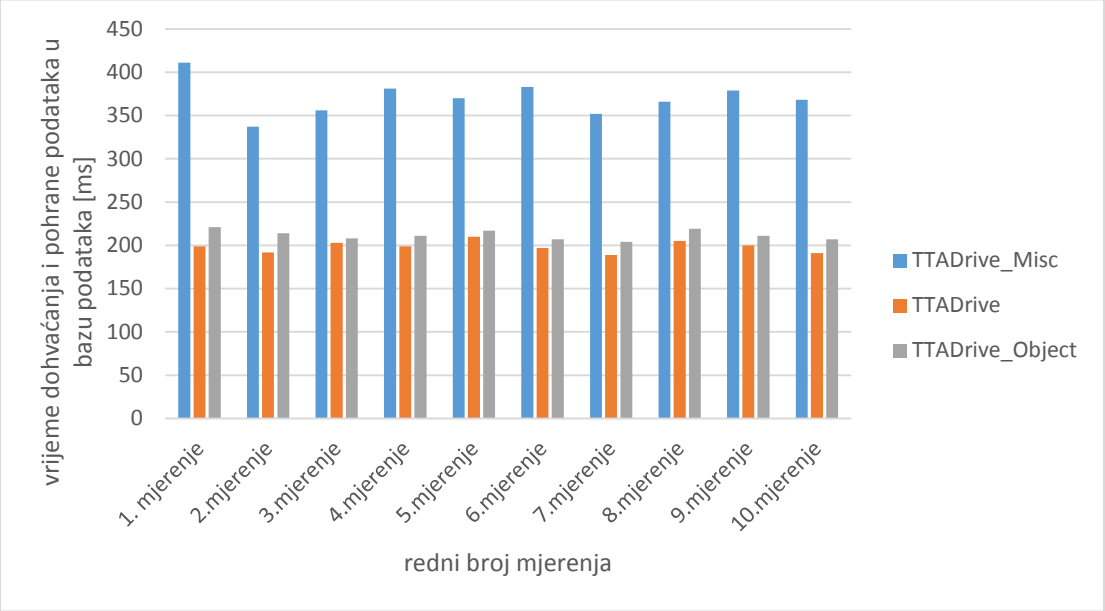
Sl 4.1. Izmjereno vrijeme dohvaćanja podataka iz ARXML datoteka u C++ rješenju

Na slici 4.2 a) je prikazano izmjereno vrijeme potrebno za dohvaćanje podataka iz ARXML datoteka TTADrive_Misc, TTADrive, te TTADrive_Object i spremanja podataka u bazu podataka u postojećem rješenju u Python2, a na slici 4.2 b) je prikazano izmjereno vrijeme potrebno za dohvaćanje podataka iz ARXML datoteka TTADrive_Misc, TTADrive, te TTADrive_Object i spremanja podataka u bazu podataka u predloženom rješenju u C++.

Usporedbom grafikona se vidi kako se vrijeme za dohvaćanje i spremanje podataka značajno smanjilo u predloženom rješenju u C++.



(a)



(b)

Sl. 4.2. Usporedba vremena potrebnog za parsiranje i pohranu podataka za svaku od tri ARXML datoteka u (a) postojećem rješenju u Pythonu i (b) predloženom rješenju u C++

Tablica 4.2. Usporedba srednjeg vremena i medijana svih 10 testova za svaku od 3 ARXML datoteke u Python2 i C++

Datoteka	Srednje vrijeme potrebno za dohvaćanje podataka iz ARXML datoteke i spremanje podataka u bazu podataka [ms]		Medijan vremena potrebno za dohvaćanje podataka iz ARXML datoteke i spremanje podataka u bazu podataka [ms]	
	Predloženo rješenje u C++	Postojeće rješenje u Python2	Predloženo rješenje u C++	Postojeće rješenje u Python2
TTDrive.arxml	198,5	19.393,2	199	19.383
TTADrive_Misc.arxml	370,3	37.551,2	369	37.509
TTADrive_Object.arxml	211,9	20.316,8	211	20.381,5

U tablici 4.2. se vidi kako se u predloženom rješenju u C++ smanjilo srednje vrijeme i medijan potrebni za dohvaćanje podataka iz ARXML datoteke i spremanje podataka u bazu podataka u odnosu na postojeće rješenje u Python2. Srednje vrijeme i medijan su se smanjili za skoro 100 puta u odnosu na postojeće rješenje što je znatno poboljšanje brzine rada parsera. Vidljivo je iz testova da se u oba rješenja vrijeme rada parsera poveća s veličinom ARXML datoteke koja se parsira. Iz testova se vidi i kako samo dohvaćanje podataka iz ARXML datoteka u C++ rješenju zahtjeva manje vremena nego spremanje podataka u bazu podataka. Vrijeme izmjereno u testovima za dohvaćanje se pokazalo do osam puta manjim nego vrijeme izmjereno u testovima za dohvaćanje podataka i njihovo spremanje u bazu podataka. To je zbog načina na koji se podaci spremaju u bazu podataka. Funkcije za spremanje podataka ubazu podataka pretražuju cijelu bazu dok ne pronađu mjesto gdje mogu smjestiti podatke dok funkcije za dohvaćanje podataka samo izdvajaju podatke iz datoteka po redu u kojem su pronađeni.

5. ZAKLJUČAK

Parser za komunikacijsku matricu je bitan dio generatora testnog okruženja jer bez njega generator ne može dobiti potrebne podatke iz ARXML datoteka. Parser djeluje kao posrednik između ARXML datoteka i baze podataka.

Parser za komunikacijsku matricu je parser koji kao ulaz dobiva ARXML datoteke, te iz njih dohvaća podatke koje slaže u smislene cjeline da bi te smislene cjeline dalje pohranio bazu podataka. U bazi podataka ti podaci se ili prepišu ili nadopunjuju postojeće podatke. Za rad parsera je korišten već gotov parser, tinyxml2 koji se pokazao kao adekvatan alat za potrebe ovog diplomskog rada. Parser za komunikacijsku matricu je primjenom tinyxml2-ovih funkcija pokazao prihvatljivu funkcionalnost s gledišta čitanja i spremanja podataka iz ARXML datoteka u daljnje podatkovne strukture. Testiranje je pokazalo znatno poboljšanje u brzini rada parsera u odnosu na postojeće rješenje.

Parser je programiran u C++-u koji ima prednosti nad postojećim rješenjem, koje je pisano u pythonu, što se tiče performansi te mogućnosti zaštite intelektualnog vlasništva. Parser je uspješno integriran u rad nove inačice generatora testnog okruženja koji je isto napisan u C++-u, te obećava što se tiče ispravnosti i performansi.

LITERATURA

[1] AUTOSAR XML Schema,

https://automotive.wiki/index.php/AUTOSAR_XML_Schema (pristupljeno 2021.)

[2] Andrija Mihalj, Generator softverskog koda namijenjenog za testiranje ADAS sustava , FERIT 2019.

[3] Dr. Frank, Höwing Effiziente Entwicklung von AUTOSAR-Komponenten mit domänenspezifischen Programmiersprachen INFORMATIK 2007.

[4] Simon Fürst, BMW Group, Jürgen Mössinger, Bosch Stefan Bunzel, Continental Thomas Weber, Daimler Frank Kirschke-Biller, Ford Motor Company Peter Heitkämper, General Motors Gerulf Kinkelin, Peugeot Citroën Automobiles Kenji Nishikawa, Toyota Motor Corporation Klaus Lange, Volkswagen AG, AUTOSAR – A Worldwide Standard is on the Road, 2009.

[5] Uwe Honekamp, Das AUTOSAR XML Schema und seine Bedeutung für die Implementierung von AUTOSAR Werkzeugen, INFORMATIK 2008.

[6] XML Parsershttps://www.javatpoint.com/xml-parsers (pristupljeno 2021.)

[7] DaYong Wu, Kien Tsong Chau, JingYi Wang, ChuTing Pan,A comparative study on performance of XML parser APIs (DOM and SAX) in parsing efficiency, 2019.

[8] LXML - XML and HTML with Pythonhttps://lxml.de/ (pristupljeno 2021.)

[9] An Overview of Programs and Programming Languageshttps://www.cplusplus.com/info/description/ (pristupljeno 2021.)

[10] TinyXML2 is a simple, small, efficient, C++ XML parser that can be easily integrated into other programs https://bestofcpp.com/repo/leethomason-tinyxml2-cpp-xml (pristupljeno 2021.)

[11] The strcmp() Function in C

https://overiq.com/c-programming-101/the-strcmp-function-inc/ (pristupljeno 2021.)

[12] Map in C++ Standard Template Library (STL)

https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/ (pristupljeno 2021.)

SAŽETAK

Ovaj rad se bavi problematikom razvoja parsera za komunikacijsku matricu u generatoru testnog okruženja (engl. *Test Environment Generator - TEG*). Parser za komunikacijsku matricu je jedan od glavnih dijelova TEG-a. Parser je zadužen za popunjavanje baze podataka TEG-a s podacima iz pojedinih komunikacijskih matrica koje dobiva u obliku AUTOSAR XML(ARXML) datoteka. Svaka ARXML datoteka koju parser dobiva je jedna komunikacijska matrica. Već postoji rješenje napisano u programskom jeziku Python2. U ovom radu se razvija novo rješenje u C++ programskom jeziku koji ima određene prednosti naspram Python2 programskom jezikom kao što su bolje performanse. Rješenje u C++ funkcionira tako što za ulaz primi jednu ili više ARXML datoteka te ih pretraži i dohvati sve podatke u njima. Podatke spremi u stl mape koje služe kao privremeno spremište podataka. Iz mapa se podaci spremaju u bazu podataka. Za cijeli taj proces se koristila tinyxml2 biblioteka i funkcije koje ona sadrži. Nakon razvoja rješenja su provedeni testovi ispravnosti spremanja podataka u bazu podataka te brzine rada parsera. Pokazalo se da su podaci koji se spremaju u bazu podataka ispravni. Brzina dohvaćanja i spremanja podataka u bazu podataka kod rješenja razvijenog u C++ se usporedila s brzinom dohvaćanja i spremanja podataka u bazu podataka kod rješenja razvijenog u Python2. Pokazalo se da rješenje razvijeno u C++ znatno brže dohvaća i sprema podatke, čak do sto puta brže, nego rješenje razvijeno u Python2.

Ključne riječi: AUTOSAR, ARXML, C++, tinyxml2, stl mape, baza podataka.

ABSTRACT

Development of a communication matrix parser with focus on the ARXML format inside the test environment generator

This work is about the problems of the development of a communication matrix parser in a test environment generator. The communication matrix parser is one of the main parts of the test environment generator. The parser's job is filling the test environment generator database with data from individual communication matrices which it gets in the form of AUTOSAR XML files. Every AUTOSAR XML file which the parser gets is a communication matrix. There exists a solution for the problem written in the Python2 programming language. In this work a new language is developed in the C++ programming language which has certain advantages over Python2 like better performance. The solution in C++ functions so that it gets one or more AUTOSAR XML files as an input and searches them to get all the data from them. The data is stored in stl maps which serve as temporary storage of data. The data is further stored in the database. For the whole process, the tinyxml library and its functions were used. After the development of the solution, tests were made to test the correctness of the data stored in the database and the speed of the parser. The stored data was shown to be correct. The speed of getting and storing of data in the database in the C++ solution was compared to the speed of getting and storing of data in the database in the Python2 solution. It was shown that the C++ solution was faster, almost one hundred times faster, in getting and storing data than the Python2 solution.

Keywords: AUTOSAR, ARXML, C++, tinyxml2, stl maps, database

ŽIVOTOPIS

Ivan Romanić rođen je 2. kolovoza 1996. godine u Našicama. Završio je osnovnu školu Augusta Harambašića u Donjem Miholjcu. Nakon toga upisuje opću gimnaziju u Donjem Miholjcu koju završava 2015. godine. Iste godine upisuje redovni preddiplomski sveučilišni studij Računarstvo na Fakultetu Elektrotehnike, Računarstva i Informacijskih Tehnologija koji završava 2019. godine. Trenutno pohađa drugu godinu Diplomskog sveučilišnog studija Računarstvo, izborni blok Računalno inženjerstvo.

_____

Potpis autora