Razvoj parsera za komunikacijsku matricu s fokusom na dbc i xml formate unutar generatora testnog okruženja

Tomašić, Magdalena

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:966715

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

Sveučilišni studij

Razvoj parsera za komunikacijsku matricu s fokusom na dbc i xml formate unutar generatora testnog okruženja

Diplomski rad

Magdalena Tomašić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 17.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Magdalena Tomašić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1097R, 06.10.2019.
OIB studenta:	53196472746
Mentor:	Izv. prof. dr. sc. Mario Vranješ
Sumentor:	
Sumentor iz tvrtke:	Marko Halak
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Marijan Herceg
Član Povjerenstva 1:	Izv. prof. dr. sc. Mario Vranješ
Član Povjerenstva 2:	Izv.prof.dr.sc. Ratko Grbić
Naslov diplomskog rada:	Razvoj parsera za komunikacijsku matricu s fokusom na dbc i xml formate unutar generatora testnog okruženja
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	U automobilskoj industriji automatsko testiranje je prijeko potrebno i sastavni je dio cjelokupnog procesa razvoja proizvoda ili usluge. Shodno tome, generator testnog okruženja nameće se kao nužna potreba cijelog procesa automatskog testiranja. Unutar generatora testnog okruženja postoji jedan dio koji je zadužen za parsiranje .xml i .dbc datoteka koje opisuju komunikaciju unutar sustava. U sklopu ovog zadatka potrebno je analizirati trenutnu implementaciju spomenutog dijela generatora testnog okruženja i pronaći njezine nedostatke. Na osnovu toga treba napraviti plan implementacije novog rješenja. Kao glavni dio rada treba implementirati novo rješenje u programskom jeziku C++. To rješenje treba podržavati univerzalni pristup parsiranju, tako da se može primijeniti na više projekata bez potrebe za adaptacijom i treba popuniti postojeću bazu podataka sa svim informacijama dostupnim u komunikacijskoj matrici. Nadalje, ono treba sve prikupljene podatke dinamično prepoznati i uskladištiti u točno odgovarajući dio glavne baze podataka. Na koncu, treba provjeriti performanse novog rješenja na više primjeraka komunikacijske matrice i usporediti ih s performansama trenutno dostupnog rješenja. Treba provesti testove koji potvrđuju ispravnost parsiranih podataka spremljenih u bazu usporedbom s onime što daje trenutno rješenje. "Tema rezervirana za: Magdalena Tomašić" "Sumentor iz tvrtke: Marko Halak (Institut RT-RK Osijek d.o.o.)"
Prijedlog ocjene pismenog dijela ispita	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	17.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 22.09.2021.

Ime i prezime studenta:	Magdalena Tomašić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1097R, 06.10.2019.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj parsera za komunikacijsku matricu s fokusom na dbc i xml formate unutar generatora testnog okruženja**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Mario Vranješ

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Automobilska industrija susreće se s velikim izazovima u osiguranju kvalitete i sigurnosti dijelova koji se koriste u automobilima. U današnjoj industriji automobila sigurnost vozila predstavlja izuzetno važan dio u njihovoj proizvodnji. Sustavi koji se koriste u automobilima sve više napreduju, a u skladu s tim nužno je osigurati njihovu pouzdanost.

U tu svrhu potrebno je odrediti način na koji će se provoditi testovi za osiguravanje pouzdanosti automobila, pripremiti resurse i obučiti kvalitetan kadar koji će ta testiranja provoditi. Osiguravanje pouzdanosti automobila najčešće je ostvareno provedbom niza testova kojima se uviđaju pogreške i neželjeni načini rada sustava.

Za cjelokupni proces proizvodnje automobila testiranje igra važnu ulogu, no nerijetko je skupo, zahtjevno, ponekad neizvedivo i sklono oštećenjima komponenata nad kojima se testiranje provodi. Zbog toga se teži što više automatizirati sam proces testiranja. Jedan od načina automatizacije testiranja je stvaranje testnog okruženja koje omogućava inženjerima provođenje raznih testiranja, a da pritom ne koriste stvarne, već softverske komponente.

Tako radi generator testnog okruženja, tzv. TEG (engl. *Test Environment Generator*), programsko rješenje kojeg je razvila firma TTTech u programskom jeziku Python. Iako omogućava testiranje komunikacije u automobilima bez korištenja stvarnih dijelova automobila, ovo rješenje sadrži velik broj nedostataka koje je bilo potrebno pronaći i smisliti novo rješenje za stvaranje testnog okruženja kako bi se broj nedostataka postojećeg TEG-a smanjio.

Jedan dio TEG-a jest i dio zadužen za parsiranje *.dbc* i *.xml* datoteka koje opisuju komunikaciju koja se odvija unutar sustava. U sklopu zadatka ovog diplomskog rada potrebno je analizirati postojeću implementaciju navedenog dijela generatora testnog okruženja te uočiti nedostatke te implementacije. Na osnovu obavljene analize bilo je potrebno predložiti novu arhitekturu rješenja koja će biti primjenjiva na više različitih komunikacijskih matrica bez potrebe za izmjenama napisanog koda. Nakon parsiranja, potrebno je dobivene podatke spremiti u bazu podataka tako da se svi potrebni podaci iz komunikacijske matrice nalaze u njoj. Na kraju treba parsirati sve navedene vrste datoteka i testirati dobivena rješenja tako da se potvrdi ispravnost parsiranih podataka koji se nalaze u bazi.

Diplomski rad sastavljen je od pet poglavlja. Drugo poglavlje daje teorijsku podlogu i opisuje arhitekturu postojećeg rješenja uz navođenje njegovih mana. U trećem poglavlju opisano je unaprijeđeno rješenje, odnosno nova verzija parsera, a u četvrtom je ispitana funkcionalnost

predloženog rješenja za parsiranje *.dbc* i *.xml* datoteka te su predstavljeni rezultati testiranja funkcionalnosti. U zadnjem poglavlju izneseni su zaključci rada.

2. PROBLEM PARSIRANJA KOMUNIKACIJSKE MATRICE KAO DIJELA GENERATORA TESTNOG OKRUŽENJA

U ovom poglavlju pobliže je opisano znanstveno područje kojim se ovaj diplomski rad bavi i detaljnije su objašnjeni problemi koje je potrebno riješiti. Zatim je opisan AUTOSAR standard na kojem se zasniva postojeći TEG i opisan je postojeći način rada TEG-a te su pojašnjene njegove prednosti i mane. Objašnjena je i potreba za parsiranjem komunikacijske matrice u sklopu TEG-a. U zasebnom potpoglavlju opisani su još neki mogući načini testiranja komunikacije unutar ADAS-a.

2.1. Problematika rada

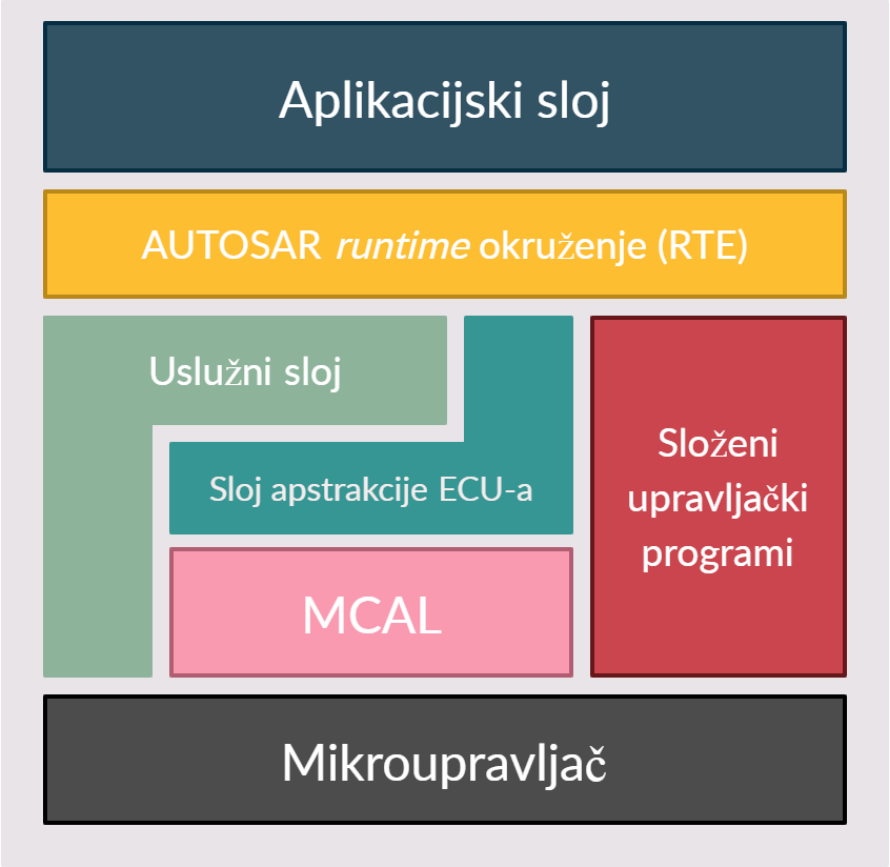
ADAS (engl. *Advanced Driver-Assistance System*) napredni je sustav senzora, procesora i aktuatora koji pružaju podršku vozaču pri upravljanju vozilom. Radari i kamere i ostali senzori u vozilu percipiraju svijet oko sebe, a prikupljene informacije prosljeđuju se procesorima na obradu. U skladu s obrađenom informacijom, sustav šalje poruku vozaču u obliku zvučnog i/ili vizualnog upozorenja, a u nekim slučajevima sustav sam reagira tako da upravlja određenom komponentom vozila (npr. automobil koči ukoliko je sustav predvidio sudar). Vozači se sve više oslanjaju na takve komponente vozila. Stoga je potrebno osigurati njihovu ispravnost i ispravnu komunikaciju među njima, a to se može učiniti izradom testova.

Želja je što više automatizirati proces testiranja jer se time znatno štede vrijeme i novac, odnosno ne troše se resursi za provedbu skupih i dugotrajnih testiranja komponenata na stvarnom modelu, već se testiranje obavlja softverski, uz pomoć različitih alata. Ovaj rad se bavi jednim od načina takvog testiranja ADAS-a.

2.2. AUTOSAR

AUTOSAR (engl. *AUTomotive Open System ARchitecture*) predstavlja otvorenu arhitekturu programske podrške nastalu kao rezultat suradnje automotiv tvrtki, u svrhu stvaranja standarda za izgradnju hardverskih i softverskih rješenja za automobile. Time se postiže uniformnost koda i iskoristivost programske podrške na različitim projektima [1].

Ovaj način standardizacije ostvaruje velik korak u odnosu na konvencionalni način programiranja jer razdvaja hardverski i softverski dio, definira metode programiranja te unificira programska sučelja i postupke konfiguracije. Na slici 2.1. prikazana je AUTOSAR arhitektura.



Slika 2.1. AUTOSAR arhitektura [2]

2.3. Generator testnog okruženja

Zasnovan na AUTOSAR-u, generator testnog okruženja omogućuje testiranje sustava komuniciranja i rada komponenata u automobilu, bez da koristi stvarne modele. Ovakav generator zasniva se na modelu koji opisuje arhitekturu cjelokupnog sustava automobila kao i komunikaciju među pojedinim komponentama, tako da omogućava simulaciju stvarnog automobila. Zato se ti testovi odvijaju bez stvarnih komponenata, što znatno olakšava testiranje. U potpoglavlju 2.6. detaljnije će biti opisano ustupljeno mi postojeće rješenje TEG-a.

2.4. Potreba za parsiranjem komunikacijske matrice

Komunikacijska matrica je dokument koji sadrži informacije od značaja, kao što su podaci o strukturi mreže, protokolima koji se koriste, signalima, o komunikaciji između upravljačkih jedinica i slično.

Zadaća je parsera traženje specifičnih vrijednosti unutar komunikacijske matrice potrebnih za rad generatora i njihovo pohranjivanje u bazu podataka. Parser na svom ulazu može primiti

različite vrste datoteka *(.dbc, .xml, .arxml, .ldf…)* koje sadrže podatke o arhitekturi elektroničkih komponenata i o njihovoj komunikaciji, a kao izlaz daje te iste vrijednosti zapisane po određenim pravilima u bazi podataka. Ti se podaci nakon parsiranja koriste u ostalim dijelovima generatora testnog okruženja kao što su generatori koda.

2.5. Pregled postojećih rješenja za automatizaciju testiranja komunikacije unutar ADAS-a

Testiranje komunikacije unutar ADAS-a moguće je provesti na više različitih načina. Jedan od načina je korištenje TTCN-3 (engl. *Testing and Test Control Notation* ver.3) jezika za testiranje, koji je standardizirani jezik za opis i definiciju testova, testnih slučajeva i testnih podataka. Uveden je za testiranje komunikacijskih protokola kakvi se mogu naći i unutar sustava automobila.

U znanstvenom radu [3] izvršeno je testiranje na svjetlima auta. Napisan je TTCN-3 test u skladu s AUTOSAR standardom i odrađeno testiranje. Za određeni ulaz mjeren je izlaz dobiven na svjetlima auta, uz poznati traženi izlaz. Rezultati su pokazali visoku uspješnost testiranja, ali u radu je obrađeno samo testiranje jednostavnih sustava kao što su svjetla automobila. Nedostatak ovog testiranja očituje se u složenosti testiranja, potrebno je mnogo vremena za svladavanje svih mogućnosti koje nudi.

Za razliku od prethodno opisanog načina, korištenjem XCP-a (engl. *Universal Measurement and Calibration Protocol*) u kombinaciji s AUTOSAR standardom moguće je testiranje tijekom čitavog razvoja softvera. Ovaj protokol omogućava čitanje iz memorije i pisanje u memoriju računala koje je dio sustava automobila, no nije ograničen samo na ugradbene računalne sustave, već se može koristiti i za dohvaćanje podataka iz softverske aplikacije, kao što je to simulacijski program osobnog računala. U radu [4] predloženo je testiranje koristeći upravo ovaj način, tako da se prvo generiraju potrebni podaci iz informacija o elektroničkim komponentama automobila i njihovih sučelja. Nakon toga se generira XCP konfiguracija, a na kraju slijedi mjerenje. S XCP-om vanjski kalibracijski sustav komunicira s ciljnom upravljačkom jedinicom putem komunikacijskog protokola (npr. CAN – engl. *Controller Area Network*, FlexRay, Ethernet itd.). Upravljačka komponenta zamijenjena je softverskim modulom, odnosno upravljačkim programom XCP-a koji tumači naredbe iz kalibracijskog sustava. Upravljački program na zahtjev dohvaća podatke izravno iz memorije upravljačke jedinice. Programer tada može analizirati te podatke i vršiti ispitivanja. Može otkriti nepravilnosti u radu i detektirati ako nešto nije u skladu s definiranom specifikacijom.

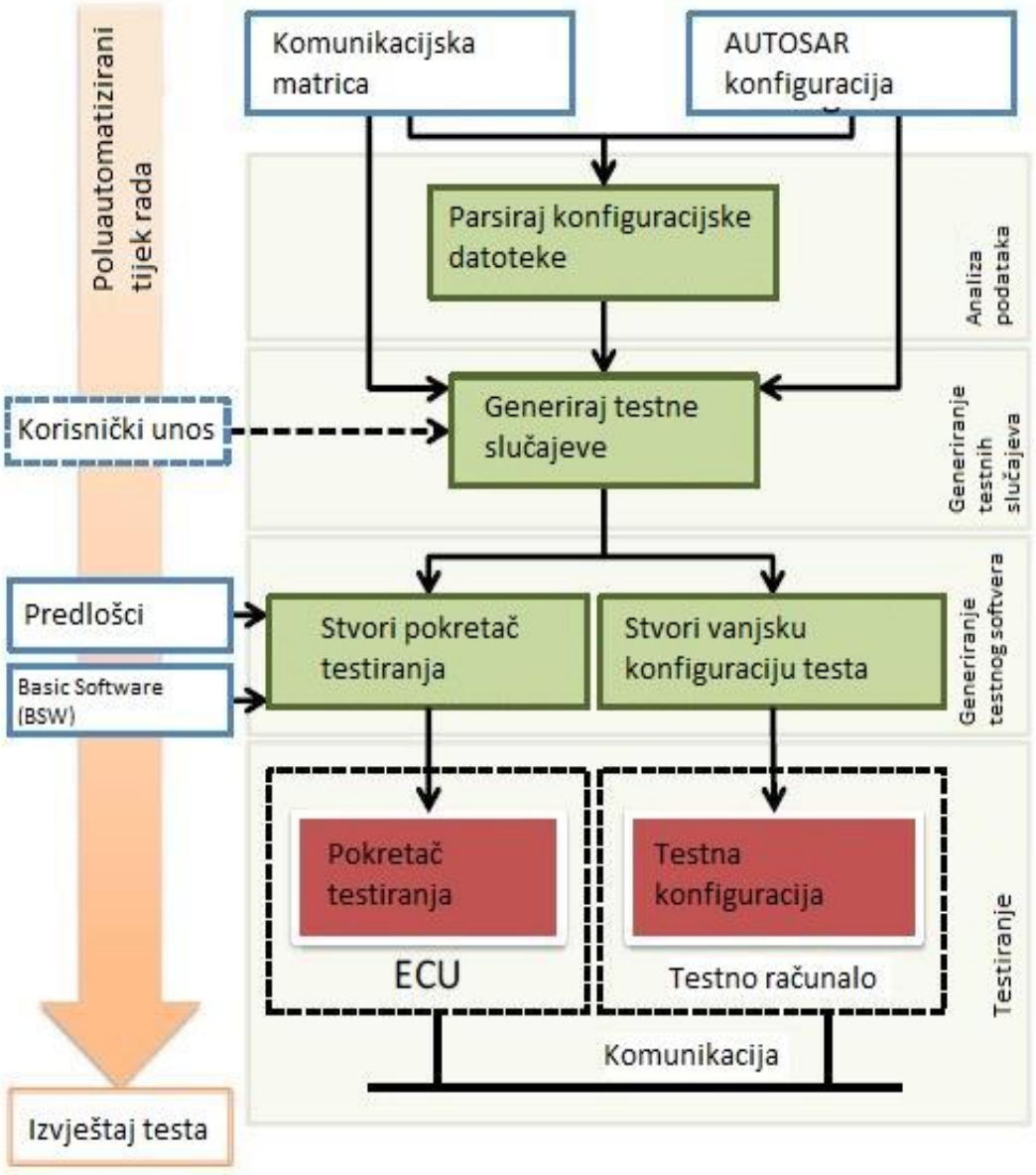
Problem ovog načina testiranja je to što ne podržava sve AUTOSAR verzije (samo verzije 4.0 i više) te programer mora naučiti raditi s alatom za ovaj način testiranja (*CANape*), što je vrlo zahtjevno i skupo. Također, nije moguća automatizacija, što rezultira povećanjem troškova i utrošenog vremena programera.

U radu [5] riješeni su navedeni nedostaci kroz tri faze izrade programske podrške za testiranje komunikacije između komponenata ADAS-a:

1. Analiza ulaznih podataka koji se nalaze unutar komunikacijske matrice i AUTOSAR konfiguracijske datoteke;
2. Generiranje testnih slučajeva;
3. Generiranje testnog softvera.

U prvoj fazi se svi potrebni podaci izdvajaju iz AUTOSAR projekta, a nalaze se unutar AUTOSAR konfiguracijske datoteke i komunikacijske matrice. Ti se podaci parsiraju te nakon toga slijedi druga faza u kojoj se generiraju testovi. U trećoj fazi se pomoću unaprijed definiranih predložaka napisanih u C programskom jeziku dobije program s nizom testova. Koristeći vanjski program, kontrolira se proces testiranja i evaluiraju se dobiveni rezultati. Rezultati se prikazuju korisniku u obliku *.csv* datoteke.

Na slici 2.2. prikazan je dijagram toka izvođenja opisanog načina testiranja komunikacije. Na ulazu programa nalaze se konfiguracijske datoteke koje opisuju sve komponente danog modela i komunikaciju među tim komponentama pomoću varijabli i njihovih vrijednosti. Nakon parsiranja svih navedenih datoteka, na osnovu prikupljenih podataka iz tih datoteka i na osnovu korisnički unesenih podataka, generiraju se testni slučajevi. Po predlošcima koji su napisani u *.c* datotekama se onda generira testno okruženje koje se izvodi na upravljačkim jedinicama tzv. ECU-ovima (engl. *Electronic Control Units*) i na osobnom računalu za provjeru testova. Na kraju se generira izvještaj testiranja kako bi se eventualne nepravilnosti mogle uvidjeti i ispraviti.

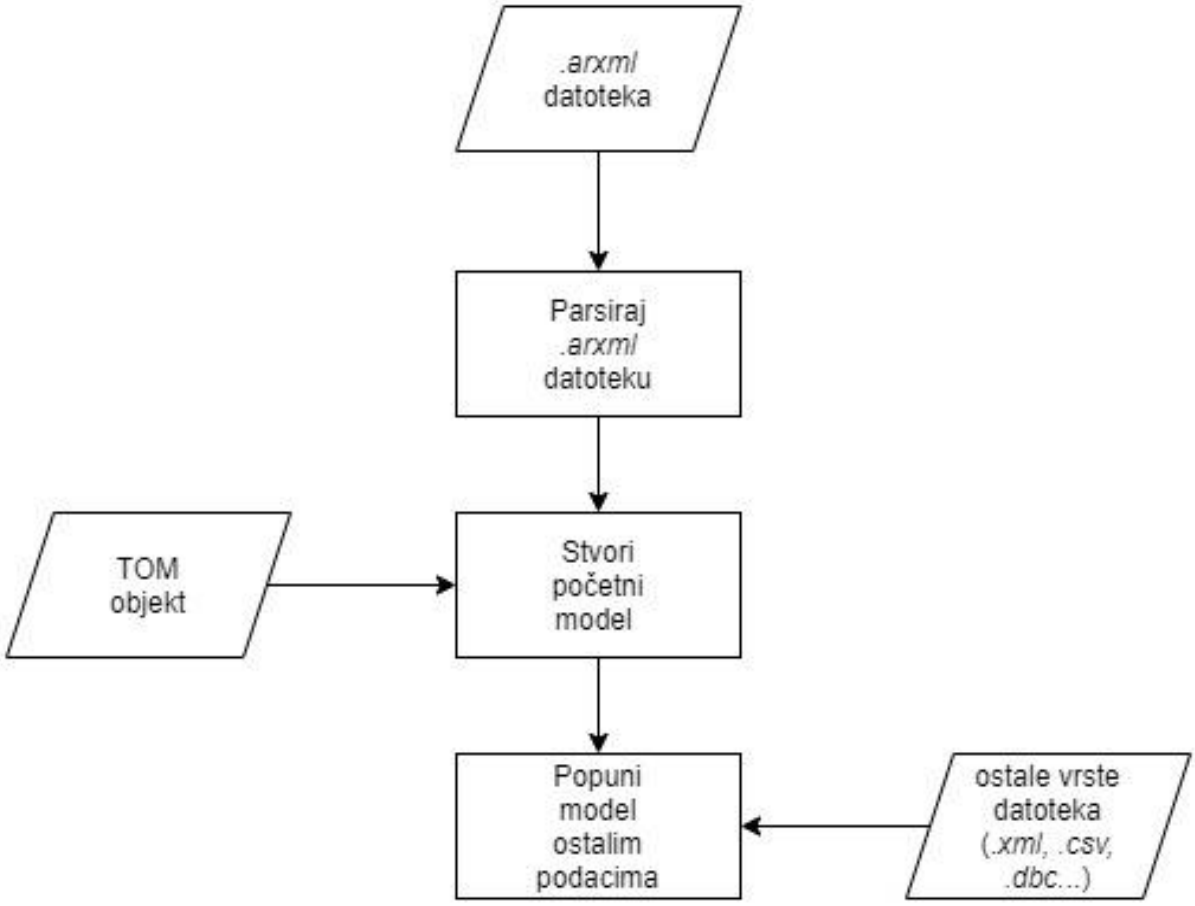


Slika 2.2. Dijagram toka izvođenja programske podrške za provođenja testiranja komunikacije unutar ADAS-a generiranjem testnih slučajeva [5]

2.6. Opis postojećeg rješenja TEG-a

Na pristupu koji je opisan u radu [5] zasniva se i postojeće rješenje TEG-a [6] koje je, za potrebe ovog diplomskog rada, ustupila firma TTTech. Sadrži tri komponente: parser, bazu podataka i generator koda. Na ulazu se nalaze različite vrsta datoteka iz kojih se iščitavaju podaci o arhitekturi modela i na temelju njih se stvara baza podataka, a na temelju nje onda rade generatori koda koji stvaraju testno okruženje.

Generiranje testnog okruženja započinje parsiranjem datoteka na samom ulazu TEG-a (*.arxml*, *.xml*, *.dbc*, i sl.). ARXML (AUTOSAR XML) je jedna vrsta *.xml* datoteke koja u sebi sadrži podatke o portovima koji se koriste u komunikaciji, o paketima koji se šalju i o softverskim komponentama, tzv. SWC (engl. *Software component*). Dakle, unutar *.arxml* datoteke nalaze se sve potrebne informacije o komunikaciji između upravljačkih jedinica ADAS-a kao i o komunikaciji unutar jednog ECU-a. Kada se ta datoteka parsira, iz nje se izdvajaju opisani podaci i dobije se model koji je zapravo baza podataka. Tu bazu čine varijable i njihove vrijednosti koje opisuju komunikaciju unutar ADAS-a. Nakon što se stvorio model, baza se nadopunjava vrijednostima koje se dobiju iz ostalih datoteka koje opisuju već spomenutu komunikaciju. Struktura ovih datoteka vrlo je složena i opisana je u nekoliko desetaka tisuća linija. Na slici 2.3. prikazan je dijagram toka izvođenja parsiranja prethodno navedenih datoteka. Prvo se parsira *.arxml* datoteka koja sadrži početne vrijednosti o komunikaciji između ECU-ova. Na temelju te datoteke i predefiniranog TOM (engl. *Test Object Model*) objekta stvara se početni model koji se popunjava ostalim podacima koji se izdvajaju iz ostalih vrsta datoteka (*.dbc*,*.xml*,*.csv*…).

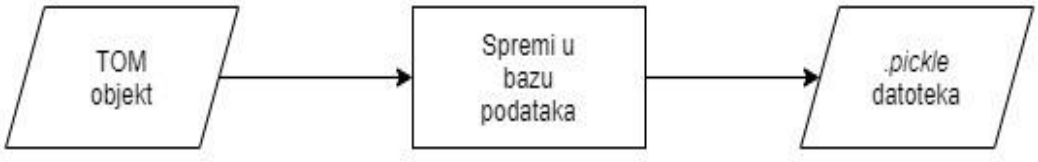


Slika 2.3. Dijagram toka parsiranja unutar postojećeg TEG-a [7]

U postojećem TEG-u, parser se zasniva na Python objektu naziva TOM. Struktura ovog objekta prethodno je definirana klasama, a korijenski objekt naziva se *TOM Root*. Unutar parsera definirana je funkcija parsiranja koja prima dva argumenta: korijenski TOM objekt i putanju do *.arxml* datoteke. Funkcija sekvencijalno prolazi kroz datoteku i pohranjuje podatke u objekte glavnog TOM objekta. Pohranom podataka u objekte, parser završava s radom.

Pohrana podataka zasniva se na standardnom Python modulu za pohranu podataka koji se naziva *Pickle*. *Pickle* modul kao metodu spremanja podataka upotrebljava serijalizaciju što znači da se podaci spremaju u obliku binarnog toka znakova. Isto tako, moguće je dobiti originalni strukturu pohranjenog objekta deserijalizacijom jer se spremanje provodi uz pomoć stoga (engl. *Stack*) koji sprema upute potrebne za rekonstrukciju tog objekta. Često se koristi ovaj modul jer je jednostavan u svojoj izvedbi i vrlo lako može pohraniti i veće i složenije Python objekte. Mana ovog modula očituje se u specifičnom binarnom zapisu koji se dobije pri pohrani podataka što rezultira mogućnošću čitanja zapisanih podataka samo uz pomoć tog istog modula. Iz toga proizlazi još jedan nedostatak, a to je nemogućnost provjere ispravnosti zapisanih podataka u *.pickle* datoteci.

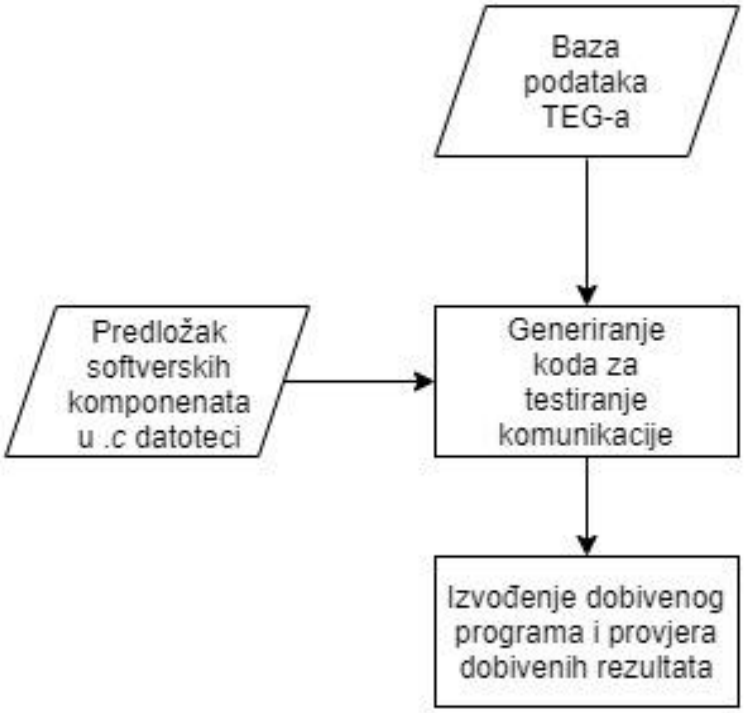
Nakon završetka parsiranja konfiguracijskih datoteka, poziva se funkcija za pohranu izdvojenih podataka koja prima korijenski objekt *TOM Root* i putanju do željene lokacije spremanja *.pickle* datoteke. Sada se sav sadržaj *TOM Root* objekta nalazi u lokalnom odabranom direktoriju i može mu se pristupiti i nakon završetka izvršavanja programa. Na slici 2.4. prikazan je dijagram toka pohranjivanja prikupljenih podataka u bazu podataka TEG-a. Iz TOM objekta dobivenog nakon parsiranja potrebnih datoteka nastaje *.pickle* datoteka.



Slika 2.4. Dijagram toka spremanja podataka u bazu podataka [7]

Kada su svi podaci spremljeni i dostupni za čitanje, moguće je početi s generiranjem testnog koda pozivanjem odgovarajuće funkcije koja pokreće generatore koda. Oni koriste podatke iz glavne baze podataka te po određenom nacrtu koji se nalazi u *.c* datoteci zapisuje podatke. Tako nastaju testni kodovi za provjeru modela komunikacije. Na slici 2.5. nalazi se dijagram toka izvođenja generatora koda. Na osnovu prikupljenih podataka i stvorenog modela komunikacije te

uz pomoć predložaka softverskih komponenata, generira se kod za testiranje komunikacije. Dobiveni kod se onda može izvoditi na ECU-ovima i utvrditi ispravnost komunikacije između pojedinih ECU-ova kao i komunikacija unutar jednog ECU-a.



Slika 2.5. Dijagram toka generiranja koda za testno okruženje [7]

Problem ovog rješenja očituje se u izboru programskog jezika (Python 2.7). Bilo bi poželjno prijeći na noviju verziju Pythona jer je kod zastario, a za komercijalizaciju je potrebno koristiti niži programski jezik koji će ujedno doprinijeti i smanjenju vremena izvođenja TEG-a jer će biti brži od Pythona. Još jedan nedostatak je nemogućnost provjere jesu li svi potrebni podaci nađeni u datotekama i spremljeni u bazu. Moguće je da neki podaci nisu pronađeni s postavljenim kriterijima ili su pronađeni, ali se nisu spremili u bazu jer nije pronađeno odgovarajuće mjesto i jednostavno se ne nalaze u bazi. TEG će i dalje raditi, ali s puno manje informacija. Isto tako, *Pickle* koji se koristi za pohranu podataka je nesiguran i nečitljiv izvan programa. Ne može se nikako provjeriti što se nalazi unutar datoteke, koji su to podaci spremljeni, a moguće je i naknadno ubaciti podatke koji su nepotrebni i mogu naštetiti cijelom programu. Uz sve navedeno, kod je previše modularan. Ima previše ponavljajućih funkcija i postoji mogućnost skraćivanja koda i njegove optimizacije.

3. POSTUPAK IZRADE PARSERA ZA *.DBC* I *.XML* DATOTEKE

Na osnovu navedenih nedostataka postojećeg rješenja parsera, bilo je potrebno definirati novu arhitekturu rješenja da bi se postigla željena dinamičnost. Da bi se to ostvarilo, parsiranje se vrši liniju po liniju, tako da se podaci izdvajaju prema unaprijed definiranim pravilima, a zatim se spremaju u bazu podataka na za to predviđena mjesta. Kao što je navedeno u naslovu poglavlja, ovaj parser izdvaja podatke spremljene u *.dbc* i *.xml* datoteke. U ovom će poglavlju detaljno biti objašnjeno parsiranje, od traženja podataka do spremanja u bazu.

3.1. Zahtjevi koje novo rješenje treba zadovoljiti

Analizom postojećeg rješenja utvrđeni su nedostaci predstavljeni u prethodnom pogavlju. S ciljem smanjenja broja tih nedostataka definirana je nova arhitektura rješenja parsera za *.dbc* i *.xml* datoteke, pri čemu je bilo potrebno zadovoljiti sljedeće zahtjeve:

1) Izraditi novo rješenje parsera u C++ programskom jeziku
2) Postići univerzalnost u smislu mogućnosti primjene istog parsera na više različitih projekata bez potrebe za prepravljanjem danog rješenja
3) Pohraniti sve podatke dobivene parsiranjem u odgovarajući dio baze podataka dobivene na temelju *.arxml* modela

Provjerom performansi novog rješenja različitim testovima i usporedbom s postojećim rješenjem utvrđuje se jesu li uklonjeni nedostaci koji su ranije otkriveni analizom postojećeg rješenja TEG-a.

3.2. Alati i tehnologije korišteni za izradu parsera

Kako bi se ispunio početni zahtjev, rješenje je programirano u C++ programskom jeziku u Visual Studiu. Uz standardne biblioteke korištene su i dodatne kao što je *tinyxml2* koja će pomoći pri parsiranju *.xml* datoteka.

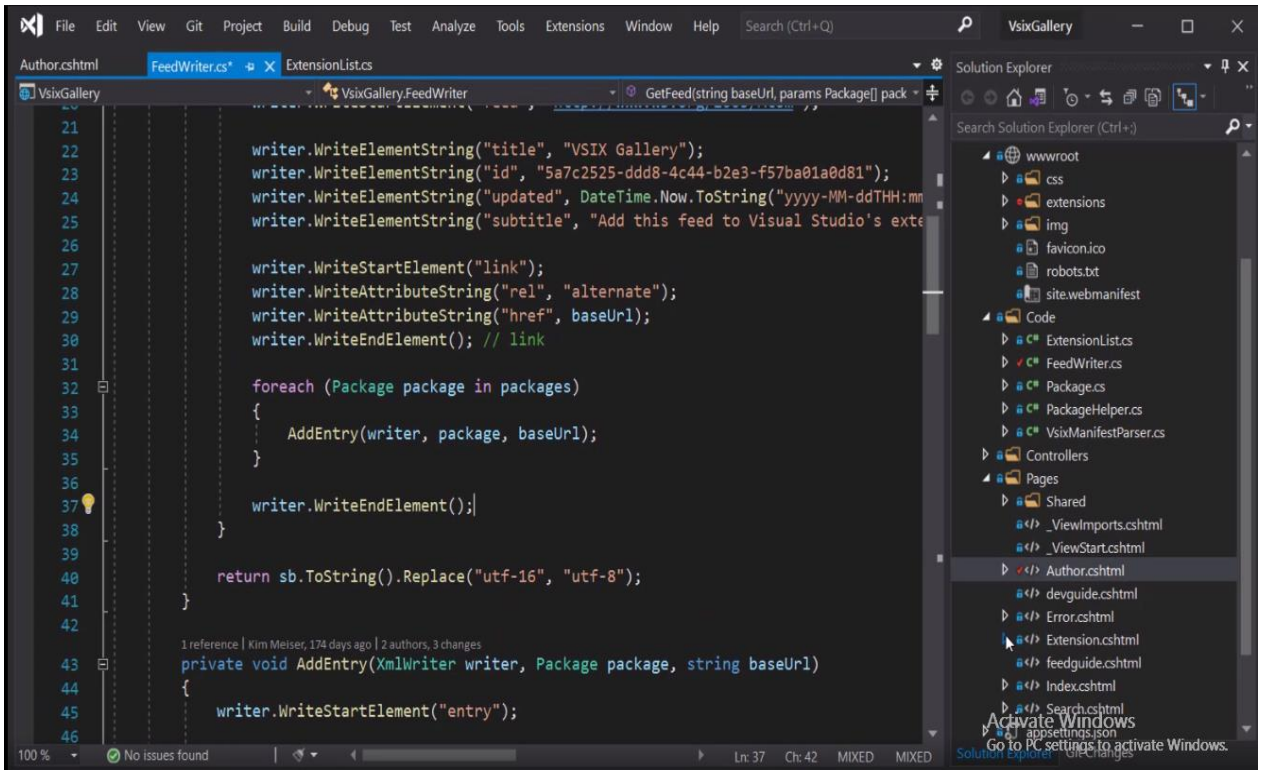
3.2.1. C++

C++ [8] je programski jezik široke namjene. Omogućava korisniku pisanje koda za različite aplikacije, igre, operacijske sustave i za mnoge druge primjene. Ovaj programski jezik podržava objektno orijentirano programiranje i njegove četiri glavne odlike su: apstrakcija, nasljeđivanje, polimorfizam i enkapsulacija. Zbog svoje prenosivosti ima veliku primjenu kod ugradbenih računalnih sustava.

U usporedbi s programskim jezikom Python postiže veće brzine izvođenja te omogućava komercijalizaciju programskih rješenja, što bi trebalo donijeti prednosti u odnosu na postojeći TEG. S druge strane, nepravilna upotreba pokazivača može prouzročiti neželjena ponašanja sustava i curenje memorije (engl. *memory leak*). Također, ne postoji dio za brigu o smeću te se programer mora sam pobrinuti o podacima koje više ne treba tako da ih obriše iz memorije.

3.2.2. Visual studio

Visual Studio [9] je integrirano razvojno okruženje tzv. IDE (engl. *integrated development environment*) kojeg je razvio Microsoft. Koristi se za programiranje različitih aplikacija, od web preko mobilnih pa sve do aplikacija za Windows, a podržava i više jezika. Sadrži i dizajnerski dio u kojem programer ima mogućnost kreiranja izgleda aplikacije i određivanja komunikacije s korisnikom preko različitih grafičkih elemenata, kao što su prozori, gumbi, različita polja za unos i prikaz teksta i sl. Na slici 3.1. prikazan je glavni prozor Visual Studia.



Slika 3.1. Izgled glavnog prozora Visual Studia [9]

3.2.3. Tinyxml2

Biblioteka *tinyxml2* koja sadrži C++ XML parser može se preuzeti s github-a i lako integrirati u vlastiti program [10]. Ulaz ovog parsera je XML dokument, a izlaz je DOM (engl. *Document Object Model*), objekt koji je moguće čitati, modificirati i spremiti.

Ako se želi koristiti, dovoljno je u projekt dodati prethodno skinutu datoteku zaglavlja *tinyxml2.h* i datoteku u *.cpp* formatu *tinyxml2.cpp*. Izdana je pod ZLib licencom, stoga ju je moguće koristiti i u komercijalne svrhe.

U nastavku su prikazani primjeri korištenja ove biblioteke, odnosno raznih funkcija parsiranja XML dokumenta. Na slici 3.2. nalazi se primjer XML datoteke.

```
<?xml version=\"1.0\"?>
<!DOCTYPE PLAY SYSTEM \"play.dtd\">
<PLAY>
  <TITLE>A Midsummer Night's Dream</TITLE>
</PLAY>;
```

Slika 3.2. Primjer XML datoteke

Na slici 3.3. nalazi se primjer koda u C++ programskom jeziku za učitavanje XML dokumenta.

```
XMLDocument doc;
doc.LoadFile( "resources/dream.xml" );
return doc.ErrorID();
```

Slika 3.3. Primjer koda u C++ programskom jeziku za učitavanje XML dokumenta

Na slici 3.4. nalazi se primjer koda u C++ programskom jeziku za dohvaćanje TITLE elementa dokumenta.

```
XMLDocument doc;
doc.Parse( xml );
XMLElement* titleElement = doc.FirstChildElement("PLAY")
->FirstChildElement("TITLE");
```

Slika 3.4. Primjer koda u C++ programskom jeziku za dohvaćanje TITLE elementa dokumenta

Na slici 3.5. nalazi se primjer koda u C++ programskom jeziku za dohvaćanje informacija iz dokumenta.

```
XMLText* textNode = titleElement->FirstChild()->ToText();
title = textNode->Value();
printf( "Name of play : %s\n", title );
```

Slika 3.5. Primjer koda u C++ programskom jeziku za dohvaćanje informacija iz dokumenta

3.3. Opis prijedloga programskog rješenja u svrhu poboljšanja postojećeg TEG-a

Na temelju ustupljenog rješenja nastalo je programsko rješenje za novi TEG. Arhitektura je vrlo slična. TEG se sastoji od sedam dijelova. Glavni dio čini okvir (engl. *framework*) i rukovatelj baze podataka (engl. *database handler*). Na ulazu u novi TEG nalaze se model parser, komunikacijski parser *.arxml* datoteka te komunikacijski parser ostalih vrsta datoteka, a na njegovom izlazu generator C i generator CAPL koda.

Velik odmak u odnosu na ustupljeno rješenje čini baza podataka koja je sastavljena od mapa s ključem i vrijednostima vezanim uz određeni ključ te pripadajućim funkcijama koje se nalaze unutar rukovatelja baze podataka. Tako svaki podataka ima svoj ključ i vrijednost koja je zapisana pod tim ključem. Na početku model parser stvara bazu takvih mapa na temelju podataka iz konfiguracijskih datoteka. Onda komunikacijski parseri mogu, uz pomoć funkcija rukovatelja baze podataka, popunjavati bazu svojim podacima izdvojenim iz ulaznih datoteka parsera.

3.4. Programsko rješenje parsera za *.dbc* i *.xml* datoteke

U okviru generatora testnog okruženja nalazi se parser čija je zadaća traženje točno određenih podataka te njihovo spremanje u bazu podataka. U ovom radu fokus parsiranja jesu *.dbc* i *.xml* datoteke koje u sebi sadrže bitne informacije o komunikaciji između pojedinih ECU-ova. Komunikacija je definirana pojedinim varijablama i njihovim vrijednostima koje se nalazi unutar tih datoteka.

3.4.1. Osnovne informacije o *.dbc* datoteci

Struktura *.dbc* datoteke definirana je određenim pravilima koja se nalaze u dokumentaciji [11], a primjer jedne takve datoteke može se vidjeti na slici 3.6. Komunikacija je definirana pomoću tri vrste objekta: mrežni čvor, poruka i signal. Unutar *.dbc* datoteke redom su definirane ključne riječi: BU za mrežni čvor, BO za poruku i SG za signal. Oznaka CM je oznaka komentara. Datoteka sadrži informacije za prikaz podataka mreže zasnovane na CAN protokolu u obliku fizičkih vrijednosti, odnosno vrijednosti signala. Kao primjer može se uzeti brzina motora kao signal (*SG_ EngSpeed*). U tom slučaju postoje vrijednosti koje definiraju iznos brzine motora te vrijednost koja govori koja je mjerna jedinica te brzine.

```
VERSION ""

NS_ :
     NS_DESC_
     CM_
     BA_DEF_
     BA_
     VAL_
     CAT_DEF_
     CAT_
     FILTER
     BA_DEF_DEF_
     EV_DATA_
     ENVVAR_DATA_
     SGTYPE_
     SGTYPE_VAL_
     BA_DEF_SGTYPE_
     BA_SGTYPE_
     SIG_TYPE_REF_
     VAL_TABLE_
     SIG_GROUP_

     SIG_VALTYPE_
     SIGTYPE_VALTYPE_
     BO_TX_BU_
     BA_DEF_REL_
     BA_REL_
     BA_DEF_DEF_REL_
     BU_SG_REL_
     BU_EV_REL_
     BU_BO_REL_

BS_:

BU_: Engine Gateway

BO_ 100 EngineData: 8 Engine
 SG_ PetrolLevel : 24|8@1+ (1,0) [0|255] "l"  Gateway
 SG_ EngPower : 48|16@1+ (0.01,0) [0|150] "kW"  Gateway
 SG_ EngForce : 32|16@1+ (1,0) [0|0] "N"  Gateway
 SG_ IdleRunning : 23|1@1+ (1,0) [0|0] ""  Gateway
 SG_ EngTemp : 16|7@1+ (2,-50) [-50|150] "degC"  Gateway
 SG_ EngSpeed : 0|16@1+ (1,0) [0|8000] "rpm"  Gateway


CM_ "CAN communication matrix for power train electronics
**********************************************************

implemented: turn lights, warning lights, windows";

VAL_ 100 IdleRunning 0 "Running" 1 "Idle" ;
```

Slika 3.6. Primjer *.dbc* datoteke

Ova tekstualna datoteka opisuje cjelokupnu komunikaciju jedne mreže zasnovane na CAN protokolu tako da se na osnovu nje lako može vršiti analiza cijele mreže te je također moguće utvrditi uvjete pod kojima je moguće dodavanje novih komponenti mreže.

3.4.2. Osnovne informacije o *.xml* datoteci

Druga vrsta datoteke koja se koristi dolazi u *.xml* formatu [12]. XML skraćenica dolazi od engleskog naziva „*eXtensible Markup Language*“, a navedeni format sadrži podatke čitljive stroju i čovjeku. Datoteka ovog projekta u .xml formatu dolazi u uobičajenom obliku s tagovima i njihovim vrijednostima, što se može vidjeti na slici 3.7. Unutar nje se nalaze podaci o domaćinima (HOST), njihovim jezgrama (CORE) i o softverskim komponentama (SWC). Pomoću biblioteke *tinyxml2* omogućeno je čitanje vrijednosti tagova i atributa ako postoje.

```
<HOSTS>
  <HOST>
    <HOST-ID accessLevel="readOnly">1</HOST-ID>
    <HOST-NAME accessLevel="readOnly">PerformanceHost00</HOST-NAME>
    <SCHEDULE-TYPE accessLevel="powerUser">TIME-TRIGGERED</SCHEDULE-TYPE>
    <CORES>
      <CORE>
        <CORE-ID accessLevel="readOnly">0</CORE-ID>
        <MACROTICK unit="us" accessLevel="powerUser">100</MACROTICK>
        <ISR-OVERHEAD-EXTENSION unit="us" accessLevel="standardUser">200</ISR-OVERHEAD-EXTENSION>
        <ISR-OVERHEAD-FACTOR accessLevel="standardUser">1.0</ISR-OVERHEAD-FACTOR>
      </CORE>
      <CORE>
        <CORE-ID accessLevel="readOnly">1</CORE-ID>
        <MACROTICK unit="us" accessLevel="powerUser">100</MACROTICK>
        <ISR-OVERHEAD-EXTENSION unit="us" accessLevel="standardUser">200</ISR-OVERHEAD-EXTENSION>
        <ISR-OVERHEAD-FACTOR accessLevel="standardUser">1.0</ISR-OVERHEAD-FACTOR>
      </CORE>
    </CORES>
  </HOST>
</HOSTS>
```

Slika 3.7. Primjer *.xml* datoteke

3.4.3. Programiranje parsera za *.dbc* i *.xml* datoteke

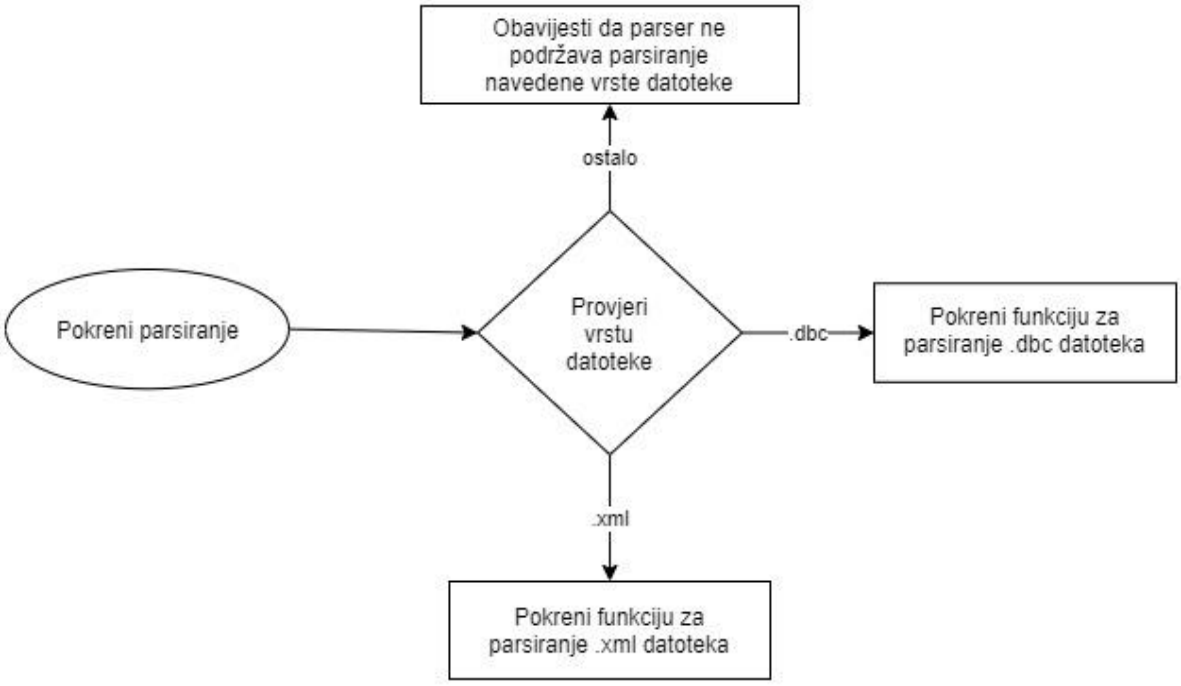
Programiranje parsera izvedeno je tako da za svaku vrstu datoteke postoji funkcija koja rješava problem parsiranja te iste datoteke. Na slici 3.8. nalaze se sve funkcije koje čine parser za *.dbc* i *.xml* datoteke. Navedeni parser je zapravo klasa *OtherParser* koji sadrži dvije privatne metode, po jednu za svaku vrstu datoteke (*parseCANdb* za parsiranje *.dbc* datoteka i *parseSchedulingData* za parsiranje *.xml* datoteka), te javnu metodu *OtherParser* kojom se parser poziva. Funkcije izvan same klase su pomoćne funkcije parsera koje se koriste unutar dviju navedenih metoda parsera. Funkcija *goToFileStart* koristi se kod čitanja datoteke te vraća pokazivač s kraja na početak

datoteke. Četiri funkcije za izdvajanje podataka *extractDatabaseName*, *extractMessages*, *extractCycleTime* i *extractSignals* koriste se unutar funkcije za parsiranje *.dbc* datoteka.

```
void goToFileStart();
void extractDatabaseName();
void extractMessages();
void extractCycleTime();
void extractSignals();


class OtherParser {

private:
   static void parseCANdb(DatabaseHandler* dbH, fs::path candb_path);
   static void parseSchedulingData(DatabaseHandler* dbH, fs::path sd_path);

public:
   static void otherParser(DatabaseHandler* dbh, vector<fs::path> paths);
};
```

Slika 3.8. Klasa *OtherParser* s pripadajućim metodama

Pokretanje parsera počinje pozivom metode *OtherParser* klase *OtherParser*. Metoda prima dva argumenta, pokazivač na bazu podataka i vektor putanji do datoteka koje je potrebno parsirati. Ta metoda provjerava koja vrsta datoteke je proslijeđena i u skladu s tim poziva odgovarajuću funkciju parsera. Navedeni postupak prikazan je na slici 3.9.



Slika 3.9. Dijagram toka programa za početni dio parsiranja

Nakon što se učita datoteka koju je potrebno parsirati, ona se čita liniju po liniju da bi se pronašli podaci. Pretraživanje podataka unutar *.dbc* datoteke podrazumijeva pronalazak one linije koja sadrži zadani izraz. Ta linija sadrži podatke odvojene razmakom. Oni se uklanjaju, a podaci spremaju u točno određeni dio baze kojem ti podaci pripadaju. Na slici 3.10. nalazi se dio koda za parsiranje jednog dijela *.dbc* datoteka, a ostali dijelovi parsiraju se na isti način.

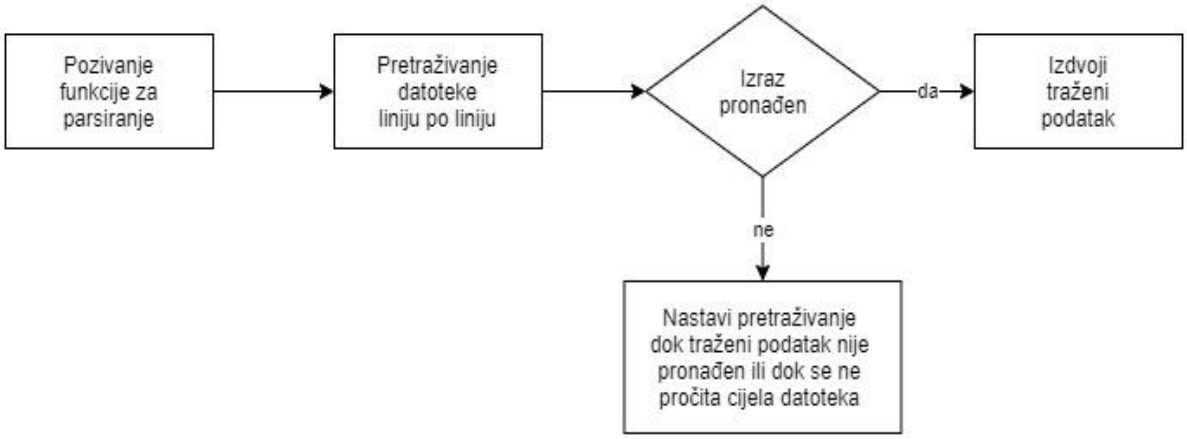
```
void extractCycleTime() {
   vector<int> msgCycleTime;
   string searchFor = "BA_ \"MsgCycleTime\" BO_";
   string cycleTimeLine, cycleTime;
   while (getline(dataCANdbFile, cycleTimeLine)) {
        if (cycleTimeLine.find(searchFor) != string::npos) {
        stringstream s(cycleTimeLine);
        getline(s, cycleTime, ' ');
        msgCycleTime.push_back(atoi(cycleTime.c_str()));
        }
   }
}
```

Slika 3.10. Primjer koda funkcije *parseCANdb* za izdvajanje podataka

Sličan postupak je i za *.xml* datoteke, samo što se tamo podaci nalaze unutar tzv. tagova i zato se za izdvajanje tih podataka koristi biblioteka *tinyxml2* s različitim funkcijama pretraživanja *.xml* datoteka. Podaci se mogu pronaći pomoću naziva, atributa, vrijednosti i sl. Na slici 3.11. nalazi se dio koda za parsiranje *.xml* datoteka, a dijagram toka programa za obje funkcije prikazan je na slici 3.12.

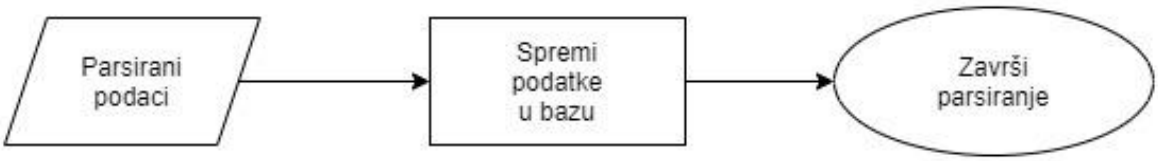
```
  vector<int> host_ids;
  vector<string> host_names;
  XMLNode* host = NULL;

     for (host = hosts->FirstChildElement("HOST");
       host != nullptr; host = host->NextSiblingElement()) {

       XMLElement* host_id = host->FirstChildElement("HOST-ID");
       const char* hostID = host_id->GetText();
       host_ids.push_back(atoi(hostID));

       XMLElement* host_name = host->FirstChildElement("HOST-NAME");
       const char* hostName = host_name->GetText();
       host_names.push_back(hostName);
     }
```

Slika 3.11. Primjer koda funkcije *parseSchedulingData* za izdvajanje podataka



Slika 3.12. Dijagram toka programa za pojedinu funkciju (*parseCANdb* i *parseSchedulingData*) unutar parsera

Funkcije parsera pozivaju se redom jedna po jedna, dok se ne parsiraju sve datoteke odnosno dok se ne pronađu i spreme svi relevantni podaci unutar tih datoteka. Ovim je ispunjen treći zahtjev te se dobije baza podataka koja sadrži bitne informacije o stanju ECU-ova i komunikaciji između njih. Ti se podaci onda dalje mogu koristiti za potrebe projekta odnosno za generiranje testnog okruženja. Time je postupak parsiranja svih datoteka završen. Na slici 3.13. prikazane su opisane faze parsiranja.



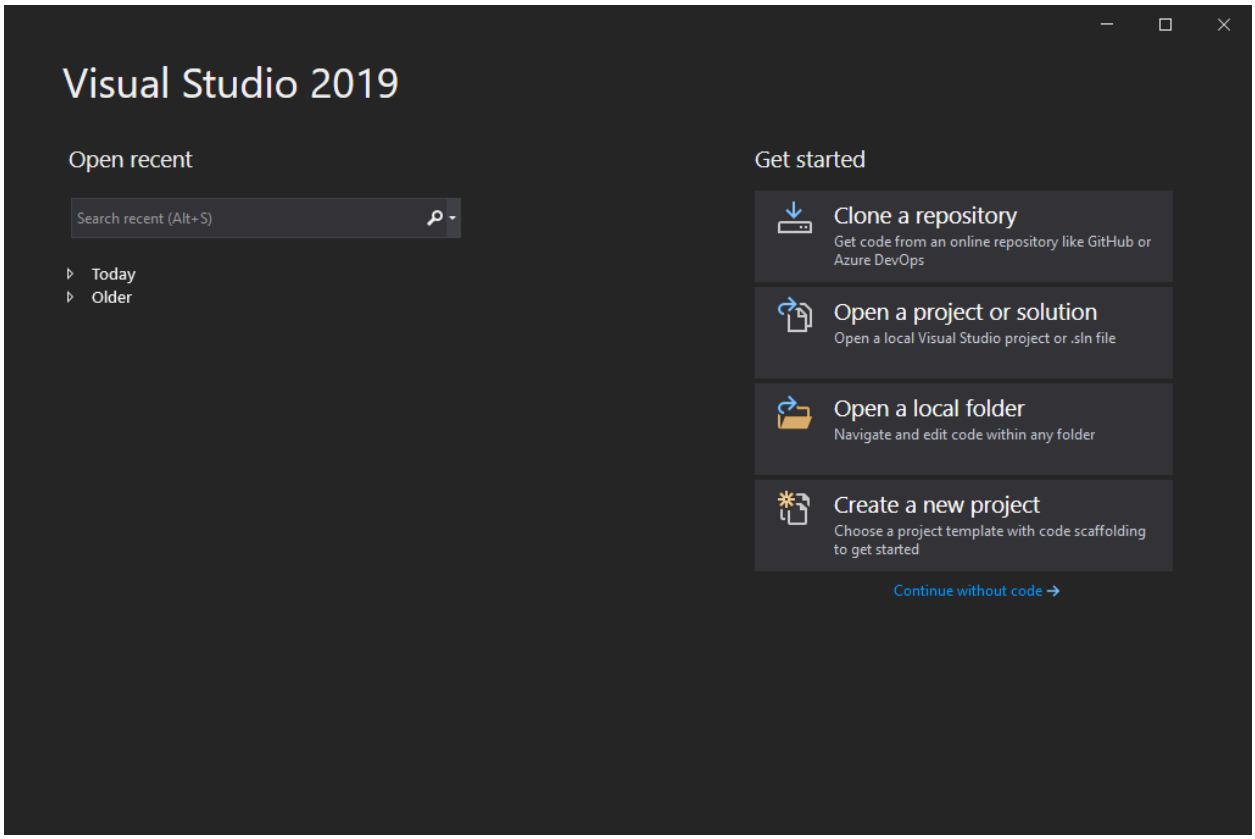
Slika 3.13. Dijagram toka programa završnog dijela parsera

Uspoređujući prethodnu i trenutnu verziju parsera, princip rada je isti. Učitane datoteke se pretražuju u potrazi za podacima, a zatim se podaci spremaju u bazu, jedino što je način na koji se oni spremaju bitno različit od prethodnog. Ne stvaraju se objekti po prethodno definiranim pravilima već se izdvojeni podaci spremaju na mjesta u bazi koja su definirana .arxml datotekom model parsera. Tako je ispunjen drugi zahtjev odnosno postiže se univerzalnost koja je veliki nedostatak prethodnog rješenja te se trenutno rješenje može primijeniti na više projekata odnosno više datoteka bez potrebe za prepravljanjem napisanog koda.

Na kraju razvoja rješenja novog parsera potrebno je testirati ispravnost rada rješenja i dobivenih podataka te ocijeniti performanse rada novog rješenja. U tu svrhu mjerit će se vrijeme izvršavanja pojedinog parsiranja, kao i parsiranje svih datoteka u cjelini, a dobiveni rezultati bit će prikazani u sljedećem poglavlju.

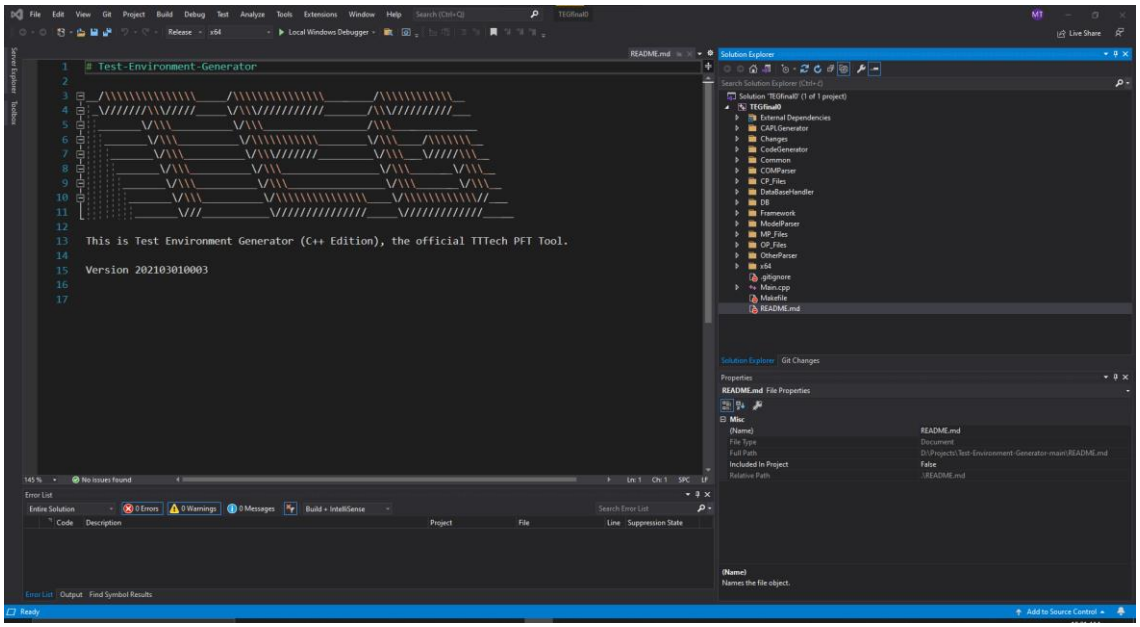
3.5. Način pokretanja programskog rješenja parsera za *.dbc* i *.xml* datoteke

Nakon pokretanja *Visual Studia*, otvara se prozor kao na slici 3.14.



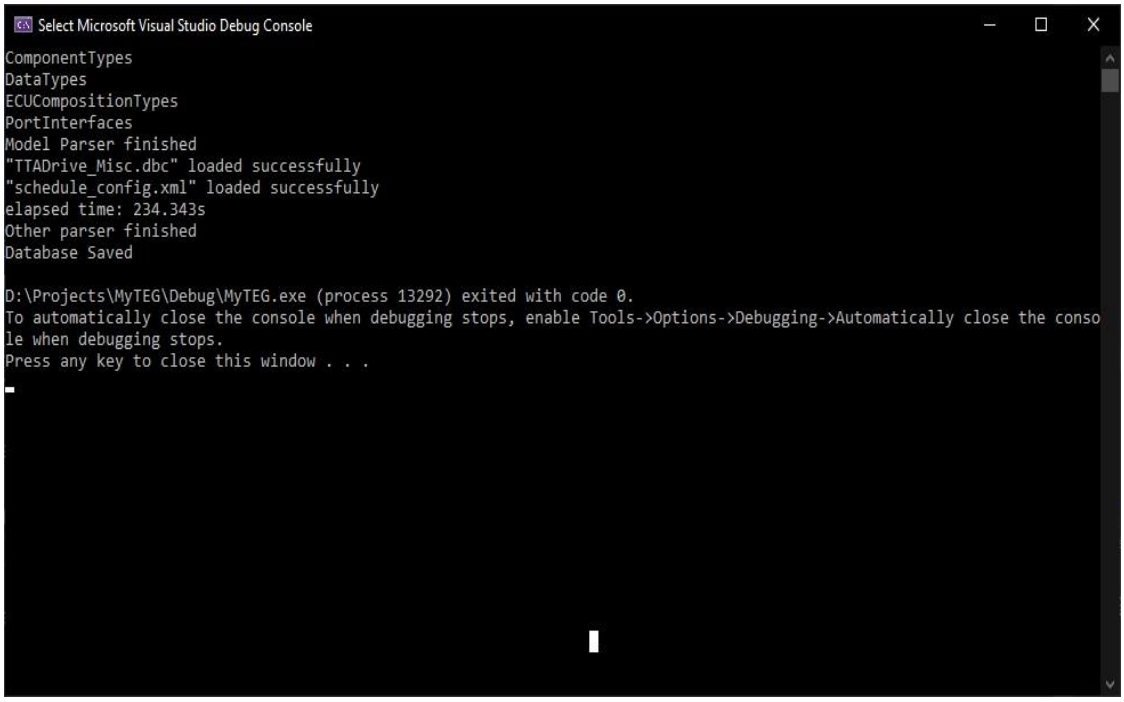
Slika 3.14. Početni prozor *Visual Studia 2019*

Projekt je moguće otvoriti klikom na „*Open a project or solution*“ unutar prozora. Potrebno je odabrati projekt naziva „TEG“. Nakon što se projekt učitao, prikazat će se zaslon kao na slici 3.15. Na desnoj strani nalaze se sve potrebne mape i datoteke projekta za ispravno funkcioniranje programskog rješenja. Projekt je podijeljen na sedam manjih dijelova i pripadajućih mapa. To su *Framework*, *DatabaseHandler*, *ModelParser*, *COMParser*, *OtherParser*, *CAPLGenerator* i *CodeGenerator*.



Slika 3.15. Prikaz otvorenog projekta u Visual Studiu

Parser za *.dbc* i *.xml* datoteke se izvodi kao jedan dio cjelokupnog projekta pod nazivom „TEG“. On se poziva kreiranjem objekta klase *OtherParser* i pozivom metode te klase, također naziva *OtherParser*. Nakon završetka izvođenja, program će dati odgovarajuću poruku o uspješnosti izvođenja te će se generirati baza podataka u obliku *.txt* datoteke unutar *TEG_FILES* mape. Na slici 3.16. je prikaz terminala nakon što je parsiranje *.dbc* i *.xml* datoteka dovršeno.



Slika 3.16. Prikaz terminala nakon što je parsiranje *.dbc* i *.xml* datoteka dovršeno

Korisnik bi, nakon što je program završio s izvođenjem, trebao na svom osobnom računalu imati novu datoteku *database* sa zapisom vremena pohranjivanja datoteke na osobno računalo u nazivu datoteke. Na slici 3.17. nalazi se isječak navedene datoteke.

```
ComponentTypes<|>CtCd_PH00<|>CtCd_PH00_InternalBehavior<|>RTM<|>+
AET<|>0.250000
WCET<|>0.500000<|>@
```

Slika 3.17. Isječak dobivene baze podataka nakon parsiranja *.dbc* i *.xml* datoteka

4. TESTIRANJE RADA PREDLOŽENOG RJEŠENJA PARSERA *.DBC* I *.XML* DATOTEKA

S obzirom na postignutu univerzalnost parsera, izmijenjeni format baze podataka i korištenje programskog jezika C++, bilo je potrebno izvršiti sljedeća testiranja:

- mjerenje vremena izvođenja parsiranja pojedinih datoteka i mjerenje vremena spremanja u bazu podataka u starom TEG-u
- mjerenje vremena izvođenja parsiranja pojedinih datoteka i mjerenje vremena spremanja u bazu podataka u novom TEG-u

4.1. Opis skupa podataka korištenih za testiranje rada predloženog rješenja parsera *.dbc* i *.xml* datoteka

Testiranja su obavljena na podacima koji se nalaze u *.xml* i *.dbc* datotekama čija je struktura opisana u potpoglavljima 3.3.1 i 3.3.2. U tablici 4.1. dani su osnovni podaci o svakoj dostupnoj datoteci. Zbog poslovne tajne ne postoji mogućnost uvida u navedene datoteke te se ne mogu priložiti uz rad.

Tablica 4.1. Pregled raspoloživih datoteka za parsiranje

Naziv datoteke	Vrsta datoteke	Broj linija	Veličina (KB)
schedule_config	*.xml*	3199	159
TTADrive	*.dbc*	299	11
TTADrive_Misc	*.dbc*	435	16
TTADrive_Object	*.dbc*	157	7

4.2. Rezultati testiranja rada predloženog rješenja parsera *.dbc* i *.xml* datoteka

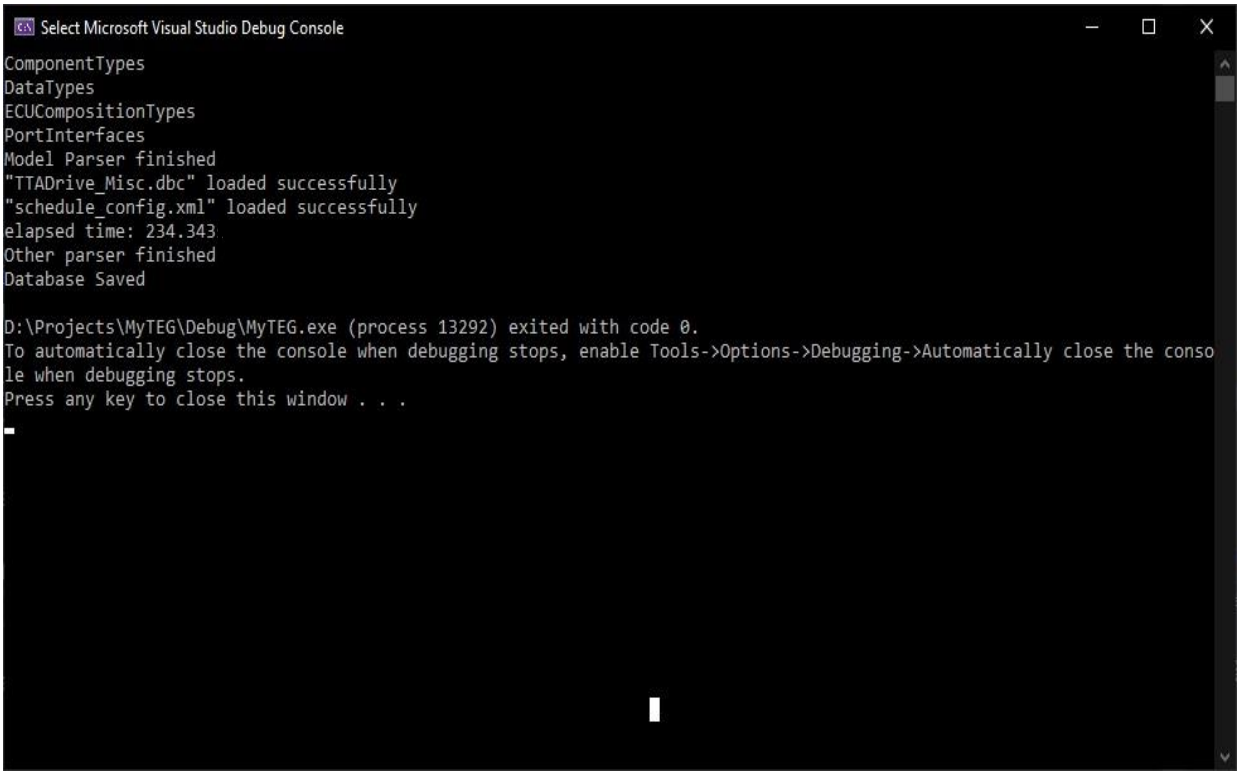
4.2.1. Rezultati mjerenja brzine parsiranja *.dbc* i *.xml* datoteka

Mjerenje brzine parsiranja *.dbc* i *.xml* datoteka obavljeno je za stari i novi TEG. Za mjerenje vremena parsiranja datoteka korištena je *chrono* biblioteka [13]. Na slici 4.1. nalazi se isječak koda za mjerenje vremena parsiranja datoteka.

```
auto start = std::chrono::steady_clock::now();
auto end = std::chrono::steady_clock::now();
std::chrono::duration<double> elapsed_mseconds = end - start;
std::cout << "elapsed time: " << elapsed_mseconds.count() << "s\n";
```

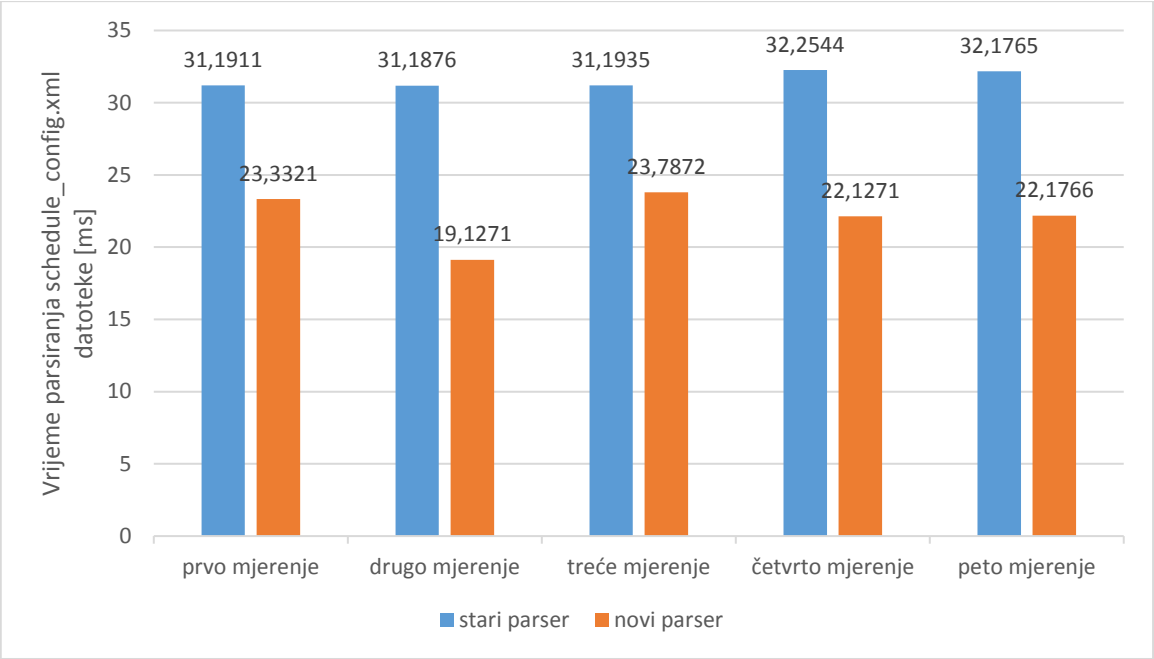
Slika 4.1. Isječak koda za mjerenje vremena parsiranja datoteka

Definirane su dvije varijable *start* i *end*. U njih se zapisuje vrijeme početka te vrijeme završetka parsiranja. Razlika tih vremena ispisuje se na terminal i predstavlja vrijeme parsiranja. Na slici 4.2. nalazi se prikaz ispisa tog vremena u terminalu.



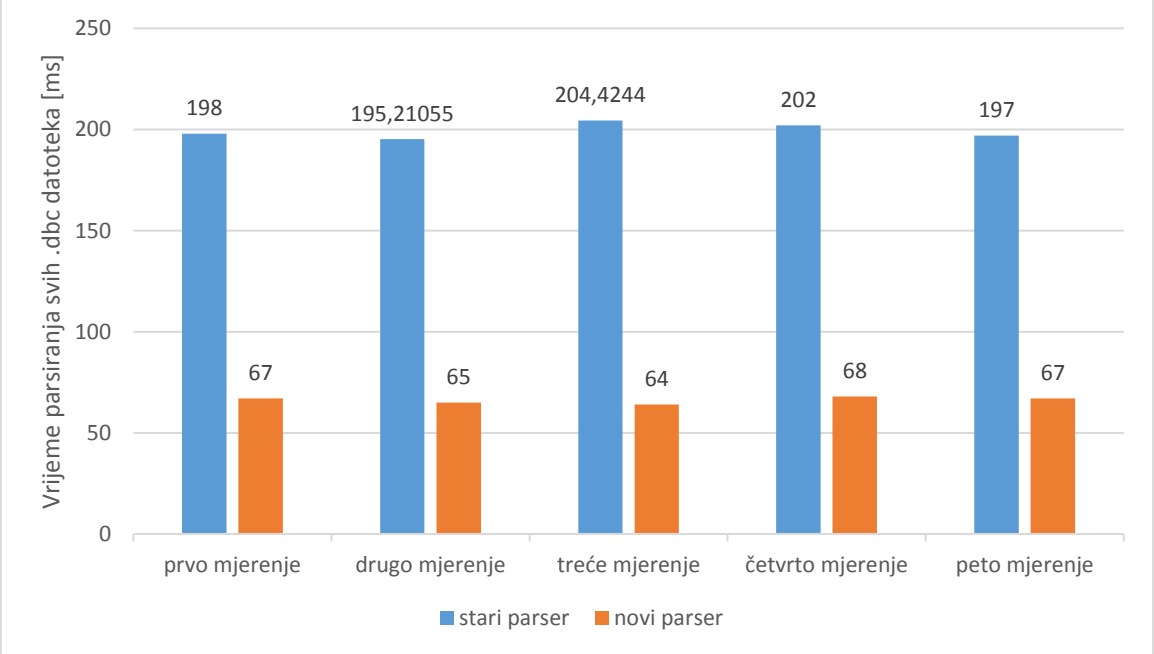
Slika 4.2. Prikaz vremena proteklog za parsiranje *.dbc* i *.xml* datoteka na terminalu

Na slici 4.3. prikaz je proteklog vremena za sva provedena mjerenja trajanja parsiranja *schedule_config.xml* datoteke. Mjereno je vrijeme izvođenja parsiranja po pet puta za tu datoteku. Plavo obojeni stupci predstavljaju vremena dobivena za stari parser, a narančasto obojeni vremena dobivena za novi parser.

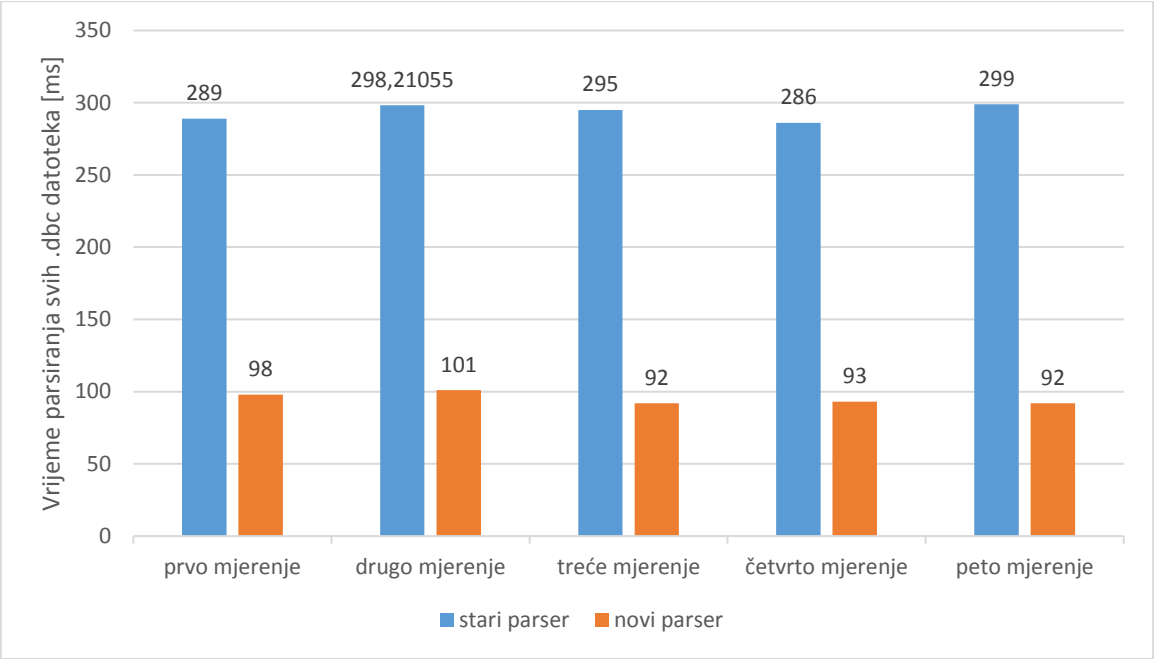


Slika 4.3. Prikaz proteklog vremena za sva provedena mjerenja trajanja parsiranja *schedule_config.xml* datoteke

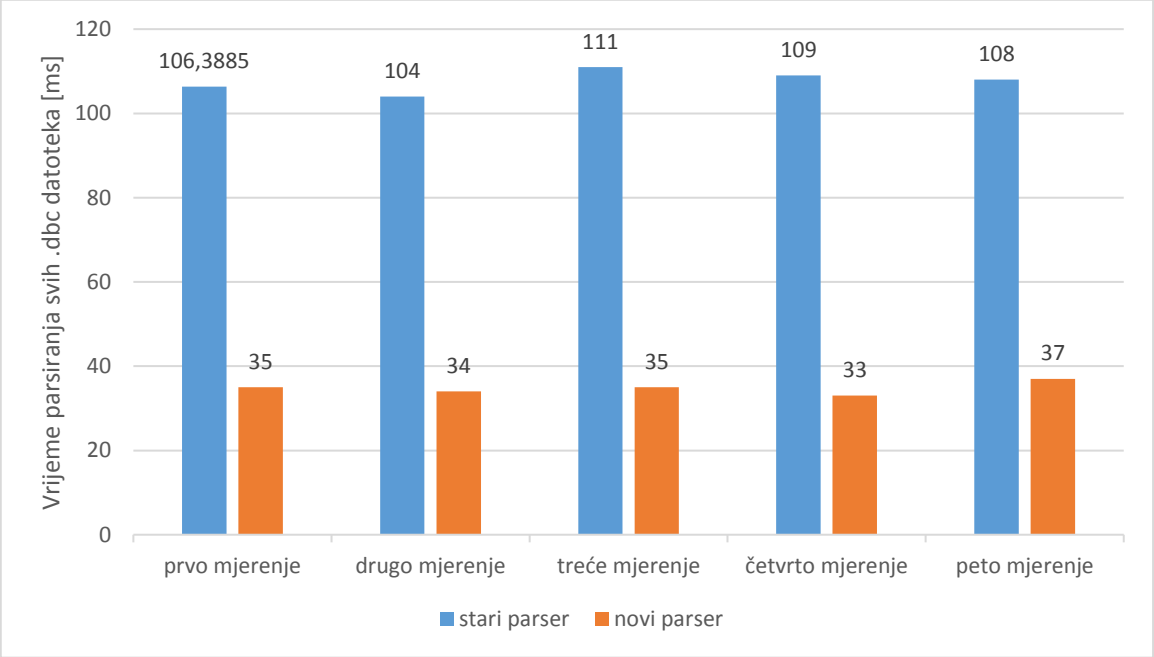
Na slikama 4.4.-4.6. prikaz je proteklog vremena za sva provedena mjerenja trajanja parsiranja *.dbc* datoteka (*TTADrive.dbc*, *TTADrive_Misc.dbc*, *TTADrive_Object.dbc*). Mjereno je vrijeme izvođenja parsiranja po pet puta za svaku datoteku. Plavo obojani stupci predstavljaju vremena dobivena za stari parser, a narančasto obojani vremena dobivena za novi parser.



Slika 4.4. Prikaz proteklog vremena za sva provedena mjerenja trajanja parsiranja *TTADrive.dbc* datoteke

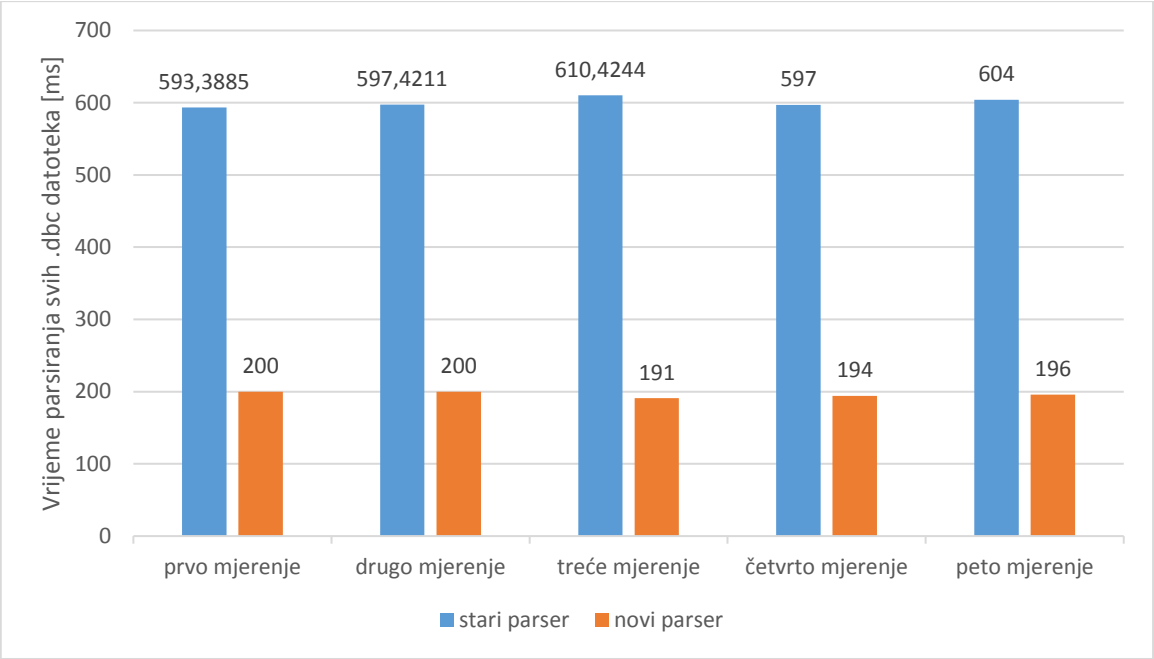


Slika 4.5. Prikaz proteklog vremena za sva provedena mjerenja trajanja parsiranja *TTADrive_Misc.dbc* datoteke



Slika 4.6. Prikaz proteklog vremena za sva provedena mjerenja trajanja parsiranja *TTADrive_Object.dbc* datoteke

Na slici 4.7. prikaz je proteklog vremena za sva provedena mjerenja trajanja parsiranja svih *.dbc* datoteka (kada se mjerilo parsiranje svih triju *.dbc* datoteka odjednom).



Slika 4.7. Prikaz proteklog vremena za sva provedena mjerenja trajanja parsiranja svih *.dbc* datoteka

U tablici 4.2. je prikazano srednje vrijeme parsiranja *.dbc* i *.xml* datoteka u starom i novom TEG-u.

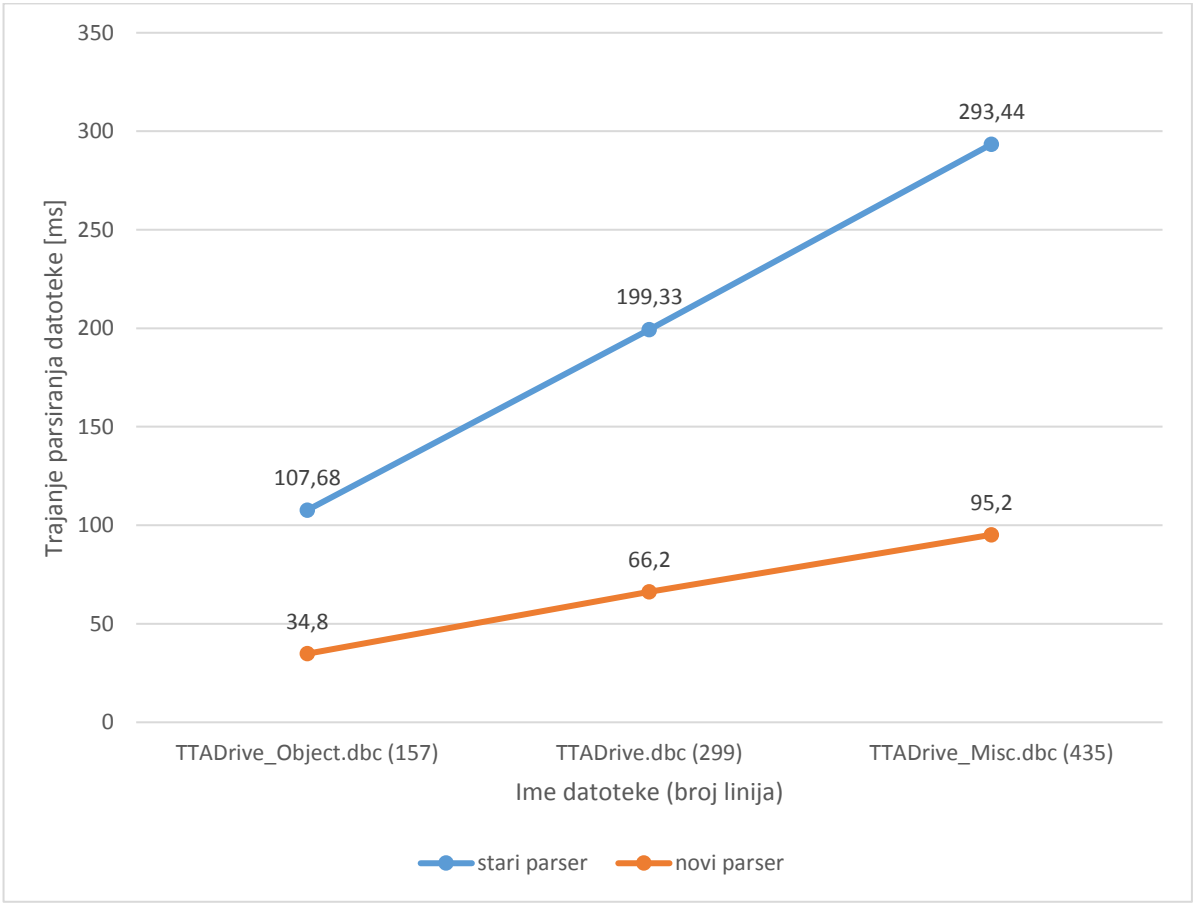
Tablica 4.2. Srednje vrijeme parsiranja *.dbc* i *.xml* datoteka za stari i novi parser [ms]

Naziv datoteke	**Srednje vrijeme parsiranja - stari parser [ms]**	**Srednje vrijeme parsiranja - novi parser [ms]**
schedule_config.xml	31,60	22,11
TTADrive.dbc	199,33	66,20
TTADrive_Misc.dbc	293,44	95,20
TTADrive_Object.dbc	107,68	34,80
sve *.dbc* datoteke zajedno	600,45	196,20

Ukupno srednje vrijeme parsiranja *.dbc* i *.xml* datoteka, koje je zbroj srednjeg vremena parsiranja *schedule_config.xml*, *TTADrive.dbc*, *TTADrive_Misc.dbc* i *TTADrive_Object.dbc* datoteka iznosi 632,05 milisekundi za stari parser te 218,31 milisekundi za novi parser. Kada se usporedi vrijeme parsiranja *.xml* datoteke unutar starog i novog TEG-a može se uočiti da novi parser postiže bolje rezultate odnosno brži je kod parsiranja ove vrste datoteka. Isti slučaj je i kod

.dbc formata. Novi parser brže parsira ovu vrstu datoteka. Bitno je naglasiti da na brzinu parsiranja kod starog i kod novog parsera utječe veličina datoteke koju je potrebno parsirati, odnosno vrijeme trajanja parsiranja ovisi o broju linija datoteke. Datoteke koje su veće sadrže veći broj linija te su zahtjevnije za obradu. Potrebno je proći kroz više podataka stoga je i vrijeme parsiranja veće.

Ovisnost trajanja parsiranja o broju linija *.dbc* datoteke koja se parsira prikazana je na slici 4.8. Datoteke su na x-osi poredane po veličini, od datoteke s najmanjim brojem linija (*TTADrive_Object.dbc)* do one s najvećim brojem linija (*TTADrive_Misc.dbc)*. Y-os predstavlja vrijeme parsiranja *.dbc* datoteke. Plava linija koja predstavlja rezultate mjerenja vremena parsiranja starog parsera strmija je i poprima veće vrijednosti nego narančasta linija koja predstavlja trend vremena parsiranja novog parsera. To znači da, osim što se datoteke brže parsiraju novim parserom, vrijeme parsiranja sporije raste s povećanjem broja linija datoteke nego u starom parseru i stoga je novo rješenje prikladnije za parsiranje većih datoteka.



Slika 4.8. Ovisnost trajanja parsiranja o broju linija datoteke

4.2.2. Rezultati testiranja ispravnosti zapisa u bazu podataka

Za provjeru ispravnosti spremanja u bazu podataka može se kao primjer uzeti podatak iz *.xml* datoteke prikazan na slici 4.9.

```
<RUNNABLE type="PLT">
      <RUNNABLE-ID accessLevel="readOnly">72</RUNNABLE-ID>
      <WCET unit="ms" accessLevel="readOnly">2</WCET>
      <FULL-RUNNABLE-NAME>
        <HOST-ID accessLevel="readOnly">5</HOST-ID>
        <RUNNABLE-NAME accessLevel="readOnly">REthComXX_TX</RUNNABLE-NAME>
        <SWC-NAME accessLevel="readOnly">CtApEthernetComXX</SWC-NAME>
      </FULL-RUNNABLE-NAME>
</RUNNABLE>
```

Slika 4.9. Prikaz jedne *runnable* komponente koju je potrebno parsirati iz ulazne komunikacijske matrice

Runnable REthComXX_TX izdvaja se iz komunikacijske matrice s nazivom softverske komponente (SWC-NAME), vremenom najgoreg izvođenja (WCET) i prosječnim vremenom izvođenja (AET). U bazi se pronalazi isti takav podatak koristeći ime softverske komponente i *runnable*-a, a vremena se onda vežu uz tu komponentu. Na slici 4.10. na primjeru ovog podatka može se vidjeti da je izdvajanje podatka iz komunikacijske matrice, traženje tog istog podatka u glavnoj bazi podataka te na kraju popunjavanje te baze uspješno odrađeno.

```
ComponentTypes<|>CtApEthernetComXX<|>CtApEthernetComXX_Behavior
<|>REthComXX_TX<|>+
AET<|>1.000000
WCET<|>2.000000<|>@
```

Slika 4.10. Prikaz popunjenog dijela glavne baze podataka s podacima o *runnable REthComXX_TX* komponenti izdvojenih iz ulazne komunikacijske matrice

Glavna baza podatka ima razumljivu i lako čitljivu strukturu, a samo ubacivanje podataka vrlo je jednostavno. Potrebno je samo usporediti naziv izdvojenog podatka s onim koji se nalazi u bazi.

Kada se usporede rezultati, vidljivo je da je novo rješenje ispunilo tražene zahtjeve ovog diplomskog rada te da se novi parser TEG-a brže izvodi u odnosu na postojeći odnosno postiže manja vremena parsiranja svih vrsta datoteka. To je rezultat korištenja C++ programskog jezika koji je brži od Python programskog jezika kao i bolje forme baze podataka. Više se ne stvaraju predefinirani objekti i grupiranje podataka, već se baza kreirala na osnovu *.arxml* datoteke pomoću model parsera.

Također, kod *.xml* formata nema prevelikog ubrzanja zbog manje veličine datoteka, ali moguće i zbog korištenja dodatne biblioteke koja sama po sebi usporava pretraživanje. Tu se ostavlja prostor za kreiranje vlastitog parsera bez korištenja vanjskih biblioteka, već osmišljavanjem vlastitog algoritma pretraživanja. Kod *.dbc* formata implementiran je vlastiti parser bez dodatnih biblioteka koji se pokazao efikasnim, jer je vrijeme parsiranja tih vrsta datoteka znatno smanjeno.

Pretraga podataka vrši se liniju po liniju sve dok se ne dođe do kraja datoteke ili dok traženi podatak nije pronađen, što znatno utječe na vrijeme parsiranja, pogotovo u slučaju kad je ulazna datoteka parsera velika odnosno sadrži velik broj linija. Također, velik broj linija glavne baze podataka usporava traženje potrebnih podataka. S ciljem poboljšanja navedenih pretraživanja, moguće je podijeliti datoteku po zadanom kriteriju na više smislenih cjelina. U tom slučaju bi se pretraživao samo jedan određeni dio, a ne cijela datoteka, što bi smanjilo ukupno vrijeme potrebno za parsiranje *.dbc* i *.xml* datoteka.

5. ZAKLJUČAK

Testiranje komunikacije između komponenata automobila vrlo je važan dio testiranja automobilskih komponenti koje se planiraju koristiti u automobilima jer se testiranjem osigurava ispravnost i pouzdanost komponenata, ali i vozila u cjelini. Iz tog razloga nastao je TEG koji će obaviti testiranje komunikacije unutar vozila.

Cilj diplomskog rada bio je izraditi novo programsko rješenje parsera unutar TEG-a na osnovu ustupljenog postojećeg rješenja, a pritom pokušati što više smanjiti broj nedostataka. To se ponajprije odnosi na implementaciju novog rješenju u C++ programskom jeziku u svrhu ubrzanja i komercijalizacije programskog rješenja, zatim postizanje univerzalnosti u smislu mogućnosti primjene istog programskog koda na više različitih komunikacijskih matrica, odnosno datoteka, te na kraju pohrana podataka u novu bazu podataka koja je također univerzalna, odnosno primjenjiva na više različitih modela komunikacije.

S obzirom na postavljene zahtjeve, nova verzija TEG-a uspješno je implementirana i unaprijeđena u odnosu na prethodnu verziju. Parser kao dio TEG-a postigao je svoju univerzalnost te je ubrzano vrijeme parsiranja podataka iz svih datoteka, a podaci su uspješno pohranjeni u točno određeni dio baze podataka, što je i pokazano u prethodnom poglavlju u kojem su pokazana i opisana sva testiranja provedena na novom programskom rješenju TEG-a. Uz sva poboljšanja, program je sada moguće i komercijalizirati jer se lako može prevesti u nečitljiv oblik i takav ponuditi kupcima.

U smislu daljnjeg napretka, moguće je kreirati vlastiti parser *.xml* datoteka da bi se postigla neovisnost o drugim bibliotekama i moguće ubrzanje parsiranja. I dalje ostaje prostora za skraćenje koda budući da je na projektu radilo više osoba. Stoga su moguća ponavljanja nekih dijelova koda odnosno funkcija određene namjene. Nadalje, parserom opisanim u ovom diplomskom radu nisu obuhvaćene sve vrste datoteka koje se mogu pojaviti na ulazu u parser. Stoga je potrebno pronaći pogodno programsko rješenje i za ostale vrste datoteka na sličan način kakav je opisan u ovom radu.

LITERATURA

[1] Što je to AUTOSAR?, https://www.globallogic.com/hr/about/news/sto-je-to-autosar/ (pristup 07.07.2021.)

[2] Kapular, B., *Testiranje verifikatora modela kod izgradnje programske podrške u automotiv industriji*, Diplomski rad, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Sveučilište u Osijeku, 2020.

[3] D. Wang, J. Kuang and W. Tan, "Conformance testing for the car lights system based on AUTOSAR standard," 2011 IEEE 3rd International Conference on Communication Software and Networks, 2011, pp. 345-348, doi: 10.1109/ICCSN.2011.6013608.

[4] P. Caliebe, C. Lauer and R. German, "Flexible integration testing of automotive ECUs by combining AUTOSAR and XCP," 2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE), 2011, pp. 67-72, doi: 10.1109/ICCAIE.2011.6162106.

[5] N. Englisch, F. Hänchen, F. Ullmann, A. Masrur and W. Hardt, "Application-Driven Evaluation of AUTOSAR Basic Software on Modern ECUs," 2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing, 2015, pp. 60-67, doi: 10.1109/EUC.2015.31.

[6] A. Mihalj, R. Grbić, N. Lukić and Z. Kaprocki, "Code Generator for ADAS Software Testing," 2020 Zooming Innovation in Consumer Technologies Conference (ZINC), 2020, pp. 184-189, doi: 10.1109/ZINC50678.2020.9161801.

[7] Mihalj, A., *Generator softverskog koda namijenjenog za testiranje ADAS sustava*, Diplomski rad, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Sveučilište u Osijeku, 2019.

[8] C programming language, https://www.techopedia.com/definition/26184/c-programming-language (pristup 16.7.2021.)

[9] Microsoft Visual Studio, https://visualstudio.microsoft.com/vs/ (pristup 18.7.2021.)

[10] Tinyxml2, https://github.com/leethomason/tinyxml2 (pristup 19.7.2021.)

[11] Vector Informatik GmbH, „DBC File Format Documentation“, 2007.

[12] XML, https://www.w3.org/standards/xml/core (pristup 19.7.2021.)

[13] Chrono in C, https://www.geeksforgeeks.org/chrono-in-c/ (pristup 22.7.2021.)

SAŽETAK

S ciljem osiguranja sigurnosti i kvalitete automobila, provodi se niz sigurnosnih testova kako bi se utvrdila ispravnost svih njegovih dijelova. U ovom radu teorijski je obrađeno nekoliko načina testiranja od kojih je jedan generator testnog okruženja (engl. *Test Environment Generator* – TEG). Na osnovu analize jednog postojećeg rješenja TEG-a i identificiranih nedostataka, implementirano je novo rješenje u C++ programskom jeziku i pritom je smanjen broj mana postojećeg TEG-a napisanog u Python programskom jeziku. Novo rješenje primjenjivo je na više različitih komunikacijskih matrica, a prikupljeni podaci spremaju se u bazu podataka tako da se i ona može primijeniti na više različitih modela komunikacije. Funkcionalnost novog rješenja potvrđena je na jednoj *.xml* i trima *.dbc* datotekama te su na tim datotekama vršena ocjenjivanja performansi dobivenog rješenja koja su pokazala da je vrijeme parsiranja smanjeno u prosjeku oko 1,5 puta za *.xml* format datoteka te 3 puta za *.dbc* format datoteka.

Ključne riječi: TEG, parser, AUTOSAR, testiranje, ECU, C++, komunikacijska matrica

DEVELOPMENT OF A PARSER FOR A COMMUNICATION MATRIX WITH A FOCUS ON DBC AND XML FORMATS WITHIN THE TEST ENVIRONMENT GENERATOR

ABSTRACT

To ensure the safety and quality of the car, a series of safety tests are carried out to determine the correctness of all its parts. This paper theoretically deals with several test methods, one of which is the test environment generator (TEG). Based on the analysis of one existing TEG solution and the identified shortcomings, a new solution was implemented in the C ++ programming language, while reducing the number of shortcomings of the existing TEG written in the Python programming language. The new solution is applicable to several different communication matrices, and the collected data is stored in a database in such way that it can also be applied to several different communication models. The functionality of the new solution was confirmed on one *.xml* and three *.dbc* files, and the performance of the obtained solution was evaluated on these files, which showed that the parsing time was reduced by an average of about 1.5 times for *.xml* file format and 3 times for. *dbc* file format.

Keywords: TEG, parser, AUTOSAR, testing, ECU, C++, communication matrix

ŽIVOTOPIS

Magdalena Tomašić rođena je u Vinkovcima 16. lipnja 1996. godine, gdje je završila osnovnu i srednju školu. Gimnaziju Matije Antuna Reljkovića završila je 2015. godine, smjer Jezična gimnazija, te iste godine upisuje Elektrotehnički fakultet u Osijeku, danas Fakultet elektrotehnike, računarstva i informacijskih tehnologija. Nakon završenog Preddiplomskog sveučilišnog studija računarstva upisuje Diplomski studij računarstva na istom fakultetu gdje trenutno pohađa drugu godinu smjera Robotika i umjetna inteligencija. Istovremeno je stipendist Instituta RT-RK u Osijeku uz čije je mentorstvo i napravljen ovaj diplomski rad.

_____

Potpis