Testiranje verifikatora modela kod izgradnje programske podrške u automotiv industriji

Kapular, Bruno

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

Sveučilišni studij

Testiranje verifikatora modela kod izgradnje programske podrške u automotiv industriji

Diplomski rad

Bruno Kapular

Osijek, 2020.

SADRŽAJ

1. UVOD

Svaki ispravni automobil bi trebao moći prevesti osobu od točke A do točke B. Uz osnovnu potrebu da taj put sa svakim novim modelom bude što brži i efikasniji, sve veća pažnja pridaje se i tome da putovanje bude udobnije i sigurnije. Iskorištavanje najnovijih tehnologija, kako bi se vozaču što više olakšalo upravljanje vozilom, iako jednostavnije i sigurnije za vozača, dovodi do razvoja sve složenijih komponenata koje se ugrađuju u sve veće sustave, koji opet imaju potrebu za složenom programskom podrškom. Unatoč tome što se izrada složenih sustava donekle olakšala uvođenjem standardizacije i alata za automatsko pisanje programskog koda u industriju, još uvijek ostaje problem provjere ispravnosti istih.

Kako je ručno pisanje programskih testova za velike sustave iznimno dugotrajno, sve se više teži automatizaciji tog procesa. Upravo zbog toga se razvija alat za automatsku provjeru modela rađenih po AUTOSAR standardu [1], koji se zove SDV *(*engl. *System definition verifier)*.

SDV ima svoje zahtjeve definirane u dokumentu koji se zove SDG (engl. *System definition guide*). Zbog složenosti alata i broja zahtjeva definiranih u SDG-u, u okviru ovog diplomskog rada izrađen je testni skup modela, čija je svrha ispitati koliko se alat pridržava opisanih zahtjeva. Također je izrađeno i grafičko sučelje, kako bi se olakšalo služenje alatom i povećale mogućnosti i lakoća statističke obrade dobivenih rezultata.

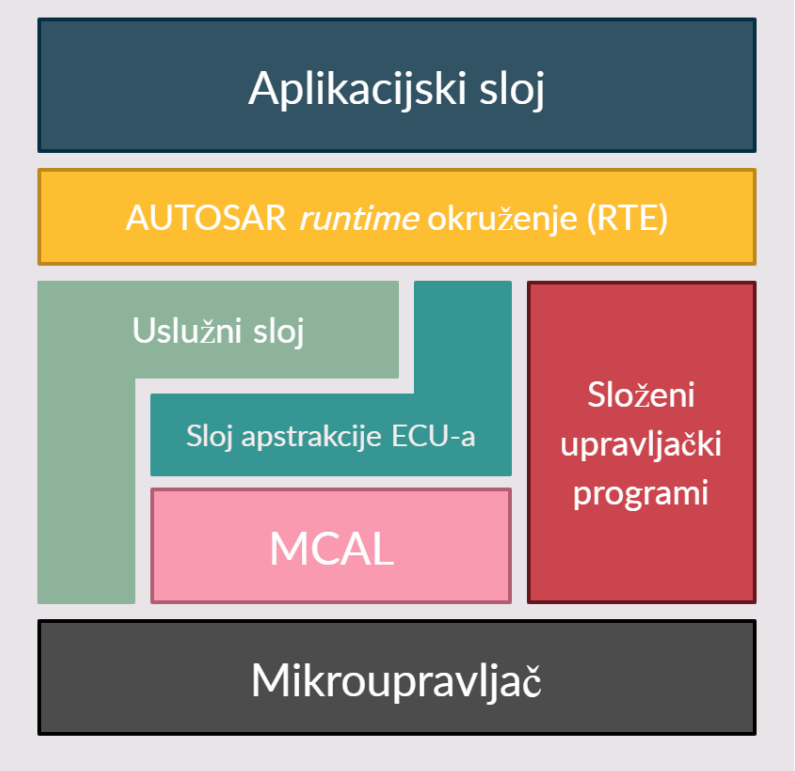
Diplomski rad podijeljen je u pet poglavlja. U drugom poglavlju opisana je arhitektura pomoću koje su rađeni modeli koje alat provjerava. U istom je poglavlju također opisano i kako se inače provjerava sigurnost alata u automotiv industriji. U trećem poglavlju je opisano rješenje napravljeno u sklopu ovog diplomskog rada, koraci njegove izrada te alati korišteni za njegovu izradu. U četvrtom poglavlju opisani su rezultati testiranja provedenog rješenjem predloženim u trećem poglavlju. U petom poglavlju izneseni su zaključci o rješenju i alatu koji se testirao.

2. ISPITIVANJE ALATA ZA PROVJERU MODELA U AUTOMOTIV INDUSTRIJI

U ovom je poglavlju ukratko dan malo širi pogled na područje kojim se ovaj diplomski rad bavi. Opisana je arhitektura u kojoj su rađeni testni modeli koji su korišteni za ispitivanje SDV-a. Također je navedeno zašto je odabran baš taj određeni način testiranja, te koji su drugi načini mogući ili čak potrebni za potpunu evaluaciju alata.

2.1. AUTOSAR

AUTOSAR (engl. *AUTomotive Open System ARchitecture*) je otvorena arhitektura programske podrške razvijena za korištenje u automotiv industriji [1]. Osnovni oblik AUTOSAR arhitekture može se vidjeti na slici 2.1 [2].

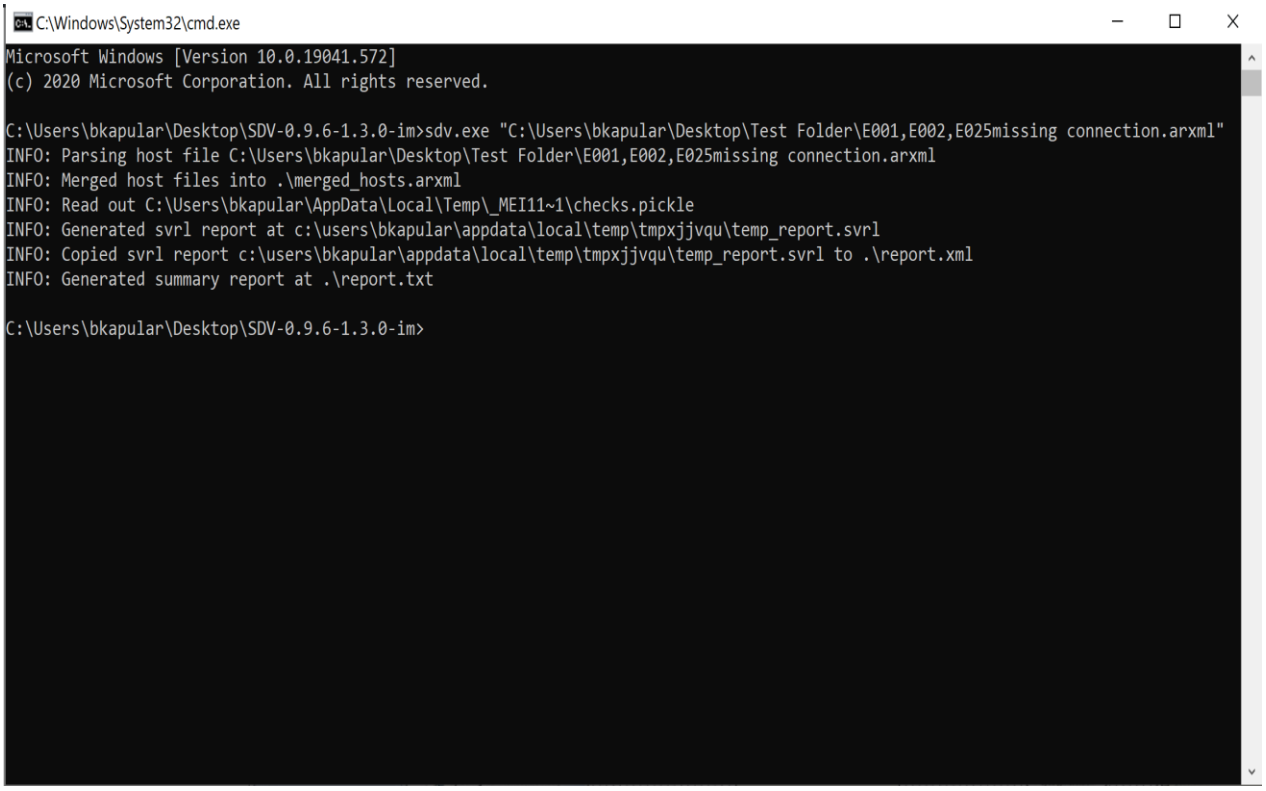


Slika 2.1 Pojednostavljeni prikaz AUTOSAR arhitekture [2]

Njegova je svrha pružiti standardizirana sučelja koja olakšavaju stvaranje fleksibilne programske podrške, koja se lako može prilagoditi i ponovno iskoristiti u novim projektima [1].

2.2. Verifikator modela sustava

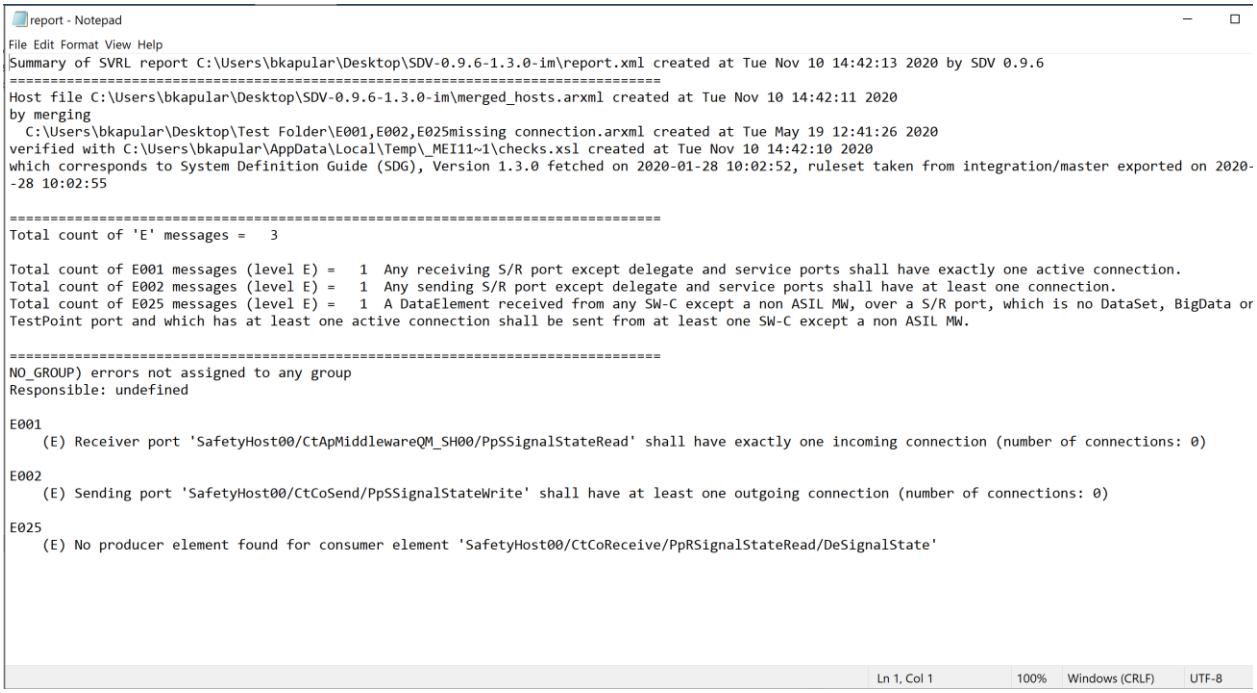
Modeli rađeni prema AUTOSAR-u opisuju se pomoću AUTOSAR XML (engl. *Extensible Markup Language*) sheme [3]. Kako su ti modeli često jako veliki, a XML nepregledan, da bi se izbjegla dugotrajna ručna provjera modela, potrebno je razviti automatski alat za provjeru istih. SDV je verifikator modela sustava (engl. *System model verifier*) koji provjerava modele prema jasno definiranom skupu pravila koja su opisana u SDG-u. U trenutnom zatečenom stanju (na samom početku izrade ovog rada), alat se koristi na najosnovniji način: preko upravljačkog prozora (slika 2.2).



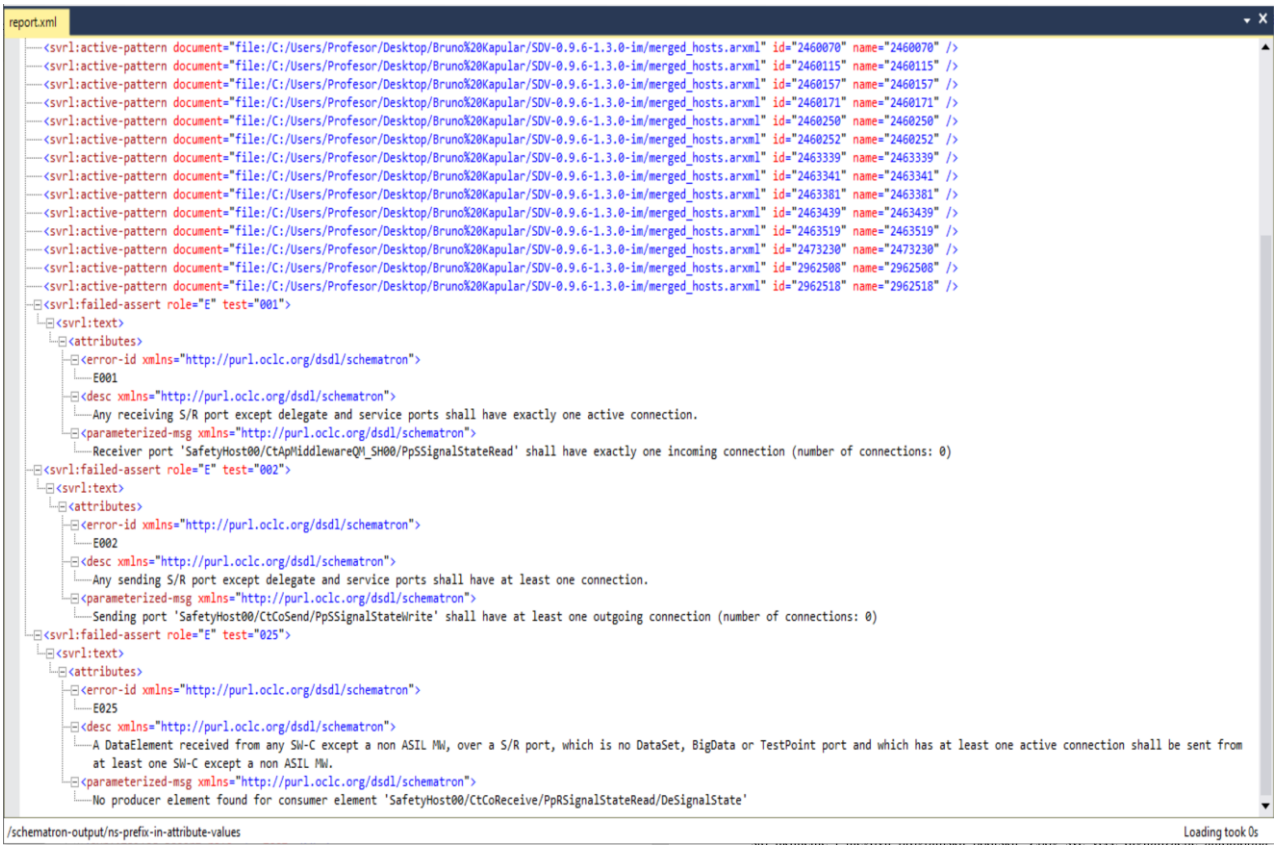
Slika 2.2 Upravljački prozor u kojem je prikazana naredba za pokretanje SDV-a

SDV-u se jednostavno dodjeljuje putanja do jednog ili više modela koje treba provjeriti, nakon čega ih on spaja i provjerava. SDV ispisuje izvještaj u dvama različitim formatima: .xml i .txt. U tekstualnoj datoteci ispisuje listu i broj svih grešaka (slika 2.3).

XML izvještaj sadrži nešto više korisnih detalja, među kojima je možda najkorisnija lista svih testova. No zbog teže čitljivosti, nezgodan je za korištenje pri ručnoj analizi, što se može vidjeti na slici 2.4 koja prikazuje izvještaj za isti model za koji je on prikazan i slikom 2.3. Međutim, zbog prirode samog formata, olakšava izvlačenje potrebnih informacija, što će biti detaljnije objašnjeno u trećem poglavlju.



Slika 2.3 Prikaz izvještaja u .txt formatu kojeg generira SDV



Slika 2.4 Prikaz izvještaja u .xml formatu kojeg generira SDV

2.3. Određivanje potrebe za kvalifikacijom alata

Zbog prirode same automotiv industrije, iznimno je važno da sve komponente automobila budu ispravne, što uključuje i njegovu programsku podršku. Zbog sve veće digitalizacije automobila, raste vjerojatnost da do greške dođe u samoj programskoj podršci, a ne u fizičkim komponentama. Zbog rastuće potrebe za brzim i efikasnim ispitivanjem modela, sve se više radi na alatima koji to rade automatski, kao što su SystemDesk [4] ili SDV. Međutim, što se više industrija oslanja na takve alate, to je važnije da i ti alati imaju visoku razinu pouzdanosti, što znači da i oni moraju proći kroz strogo ispitivanje. Prvi korak u kvalifikaciji alata je odrediti treba li alat uopće kvalificirati. Proces određivanja potrebe kvalifikacije alata definiran je ISO26262 standardom [5] i ima tri koraka [6].

2.3.1. Analiza utjecaja alata

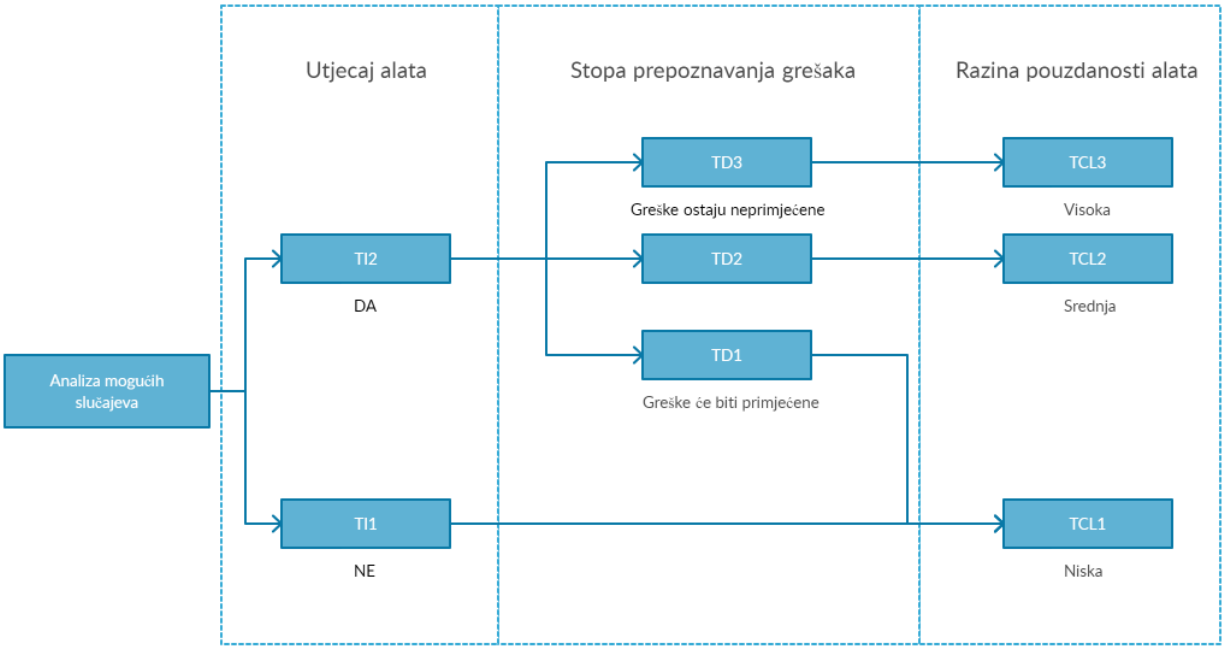
Najjednostavniji način za određivanje potrebe za razinom kvalifikacije alata je taj da se postavi pitanje ima li alat utjecaj na konačni proizvod. U slučajevima kada utjecaj alata (engl. *Tool impact* ili TI) na konačni proizvod ne postoji, može ga se svrstati u kategoriju TI1 [6]. U suprotnom, ako alat može izazvati ili ne primijetiti potencijalne greške koje bi inače trebao, onda mu se može dodijeliti oznaka TI2 [6]. Ako je u ovom koraku alat određen kao TI1, onda ga se odmah može proglasiti alatom najniže razine pouzdanosti ili TCL1 [6].

2.3.2. Analiza stope prepoznavanja grešaka

U slučajevima kada se u prethodnom koraku alat odredi kao TI2, nužno je izvršiti analizu stope prepoznavanja grešaka (engl. *Tool error detection* ili TD). Ovisno o tome kolika je vjerojatnost da se greška alata prepozna u kasnijim fazama procesa, alati se mogu ponovno podijeliti. Ako je vjerojatnost da se greška prepozna velika, onda se može pretpostaviti da nije potrebno imati veliku pouzdanost u alat te se može svrstati u TD1. U suprotnom, alat se svrstava u TD2 ili TD3 (Slika 2.5) [6].

2.3.3. Određivanje razine pouzdanosti alata

Ovisno o podjeli u prošlom koraku, može se odrediti razina pouzdanosti alata (engl. *Tool confidence level* ili TCL). Već u prvom koraku se moglo odrediti da TI1 alati pripadaju alatima niske razine pouzdanosti ili TCL1, međutim tu spadaju i alati koji su u drugom koraku određeni kao TD1 [6]. Alati koji su određeni kao TD2, smatraju se alatima srednje razine pouzdanosti ili TCL2 [6]. TD3 alati smatraju se alatima visoke razine pouzdanosti TCL3 [6].



Slika 2.5 Proces određivanja potrebne razine pouzdanosti alata [6]

2.4. Kvalifikacija alata

Alati koji su određeni kao TCL2 ili TCL3 potrebno je testirati. Postoji nekoliko metoda testiranja prema ISO26262 standardu, a to su [6]:

- Povećanje pouzdanosti kroz korištenje (engl. *Increased confidence from use*)
- Evaluacija razvojnog procesa (engl. *Evaluation of tool development process*)
- Validacija alata (engl. *Tool validation*)
- Razvoj prema sigurnosnom standardu (engl. *Development in accordance with a safety standard*)

Kojom metodom bi se trebalo koristiti, ovisi o ASIL-u (engl. *Automotive Safety Integrity Level*) ili razini integriteta automobilske sigurnosti [6]. ASIL je ključna komponenta u ISO26262 standardu, koja definira potrebnu razinu sigurnosti alata, ovisno o vjerojatnosti pojave kvara i mogućim posljedicama kvara [5].

Postoje četiri ASIL razine: A, B, C i D [5]. Svaka ASIL razina predstavlja potrebu za progresivno većom pouzdanošću alata, počevši od razine A koja zahtjeva najmanju, prema razini D koji zahtjeva najveću pouzdanost [5]. U tablici 2.1 mogu se vidjeti preporučene metode testiranja alata u ovisnosti o njegovoj ASIL razini [6]. Iz tablice 2.1 može se vidjeti da se zapravo

sve metode ispitivanja preporučuju pri kvalifikaciji alata, međutim ovisno o ASIL-u alata, veća se važnost pridodaje određenim metodama.

Tablica 2.1 Preporučene metode testiranja alata u ovisnosti o njegovoj ASIL razini [6]

Metode		ASIL			
		A	B	C	D
1a	Povećanje pouzdanosti kroz korištenje	++	++	+	+
1b	Evaluacija razvojnog procesa	++	++	+	+
1c	Validacija alata	+	+	++	++
1d	Razvoj prema sigurnosnom standardu	+	+	++	++

S obzirom na trenutno stanje dovršenosti postojećeg alata za kvalifikaciju SDV-a, za temu ovog diplomskog rada odabrana je metoda validacije (1c iz tablice 2.1). Metoda validacije podrazumijeva izradu testnog skupa kojim bi se trebali pokriti svi mogući slučajevi korištenja alata [6]. Izrađeni testni skup podrazumijeva skup nekoliko modela koji pokrivaju dobar dio testova prisutnih u verziji **0.9.6** SDV-a, na kojoj je ovaj diplomski rad zasnovan. U sljedećem su poglavlju opisani alati i način izrade spomenutog skupa modela i grafičkog sučelja alata.

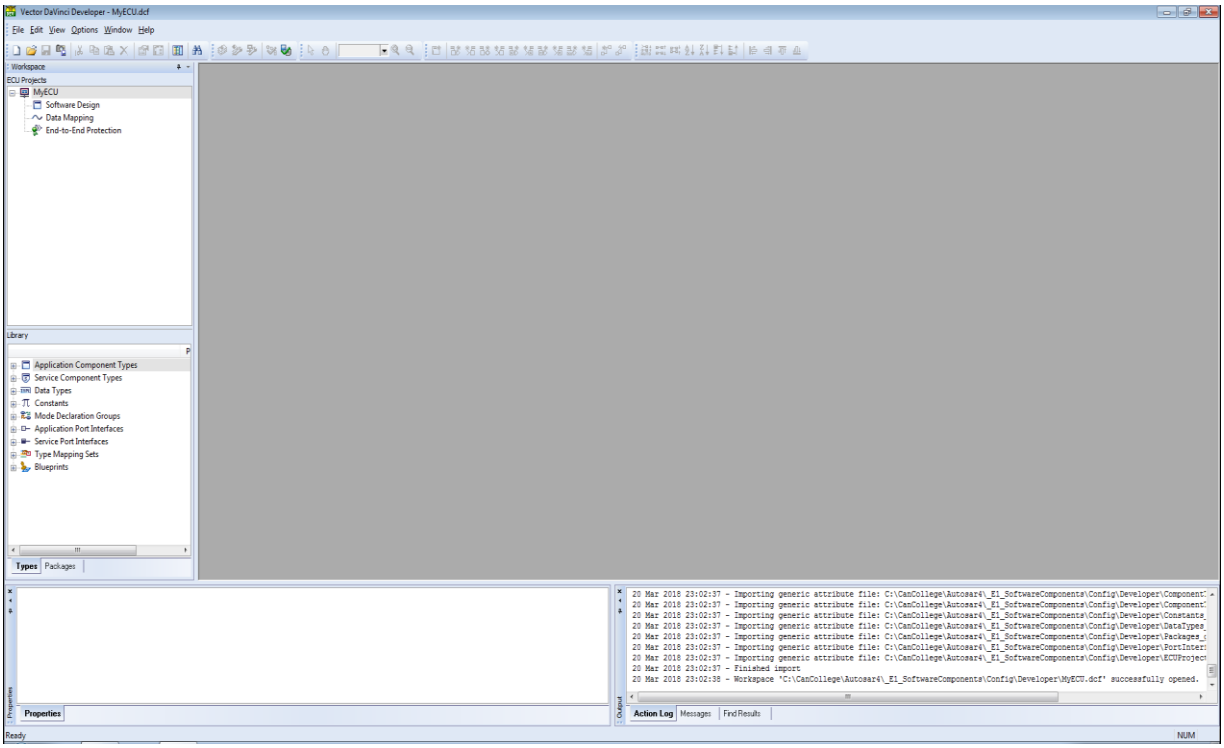
3. IZRADA TESTNOG SKUPA MODELA I GRAFIČKOG SUČELJA SDV ALATA

S obzirom da se od SDV-a očekuje određena razina pouzdanosti zbog činjenice da bi trebao detektirati greške koje potencijalno mogu dovesti do kvara, u sklopu ovog diplomskog rada bilo je potrebno izraditi testni skup modela koji bi trebao ispitati što veći broj mogućih grešaka koje bi SDV trebao prepoznati u svojoj trenutnoj verziji. Također je, zbog trenutno nezgrapnog načina korištenja putem upravljačkog prozora, kao dodatni zahtjev definirana i izrada grafičkog sučelja, koje bi trebalo olakšati korištenje alata i samu analizu rezultata.

3.1. Alati i tehnologije korištene za izradu testnih modela

3.1.1. DaVinci Developer

Za izradu samih modela korišten je alat tvrtke Vector [7] koji se zove DaVinci Developer. Kao što je navedeno u potpoglavlju 2.2, AUTOSAR modeli se spremaju u obliku AUTOSAR XML ili ARXML datoteka. Zbog kompleksnosti ručnog pisanja modela u ARXML formatu, korišten je DaVinci Developer koji taj proces automatizira. Umjesto da korisnik piše XML, na korisniku je samo da definira sve dijelove modela i poveže ih u DaVinci Developeru, koristeći njegovo korisničko sučelje (slika 3.1).



Slika 3.1 Korisničko sučelje DaVinci Developera

Nakon izrade modela, DaVinci Developer može ispisati napravljeni model ili njegove pojedinačne dijelove direktno u ARXML format, koji se onda onda može provjeriti pomoću SDV-a.

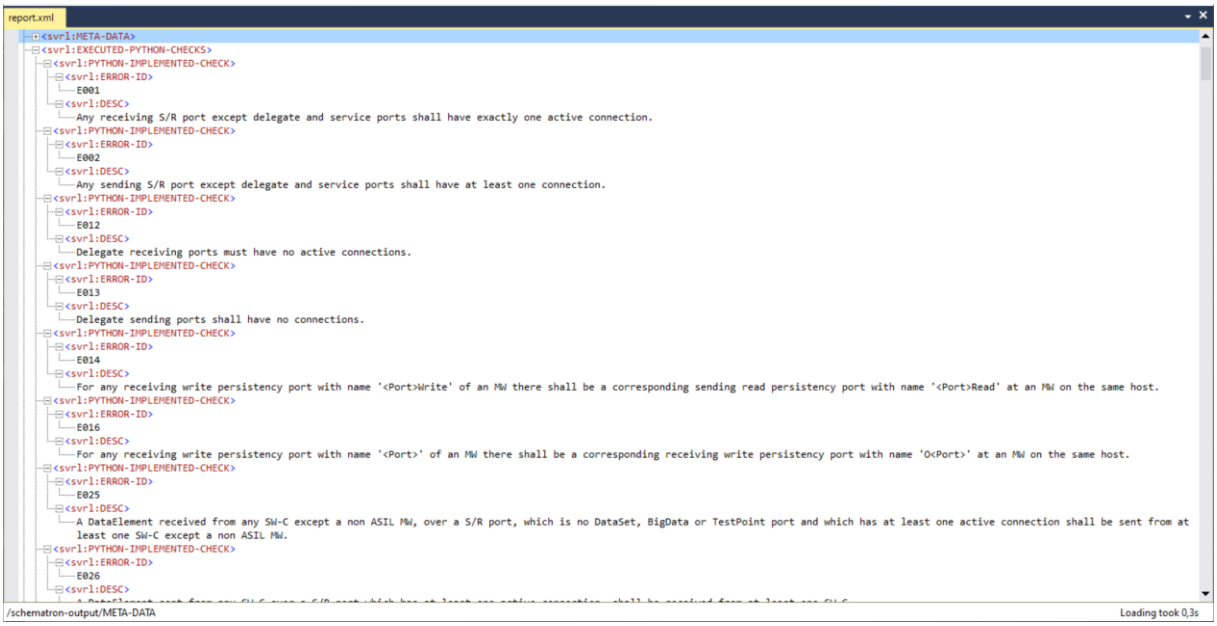
3.1.2. XML

XML je takozvani označni jezik (engl. *markup language*) koji je napravljen za skladištenje i prijenos podataka [8]. Rađen je da bude čitljiv i ljudima i računalu. Svrha XML-a je olakšati dijeljenje i prijenos podataka spremanjem istih u obliku običnog teksta pisanog po određenom skupu pravila [8]. Spremanjem u obliku običnog teksta, omogućava se prijenos i čitanje podataka neovisno o programskoj podršci i sklopovlju računala. Zbog toga je prikladan i za zapisivanje AUTOSAR modela. ARXML format u kojem se modeli spremaju je, zapravo, samo XML format s dodatnim definiranim pravilima za AUTOSAR modele [3].

3.2. Opis izrade testnog skupa modela za validaciju SDV-a

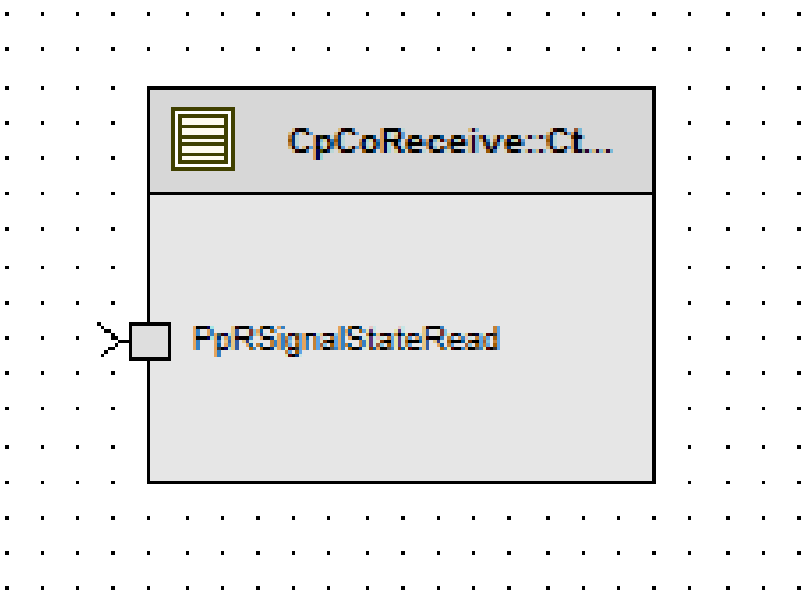
Kao što je navedeno u podpoglavlju 2.4, validacija alata podrazumijeva izradu testnog skupa koji pokriva moguće slučajeve korištenja alata. U slučaju SDV-a, to uključuje izradu dovoljnog broja modela, da bi se provjerila sposobnost alata da uoči pogreške koje bi trebao prepoznati prema njegovim specifikacijama u SDG-u. U sklopu ovog diplomskog rada izrađena su 44 modela koja su poslužila kao testni skup, a više informacija o njima dano je ispod. S obzirom na trenutnu nedovršenost zatečenog alata (u vrijeme izrade ovog diplomskog rada), kako bi se olakšala izrada modela, testni skup je rađen za najnoviju/zadnju verziju alata koja je postojala na početku izrade diplomskog rada, a to je verzija **0.9.6.** Testni skup je rađen isključivo za tu specifičnu verziju, kako bi se izbjegli mogući problemi i potencijalne potrebe za stalnom nadogradnjom modela sa svakom novom verzijom alata koja izađe za vrijeme izrade ovog diplomskog rada.

Proces izrade modela je započeo proučavanjem zahtjeva opisanih u SDG-u. Međutim, s obzirom da nisu svi zahtjevi opisani u SDG-u još implementirani, kao vodilja za izradu potrebnih modela korištena je XML verzija izvješća koje SDV generira, jer u njoj postoji zapis svih implementiranih testova (slika 3.2). Izgradnja svakog modela započela je definiranjem zahtjeva na željeni model. Prvo bi se odabrala jedna greška koju model treba sadržavati, kako bi se provjerilo detektira li ju SDV. Nakon što bi se odabrala greška koju model treba sadržavati, započela bi izrada modela u DaVinci Developeru. Pri izradi modela potrebno je prvo definirati komponente koje model mora sadržavati, da bi se uopće mogla provjeriti prisutnost greške u modelu.



Slika 3.2 Popis implementiranih testova u XML izvješću koje generira SDV

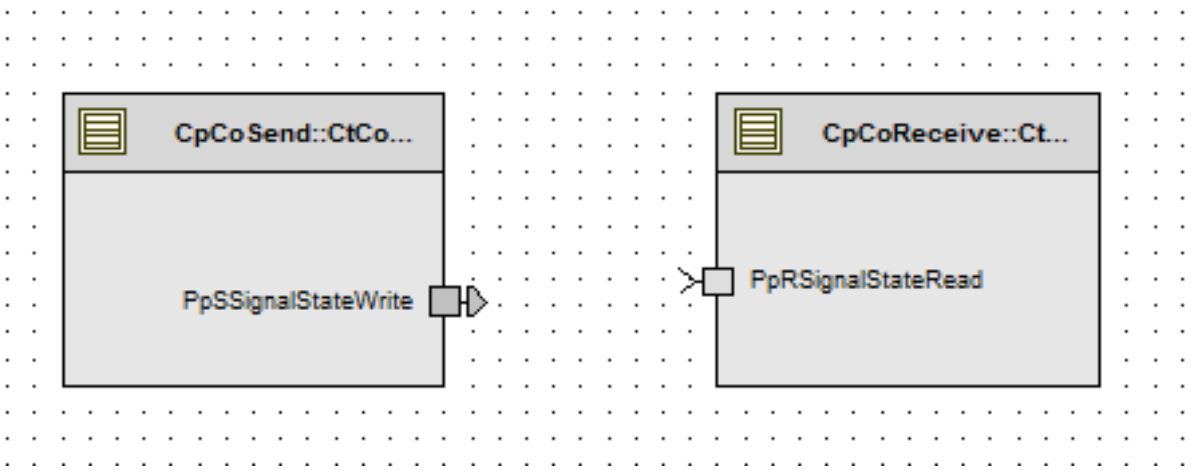
Kako bi se dao nešto bolji uvid u to što neki model mora sadržavati, proces je opisan na primjeru modela koji provjerava grešku E001. Greška E001 javlja se kada priključak za primanje podataka nema točno jednu aktivnu vezu. Da bi se uopće dodao priključak u model, on mora sadržavati komponentu na koju bi se spomenuti priključak dodao. Prema tome, minimum što model koji ispituje pojavu greške E001 mora sadržavati je komponenta i priključak za primanje podataka (slika 3.3). Time su definirani minimalni zahtjevi na sadržaj modela.



Slika 3.3 Vizualni prikaz najjednostavnijeg mogućeg modela rađenog u DaVinci developeru koji sadrži grešku E001

Međutim, kako su modeli rađeni ciljano prema implementiranim testovima, gledalo se da je broj grešaka koji se testira u svakom modelu što manji, tj. minimiziran. Razlog za minimizaciju broja grešaka po modelu je da bi se što više izbjeglo ponavljanje grešaka i time pojednostavila analiza rezultata i praćenje ispitanih testova. Dodavanje samo jedne komponente i samo jednog priključka u model, zbog određenih bi zahtjeva na svaku komponentu rezultiralo puno većim brojem grešaka od samo one na koju se cilja (u ovom slučaju to je E001). Kako bi se broj takvih neželjenih grešaka što više smanjio, potrebno je ispuniti spomenute zahtjeve. Većina komponenata zahtjeva da im se pravilno definira ime, da ih se doda u listu svih komponenata, definiraju procesi s kojima radi i sl. Tek nakon dodavanja općih zahtjeva na svaki model, moglo se fokusirati na izazivanje željene greške.

Iako bi zbog buduće analize bilo najbolje da jedan model služi samo za provjeru jednog zahtjeva, zbog prirode mnogih zahtjeva to nije bilo moguće. Nadalje, nastojalo se da bar donekle modeli mogu sličiti na moguće stvarne situacije. Tako je u slučaju navedenog primjera odlučeno da se umjesto samo jedne greške (E001) ispitaju dvije greške odjednom. Greška E002 javlja se u situacijama kada priključak za slanje nema bar jednu vezu (naspram priključka za primanje u slučaju E001). Prema tome, dodavanje još jedne komponente i relevantnog priključka te izostavljanje veze između priključaka, može simulirati situaciju kada dvije komponente slučajno nisu spojene (slika 3.4).



Slika 3.4 Vizualni prikaz modela rađenog u DaVinci Developeru za provjeru sposobnosti SDV-a da detektira greške E001 i E002

Verifikacijom opisanog modela, SDV javlja tri greške. Javlja već spomenute greške E001 i E002. Uz njih javlja i treću grešku, E025. Navedena se greška javlja kada se definirani podatak nigdje ne šalje. Kako se u sklopu simulacije potencijalno stvarne situacije željelo definirati sve što bi model u takvoj situaciji trebao sadržavati, to je uključivalo i definiranje nekog podatka koji bi

se inače slao. Kako je prisutnost treće greške logički smislena i SDV je uspješno detektirao greške koje je trebao detektirati, može se reći da je uspješno kreiran model čija je svrha provjeriti greške E001, E002 i E025.

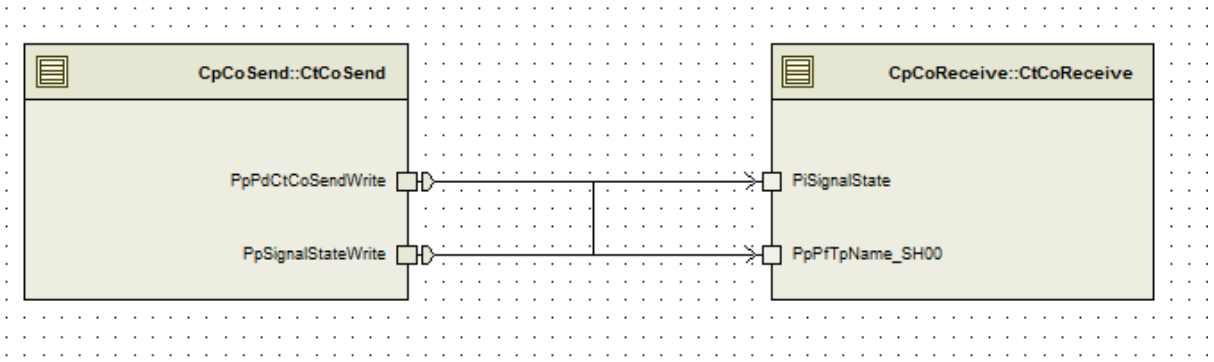
Modeli su rađeni od jednostavnih prema složenijima. Štoviše, gotovo svaki model napravljen poslije prvog, nadogradnja je nekog drugog modela. Time se pojednostavila izrada novih i složenijih modela, jer se nadogradnjom izbjegava potreba za ponovnim dodavanjem komponenata koje bi svaki model trebao sadržavati. Također, dobar je to način za potvrdu konzistentnosti alata, jer dodavanje novih pravilnih komponenata ne bi trebalo dovesti do problema s već provjerenim, starim, komponentama s kojima nisu ni u kakvoj interakciji.

Kao primjer korištenja starog modela za izgradnju novog, može se uzeti model koji je direktna nadogradnja prošlog. Proučavanjem zahtjeva definiranih u SDG-u, odlučeno je napraviti model koji sadrži grešku E032. Greška E032 javlja se u slučajevima kada bilo koji priključak koji se ne na nalazi na međukomponenti i ne služi za obradu skupova podataka, velikih podataka i testnih podataka, ima više od jedne aktivne veze. Prema opisu greške može se vrlo jednostavno zaključiti što mora sadržavati model.

Da bi uopće bilo moguće spojiti više od jedne veze na jedan priključak, potrebna su bar tri priključka. Priključci mogu biti u kombinaciji koja sadrži dva priključka koja šalju i jedan koji prima njihove podatke, ili jedan koji šalje podatke na dva. Kako bi se provjerile obje situacije, odlučeno je na istom modelu uključiti obje kombinacije odjednom. Iako je moguće ispitati prepoznavanje greške E032 tako da se svi priključci naprave na jednoj komponenti, kako to nije slučaj u praksi, odlučeno je da su potrebne bar dvije komponente.

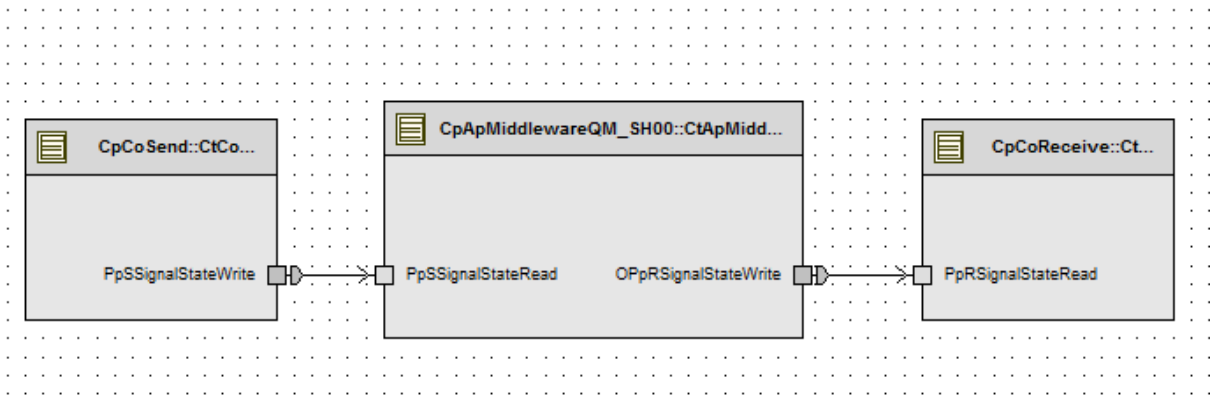
Nakon definicije potrebnih zahtjeva, primjećeno je da je vrlo sličan model već napravljen. Model koji je naveden kao prošli primjer već sadrži dvije komponente i dva priključka. Uz to već zadovoljava opće zahtjeve na navedene komponente. Prema tome, logično je samo napraviti model koji je njegova nadogradnja.

Nadalje kako je prema SDG-u definirano da dvije komponente (uz neke iznimke) ne smiju biti povezane direktnom vezom, već moraju biti povezane preko međukomponente (engl. *Middleware*), pružila se prilika za ubacivanje još jedne greške. Greška koja se javlja kada navedeni uvjet nije zadovoljen je greška E028. S obzirom da za tu grešku nije potrebno ništa nadodavati u model, na postojeće su komponente samo dodani dodatni priključci, nakon čega su napravljene sve moguće veze između njih (slika 3.5).



Slika 3.5 Vizualni prikaz modela rađenog u DaVinci Developeru za provjeru sposobnosti SDV-a da detektira greške E028 i E032

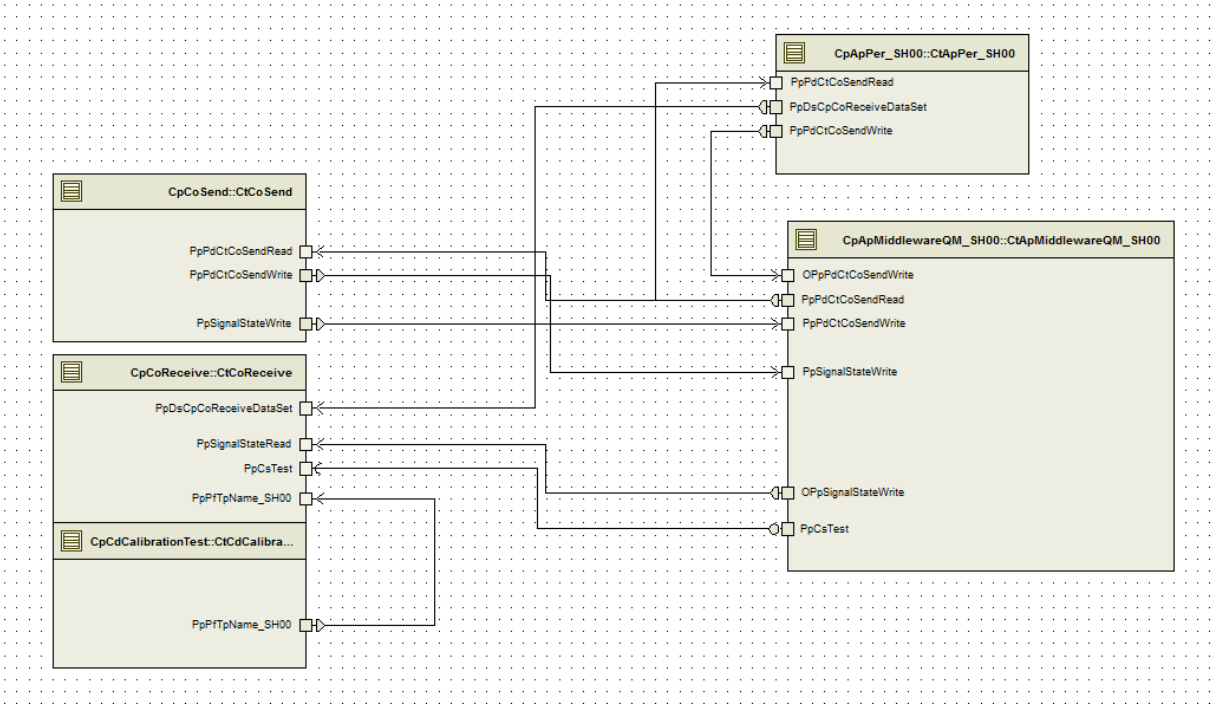
Verifikacijom modela SDV je ispisao tri greške umjesto dvije za koje je model definiran. Uz greške E028 i E032, SDV je ponovno prepoznao grešku E001. S obzirom da se definicije grešaka E001 i E032 preklapaju, u ovoj specifičnoj situaciji ta greška je očekivana i ne narušava ispravnost modela. Valja ponovno naglasiti da ponavljanje grešaka u mnogim slučajevima nije moguće spriječiti. Kao što se može vidjeti u ovom specifičnom slučaju, nekada se zahtjevi opisani u SDG-u preklapaju. Mnogi su zahtjevi definirani iz više različitih perspektiva, zbog čega se neke greške uvjek javljaju zajedno. Kako bi se potvrdila ispravnost SDV-a, bilo je potrebno provjeriti i hoće li prepoznavati greške na pravilnom modelu tamo gdje ih nema. Da bi se to utvrdilo, ponovno je iskorišten model sa slike 3.4. U njega je dodana međukomponenta spomenuta u opisu greške E028, nakon čega su već postojeće komponente spojene na nju (slika 3.6). Verifikacijom rezultantnog modela, SDV nije javio nijednu grešku.



Slika 3.6 Vizualni prikaz modela rađenog u DaVinci Developeru koji se koristi kao baza za većinu izrađenih modela

Kako se veliki broj zahtjeva u SDG-u odnosi na međukomponentu, napravljeni model je korišten kao bazni model za veliku većinu izrađenih modela, jer se modifikacijom napravljenog modela, uz manje izmjene može napraviti model za veliku većinu implementiranih testova.

Primjer modela koji koristi navedeni bazni model kao osnovu (uz veće modifikacije) je model vidljiv na slici 3.7. Iako model na slici provjerava prepoznavanje svega jedne greške, neke zahtjeve nije moguće provjeriti bez većeg broja komponenata.



Slika 3.7 Prikaz primjera kompleksnog modela koji koristi model prikazan na slici 3.6 kao bazni model

Nadalje, kako je navedeno nešto ranije u poglavlju, u svrhu provjere konzistentnosti SDV-a, mnoge su komponente prenošene iz jednog modela u drugi, što znači da nekad sadrže i višak komponenti. U slučaju ovog specifičnog modela, višak je modul za trajnu memoriju. Kako je primjećeno da SDV relativno teško prepoznaje koja bi komponenta trebala biti navedeni modul, modul je ostavljen u mnogim modelima kako bi se vidjelo stvara li dodavanje dodatnih komponenata nove probleme.

Nisu svi modeli zahtjevali ubacivanje novih komponenata i brisanje starih. Popriličan broj modela rađen je samo modifikacijom imena varijabli, procesa i sl. Jedan dobar primjer modela rađenog za provjeru imena vezanih za tip podatka je model rađen za provjeru prepoznavanja greške E178. Navedena se greška javlja kada elementi vezani uz trajnu memoriju, velike podatke i skupove podataka ne sadrže pod-elemente: *DeStatus*, *DeVersion* i *DeData* u navedenom redoslijedu. Model je napravljen modifikacijom ispravljenog modela koji je korišten u svrhu ispitivanja modula za trajnu memoriju. Promijenjena su imena odgovarajućih elemenata, nakon čega je model verificiran. SDV je prepoznao grešku. Međutim vraćanjem na modele u kojima je u relevantnim elementima slučajno definiran pogrešan broj pod-elemenata, primjećeno je da u toj

situaciji SDV nije prepoznao grešku. Štoviše, pri testiranju tih modela javila se neobrađena iznimka. Navedena iznimka se nešto detaljnije opisuje u četvrtom poglavlju.

Nisu apsolutni svi modeli dobiveni isključivo korištenjem DaVinci Developera. Kako DaVinci Developer ne dopušta izvođenje nekih akcija, kao što je stvaranje više istoimenih varijabli, u tim slučajevima su testni modeli rađeni direktnom modifikacijom ARXML datoteke već postojećeg modela. Problem kod navedenog načina izrade modela je to što je teško odrediti ne prepoznaje li SDV grešku koja postoji ili jednostavno ARXML datoteka nije dobro modificirana. Tako je npr., u slučaju modela koji sadrži grešku E048, koja se javlja kada element nije definiran i u ulaznom i u izlaznom priključku, uspješno modificirana ARXML datoteka. Jedna definicija elementa je obrisana, nakon čega je model verificiran. U ovom slučaju je SDV primjetio grešku i zapisao je u izvješću. Međutim pri izradi modela koji je trebao sadržavati tip podatka i konstantu istog imena (greška E089), SDV nije uspješno prepoznao grešku.

U nastavku je priložena tablica sa svim izrađenim modelima (tablica 3.1) te je u njoj dano više informacija o svakom pojedinom modelu.

Tablica 3.1 Prikaz svih kreiranih testnih modela s kratkim opisom svakog modela

Redni broj modela	Ime modela	Broj različitih grešaka prisutnih u modelu	Broj grešaka koje model testira	Opis problema koji se nalaze u modelu
1	*E001,E002,E025 - missing connection*	3	3	Uklonjene veze između komponenata
2	*E014, E016 - persistency reversed*	4	2	Obrtanje definiranog smjera veze za priključke trajne memorije
3	*E028,E032 - too many active connections and no MW*	3	2	Dodijeljeno previše veza priključku i nedostatak međukomponente
4	*E029 - Init value not specified*	2	1	Nedodjeljene početne vrijednosti na ulaznom priključku međukomponente
5	*E044 - element not queued at both ports*	1	1	Elementi ne ulaze u red čekanja i na ulaznoj i na izlaznoj komponenti
6	*E045, E046 - invalid queue lenght*	2	2	Netočan broj elemenata u redu čekanja
7	*E047,E152 - incorrectly defined delegation ports*	4	2	Netočno definirani delegacijski priključci
8	*E048 - element not defined in both ports*	1	1	Element nije definiran u oba priključka
9	*E052 - implicit access on non persistency SWC*	2	1	Implicitan pristup priključka trajne memorija na komponenti koja njom ne rukuje

10	*E053, E169 - queued persistency*	3	2	Stavljanje podatkovnog elementa priključka za trajnu memoriju u red čekanja s implicitnim pristupom
11	*E055, E059, E066, E080, E097, E2463381, E268 - persistency 2*	8	7	Dobro spajanje komponenata trajne memorije bez dobro podešenih imena priključaka i procesa
12	*E056 - multiple runnables accessing the same queued element*	2	1	Više periodičnih procesa želi pristupiti jednom elementu u redu čekanja
13	*E057 - multiple runnables accessing the same non-queued element*	2	1	Više periodičnih procesa želi pristupiti jednom elementu koji nije u redu čekanja
14	*E060, E2463439, W2460070 - persistency - wrong naming*	5	3	Imena priključaka trajne memorije ne prate konvencije
15	*E063, E064 -wrong name on MW - persistency*	5	2	Pogrešno ime priključka trajne memorije na međukomponenti
16	*E066, E80, E171 - element does not end with _normal or _critical, wrong persistency port name*	7	3	Pogrešno ime elementa, nepridržavanje definiranog načina imenovanja priključka trajne memorije
17	*E067 - no connection to persistency module*	3	1	Nedostatak priključka za primanje podataka na modulu trajne memorije
18	*E081, E230, E237, E2962508 - fresh ASIL MW*	4	4	Prazna ASIL međukomponenta
19	*E082, E197 - wrong period on MW*	2	2	Pogrešan period pokretanja procesa na međukomponenti
20	*E089 - datatype shall not have the same name as constant and vice versa*	1	1	Jednako ime tipa podataka i konstante
21	*E097, E047 - no runnable listed*	3	2	Podatak nije dodijeljen periodičnom procesu
22	*E098 - Descripition of data type must not contain string*	1	1	Opis podatkovnog tipa sadrži nedozvoljene znakove
23	*E099, E100 - CS lacking connection*	4	2	Obrisana veza između klijent-poslužitelj priključaka
24	*E121, E122, E2473230 - data set can not be on persistency*	4	3	Pogrešna lokacija priključaka za slanje setova podataka i netočan pristup podatcima
25	*E123, E254 - MW cant receive data sets*	4	2	Međukomponenta prima setove podataka
26	*E155 - bigdata read*	2	1	Priključak za čitanje velikih podataka nije na modulu trajne memorije

27	*E156, E158 - bigdata on MW*	10	2	Priključci za čitanje i pisanje velikih podataka se nalaze na međukomponenti
28	*E159 - bigdata write*	2	1	Priključak za pisanje velikih podataka nije na modulu trajne memorije
29	*E160 - bigdata on MW*	10	1	Priključak za čitanje i slanje velikih podataka se nalazi na međukomponenti
30	*E162 - bigdata must use implicit acess*	2	1	Elementima velikih podataka se ne pristupa implicitno
31	*E164, E169 - queued data set*	4	2	Veza priključka za setove podataka je implicitna i ulazi u red čekanja
32	*E170 - illegal asil lvl in data type*	3	1	Tipu podataka je dodjeljena nedopuštena ASIL razina
33	*E174 - data set port does not end in dataSet*	3	1	Ime priključka za setove podataka ne završava na *dataSet*
34	*E178 - not expected names in data type*	3	1	Tip podataka ne sadrži pravilna imena varijabli
35	*E210 - unspecified enum*	2	1	Enumerator sadrži neočekivano ime
36	*E218 - Id out of range*	2	1	Identifikacijski broj komponente nije u definiranom rasponu
37	*E219 - illegal asil lvl in SWC*	3	1	Komponenta sadrži nedopuštenu ASIL razinu u opisu
38	*E241, W2460070 - elements must use buffered access*	2	1	Elementi u priključku trajne memorije ne koriste implicitan pristup
39	*E249 , E250 - Wrong component type in enum*	3	2	Pogrešna definicija komponente u enumeratoru
40	*E256 - bigdata ports must have the same name*	2	1	Spojeni priključci za velike podatke nemaju isto ime
41	*E2426155 - no system tag*	1	1	Uklonjena oznaka sustava iz ARXML-a
42	*E2460057 - expected S/R*	2	1	Pogrešna metoda komunikacije između dviju komponenata
43	*E2463339 - calibration swc has internal behaviour*	4	1	Kalibracijska komponenta ima definirano ponašanje
44	*Sve_zadovoljeno*	0	0	Model nema grešku

Zbog lakšeg snalaženja pri radu s modelima, ime svakog modela sastoji se od oznake grešaka koje sadrži i isječka zahtjeva opisanog u SDG-u. U svrhu skraćivanja svakog imena što je više moguće, u imenu samo pišu greške koje se prvi puta pojavljuju i koje on ciljano sadrži.

3.3. Alati i tehnologije korištene za izradu grafičkog sučelja za SDV

3.3.1. C#

C# (čitati *C-sharp*) objektno je orijentirani programski jezik koji kao osnovu koristi programski jezik C. Razvio ga je Microsoft u .NET razvojnom okviru još 2002. godine, s tim da je najnovije verzija, C# 9, izdana 2020. godine [9]. Zbog jednostavnosti i široke primjenjivosti, jedan je od najpopularnijih programskih jezika u svijetu. C# sadrži mnoge značajke korisne za izradu aplikacija. Omogućava automatsko brisanje nekorištene dinamički alocirane memorije, što smanjuje rizik od curenja memorije. Podržava asinkrone operacije koje omogućavaju izradu složenijih sustava. Definirana je i sintaksa za upite (engl. *Query syntax*), koja uvelike olakšava pretraživanje podataka [10]. To je ujedno i jedan od razloga zašto je C# odabran za izradu grafičkog sučelja za rješenje izrađeno u sklopu ovog diplomskog rada. Jezik čini pretragu ARXML datoteka, kojim se ovaj rad bavi, iznimno jednostavnim pomoću definirane sintakse za upite koja se zove LINQ (engl. *Language-Integrated Query*). Također, podržava i Windows Forms razvojni okvir za izradu grafičkih sučelja aplikacija koji je korišten za izradu istog.

3.3.2. LINQ

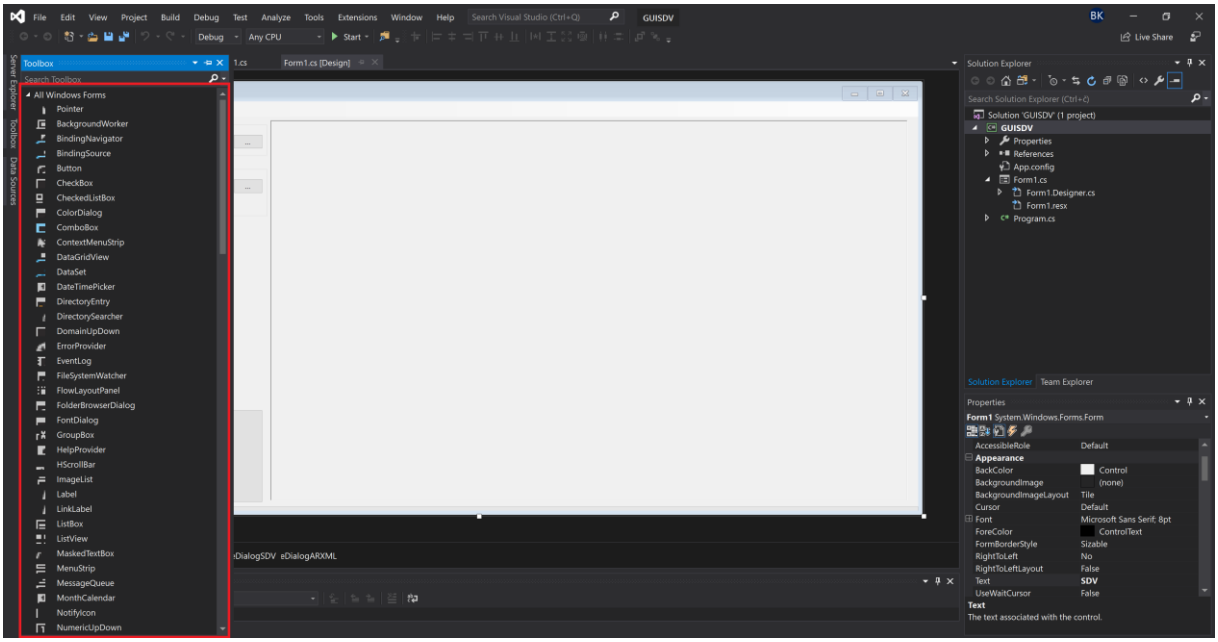
LINQ je skup tehnologija stvoren kako bi se mogućnost upita integrirala direktno u C# [11]. LINQ omogućava pretragu različitih tipova datoteka, kao što su SQL baze podataka, XML dokumenti, web servisi i sl. Inače bi se sve navedeno pretraživalo različitim jezicima, no LINQ pruža jedinstvenu sintaksu za upit, neovisno o tome što pretražuje [11].

Njegova mogućnost čitanja različitih izvora podataka preslikava se i na njegovu mogućnost spremanja istih. LINQ je sposoban čitati jedan format (npr. SQL baze podataka) i pisati direktno u drugi (npr. XML). LINQ upiti se mogu pisati na dva različita načina: sintaksom za upite ili pozivom metode. Način pisanja preko metoda se rjeđe koristi, međutim neke operacije imaju isključivo taj oblik (npr. operacija za traženje najveće vrijednosti *Max*) [11].

3.3.3. Windows Forms

Windows Forms je razvojni okvir za grafička sučelja u Windows operacijskim sustavima. Kao razvojna platforma pruža korisniku većinu potrebnih značajki razvoja Windows aplikacije [12]. Windows Forms sadrži funkcije za upravljanje grafikom, korisničkim unosom, obradu podataka i sl. Zajedno s razvojnim okruženjem kao što je Microsoft Visual Studio, Windows Forms omogućuje izradu grafičkog sučelja uz pomoć različitih predefiniranih kontrola. Kontrola je element grafičkog sučelja koji tipično prikazuje podatke ili vrši neku interakciju s korisnikom

(npr. registriraju klik miša) [12]. U Visual Studiju se te kontrole nalaze u Windows Forms dizajneru, iz kojeg se željene kontrole mogu povući direktno na formu (slika 3.7).

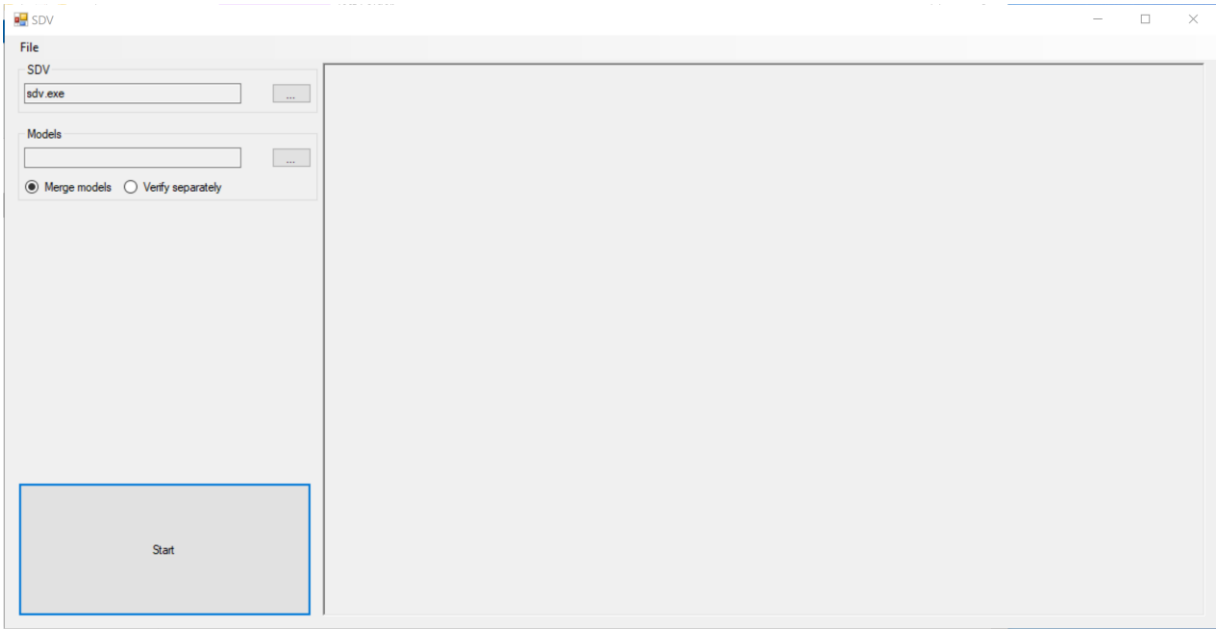


Slika 3.8 Korisničko sučelje Microsoft Visual Studija s naznačenim predefiniranim kontrolama Windows formi

Forma je površina koju korisnik vidi i na kojoj se nalaze sve dodane kontrole [12]. Predefinirane kontrole sadrže većinu toga potrebnog za izradu aplikacije, kao što su tipke, kućice za prikaz teksta, labele i sl. Također su podržane i kontrole definirane od strane korisnika, za slučajeve kada predefinirane kontrole nisu dovoljne [12]. Sve navedeno čini Windows Forms razvojni okvir vrlo dobrim odabirom za brzu i efikasnu izradu aplikacija i alata.

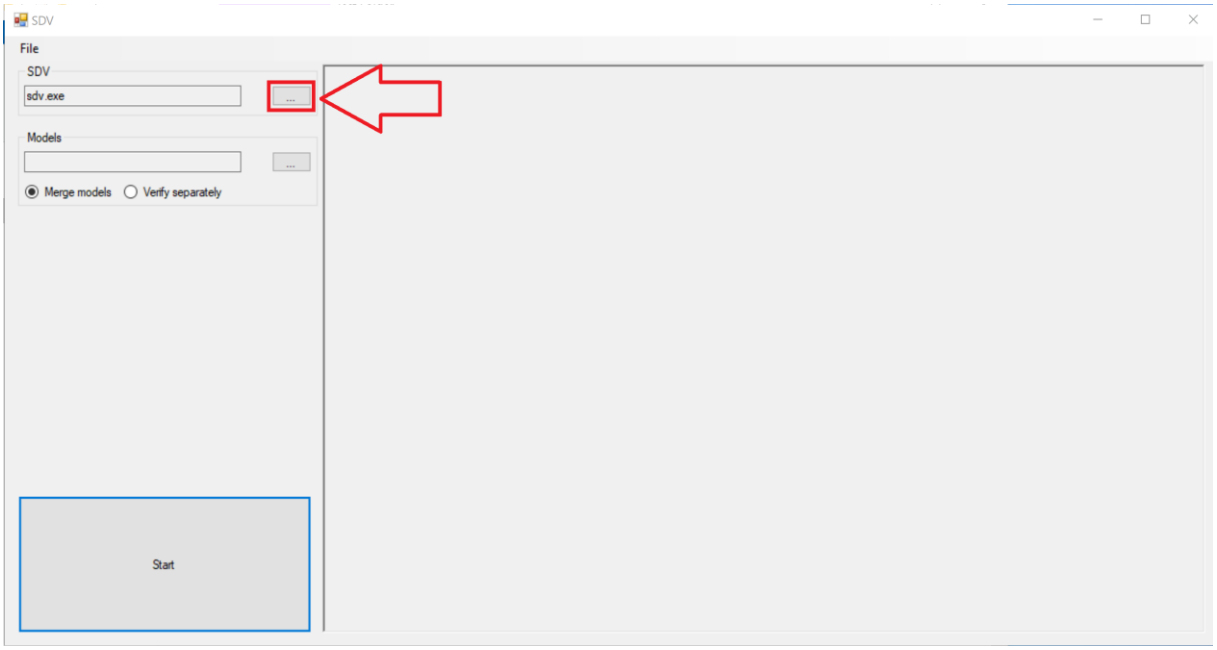
3.4. Opis izrađenog grafičkog sučelja za SDV

Kao što je već rečeno na samom početku 3. poglavlja, svrha izrade ovog grafičkog sučelja je bila da se olakša korištenje SDV alata. Konačni izgled grafičkog sučelja je vidljiv na slici 3.9. Grafičko sučelje je rađeno da bude što jednostavnije, kako bi bilo lakše za korištenje. Sučelje se sastoji od svega nekoliko tipki koje služe za korisnički unos. Kao što je navedeno već u podpoglavlju 2.2, korištenje SDV-a zahtjeva od korisnika da unese putanju do samog alata i putanju do modela kojeg želi verificirati kao argument u upravljački prozor. Prema tome, prvi korak koji korisnik mora napraviti je predati alatu putanju gdje se nalazi SDV. U slučaju prikazanog grafičkog sučelja, to se može postići pritiskom na prvi gumb (slika 3.10). Alternativno, korisnik može, ako si želi skratiti cijeli proces, smjestiti SDV i grafičko sučelje na istu lokaciju.



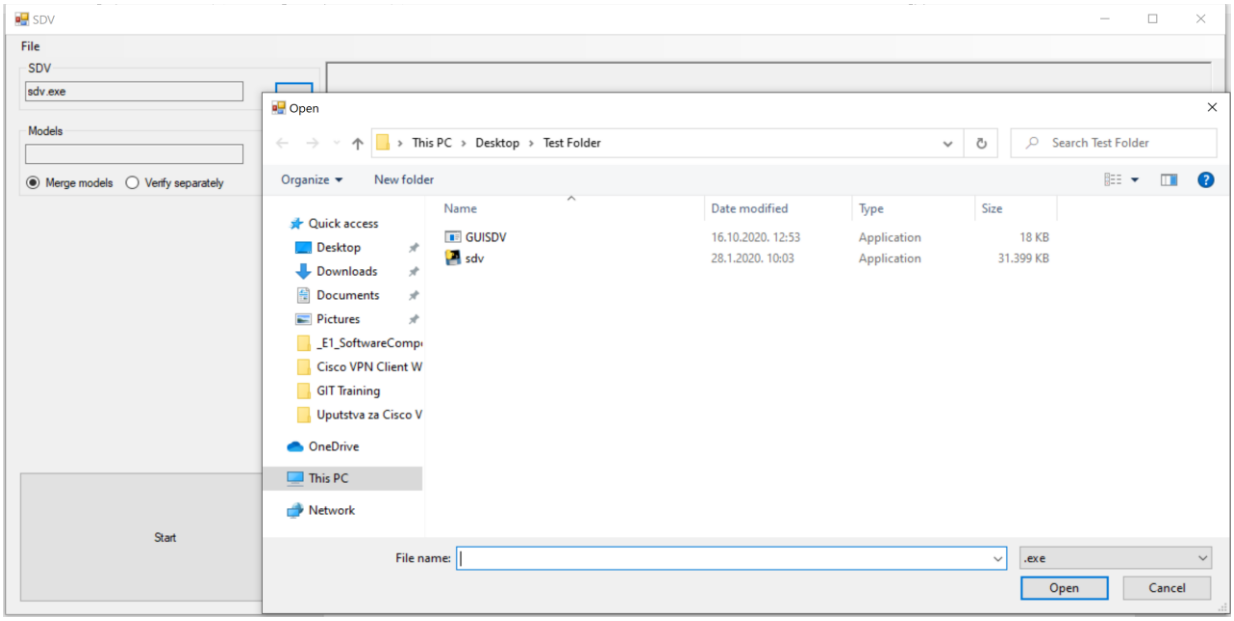
Slika 3.9 Grafičko sučelje za SDV izrađeno u sklopu ovog diplomskog rada

Sučelje provjerava nalazi li se datoteka pod imenom sdv.exe u istoj mapi i automatski ju učitava u tom slučaju. Problem kod navedenog načina korištenja je u tome što neće funkcionirati u slučaju ako se način imenovanja datoteke u budućnosti promjeni, međutim, u tom slučaju se novo ime vrlo lako nadoda u izvorni kod.



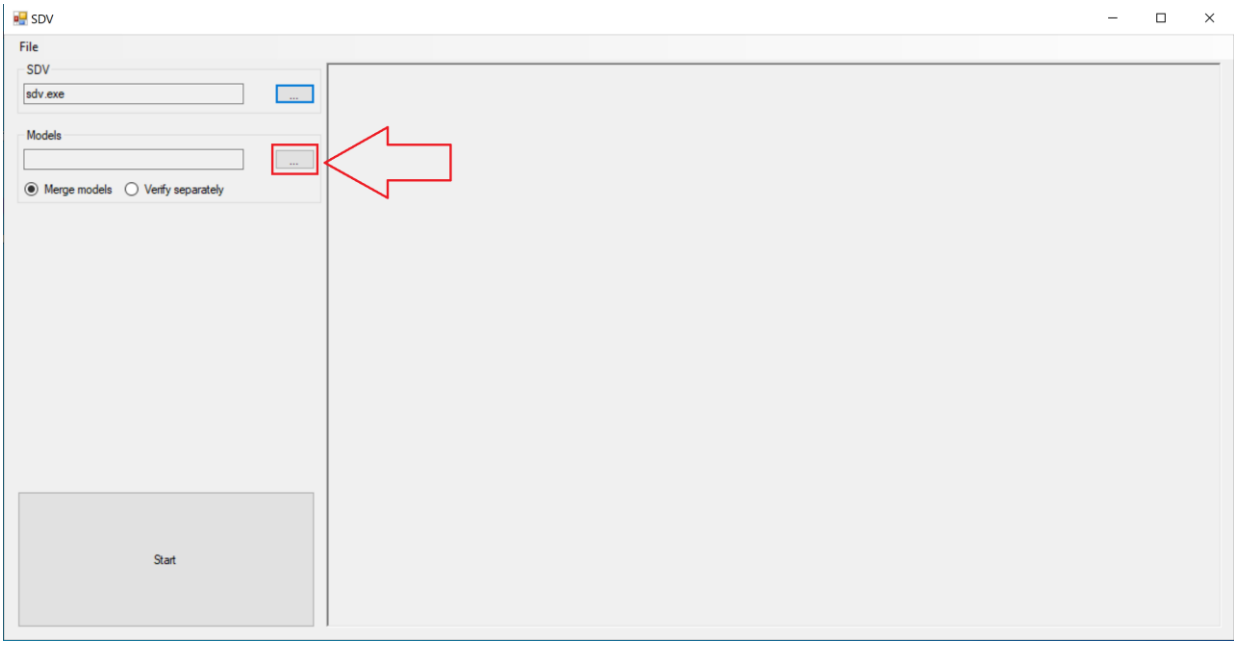
Slika 3.10 Lokacija gumba za odabir putanje do SDV-a u kreiranom grafičkom sučelju

U slučaju da korisnik to iz nekog razloga ne želi raditi na taj način, onda će se pritiskom na spomenuti gumb otvoriti prozor u kojem korisnik treba odabrati odgovarajuću datoteku (slika 3.10).



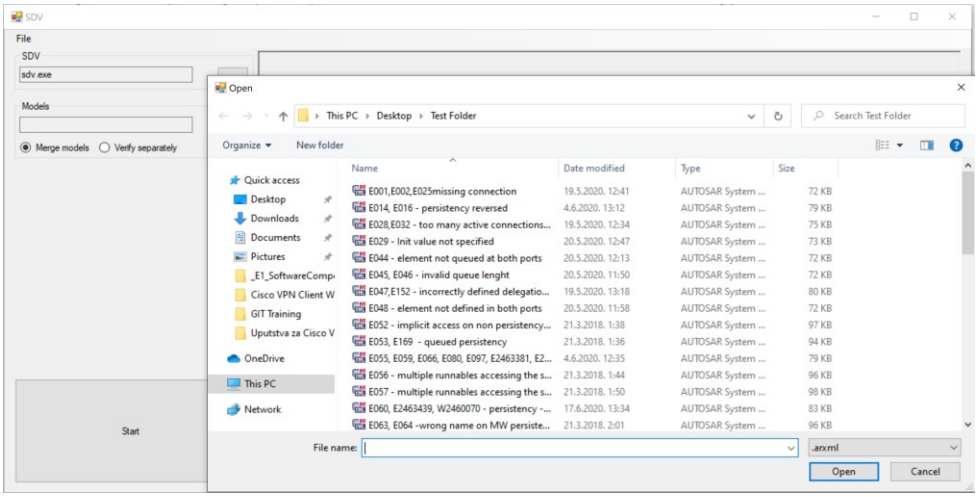
Slika 3.11 Prozor za odabir SDV-a koji se prikazuje nakon pritiska na odgovarajući gumb

Nakon odabira verzije SDV-a kojom se korisnik želi služiti, mora odabrati modele koje želi ispitati, kao i inače. To može postići pritiskom na drugi gumb (slika 3.12).



Slika 3.12 Lokacija gumba za odabir putanje do modela u kreiranom grafičkom sučelju

Kao i pri odabiru SDV-a, pritiskom gumba otvara se prozor u kojem treba odabrati model koji korisnik želi ispitati (slika 3.13).



Slika 3.13 Prozor za odabir modela koji se pojavljuje nakon pritiska na odgovarajući gumb

Prozori na slikama 3.10 i 3.12 su identični u svemu, osim u vrsti datoteka koje filtriraju. Može se vidjeti da prozor na slici 3.12 filtrira van datoteke koje ne sadrže nastavak *.arxml*, u kojem se spremaju modeli (kao što je već nekoliko puta navedeno).

Jedna dodatna razlika je i činjenica da prozor za odabir modela podržava odabir više datoteka odjednom. Ta opcija je omogućena jer se SDV-u, kao što je navedeno u podpoglavlju 2.2, može predati više od jednog modela kao argument. U tom slučaju ih on spaja u jedan model koji onda provjerava. Svi modeli odabrani preko prozora za odabir spremaju se u jednu listu, iz koje se onda čitaju za prikaz imena odabranih modela (slika 3.13).

```
private void OpenFileDialogARXML_FileOk(object sender, CancelEventArgs e)
{

    textBoxARXML.Text = openFileDialogARXML.FileNames[0];

}
```

Slika 3.14 Čitanje liste u kojoj su spremljeni modeli, kako bi se prikazalo ime modela

Ista lista se koristi i za predaju argumenata SDV-u. Preostaje još jedan korak prije pokretanja procesa provjere odabranih modela, a to je odabir načina rada. Za razliku od klasičnog rada SDV-a, gdje je jedina opcija da se odabrani modeli spajaju u jedan i onda provjeravaju, u sklopu napravljenog sučelja je dodana druga mogućnost. U sklopu grafičkog sučelja dodana je i

mogućnost da se više različitih modela provjere zasebno, označavanjem odgovarajućeg gumba (slika 3.15).

Slika 3.15 Lokacija gumbova za odabir načina provjere odabranih modela

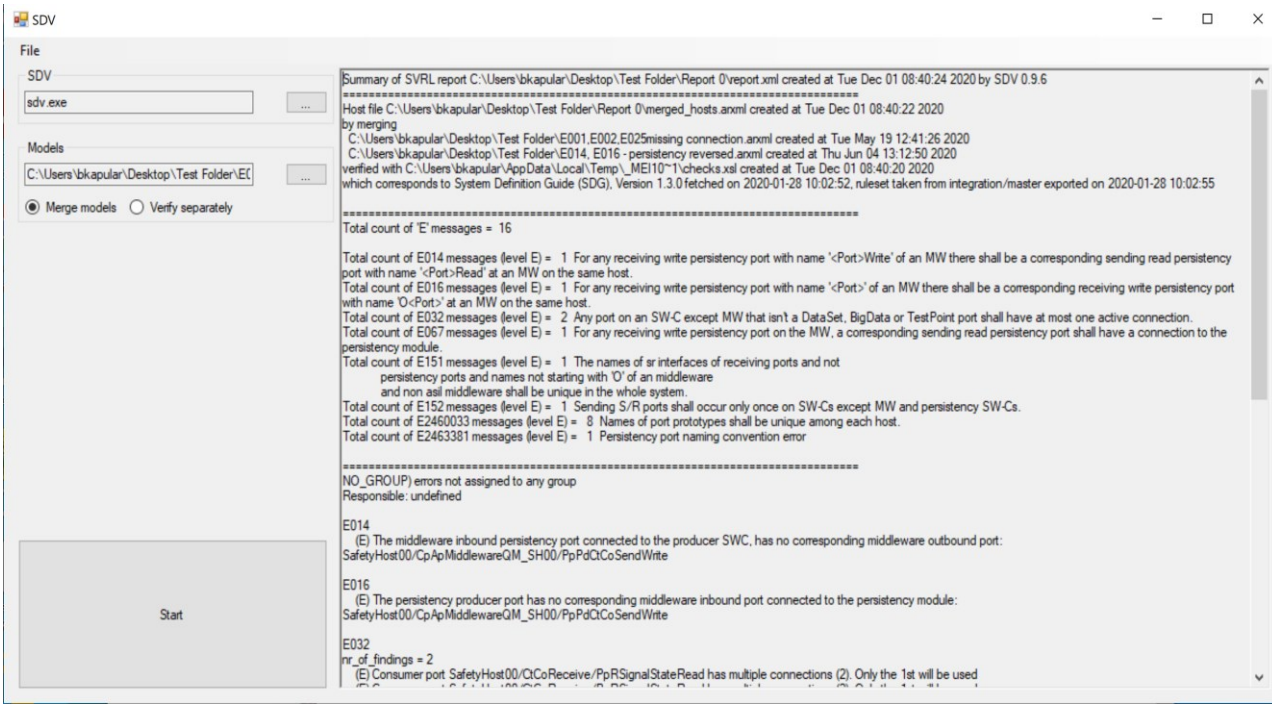
Drugi način rada je dodan kako bi se ubrzala provjera više različitih modela. Bez ovog načina, prije se moralo zasebno pokretati SDV za svaki model i nakon završetka jednog ponovno pokretati sljedeći, itd. Način na koji sučelje učitava modele ne mijenja se ovisno o odabranom načinu rada, što znači da ga korisnik ne mora odabrati prije samih modela.

Posljednji korak za korisnika sučelja je kliknuti na veliku tipku na kojoj piše *Start*, vidljivu još na ranijim slikama u donjem lijevom uglu sučelja. Pritiskom na *Start* pokreće se proces provjere odabranih modela na jedan od dva već navedena načina, ovisno o odabiru korisnika.

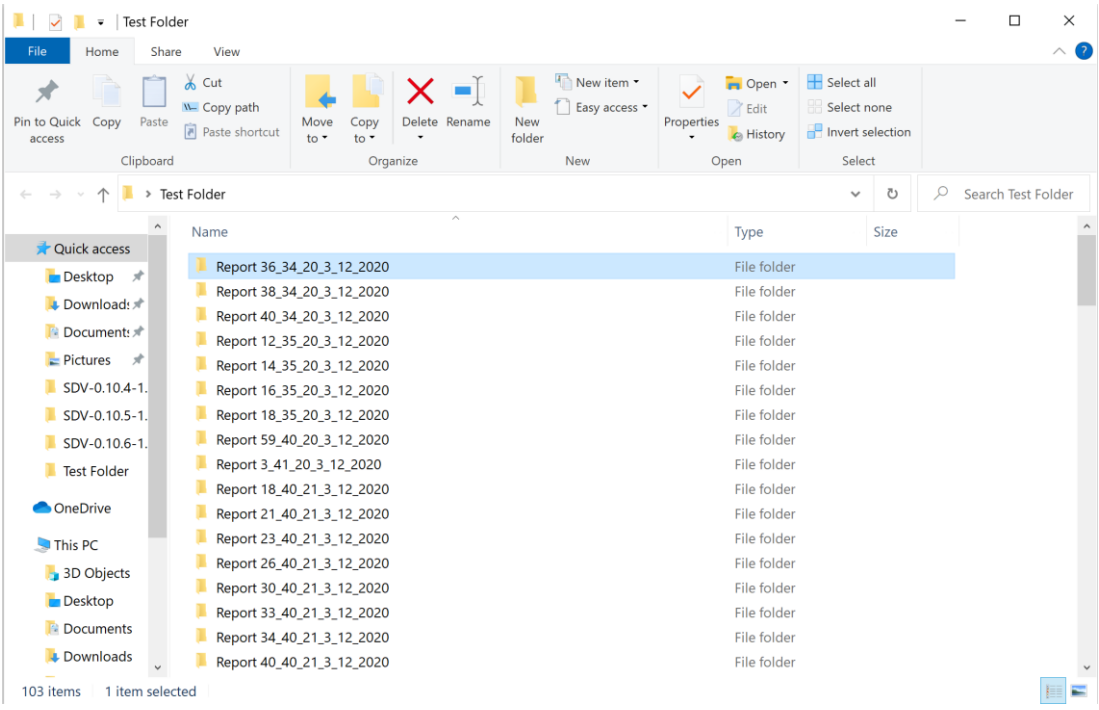
3.4.1. *Merge models* opcija

U slučaju da je pri pokretanju bilo odabrano *Merge models*, alat se pokreće na klasičan način. Svi se odabrani modeli spajaju u jedan i alat ih provjerava kao da su jedan model. Izvještaj koji se generira nakon što proces završi učitava se direktno u grafičko sučelje, što znači da ga korisnik ne mora samostalno otvarati kao inače (slika 3.16). Stvorene izvještaje i spojene modele korisnik može pronaći u istoj mapi u kojoj se nalazi i grafičko sučelje. Točnije, nalaze se u zasebnim mapama koje se stvaraju pri obradi svakog modela. Kako bi se izbjeglo prepisivanje starih izvještaja, ime svake mape ovisi o vremenu u kojem je stvorena (slika 3.17). Ime svake stvorene mape formira se na sljedeći način: *Report sekunda_minuta_sat_dan_mjesec_godina*. Na

slici 3.17 se tako može vidjeti da je izvještaj stvoren 3.12.2020. u 20 sati, 34 minute i 36 sekundi, spremljen u mapu imena *Report 36_34_20_3_12_2020*.

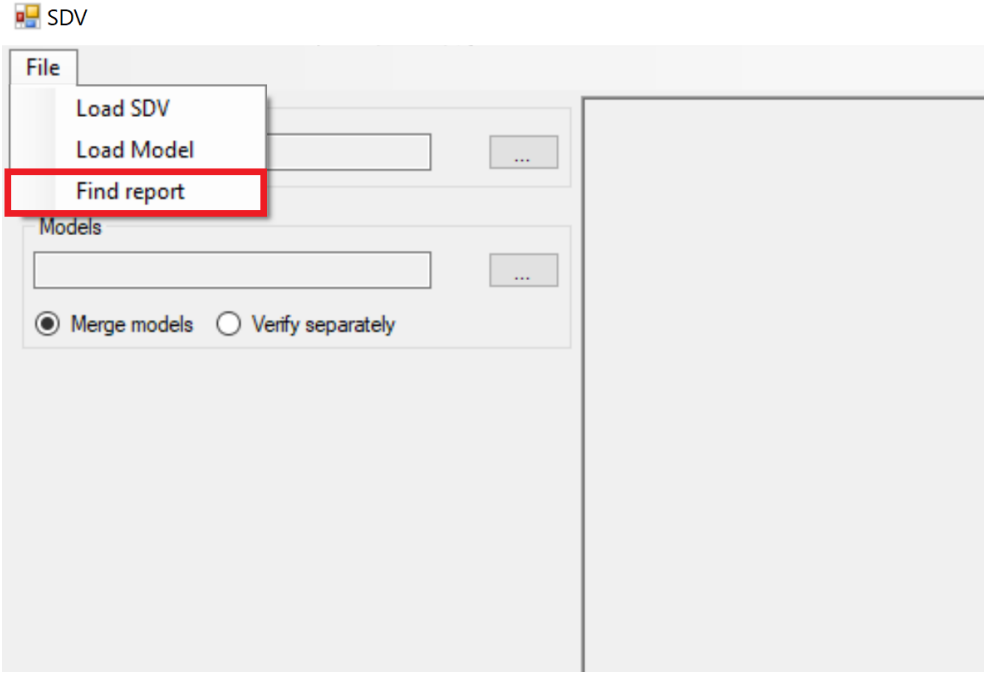


Slika 3.16 Prikaz grafičkog sučelja s učitanim izvještajem koji je generiran pri ispitivanju modela



Slika 3.17 Prikaz mapa koje se automatski stvaraju za vrijeme rada SDV-a kako bi se izbjeglo prepisivanje izvještaja

U svrhu lakšeg pronalaska trenutnog izvještaja, dodana je i opcija pronalaska istog. Pritiskom na gumb u gornjem lijevom uglu sučelja, na kojem piše *File*, otvara se izbornik s gumbom na kojem piše *Find report* (slika 3.18). Pritiskom na navedeni gumb, otvara se prozor na memorijskoj lokaciji izvještaja.

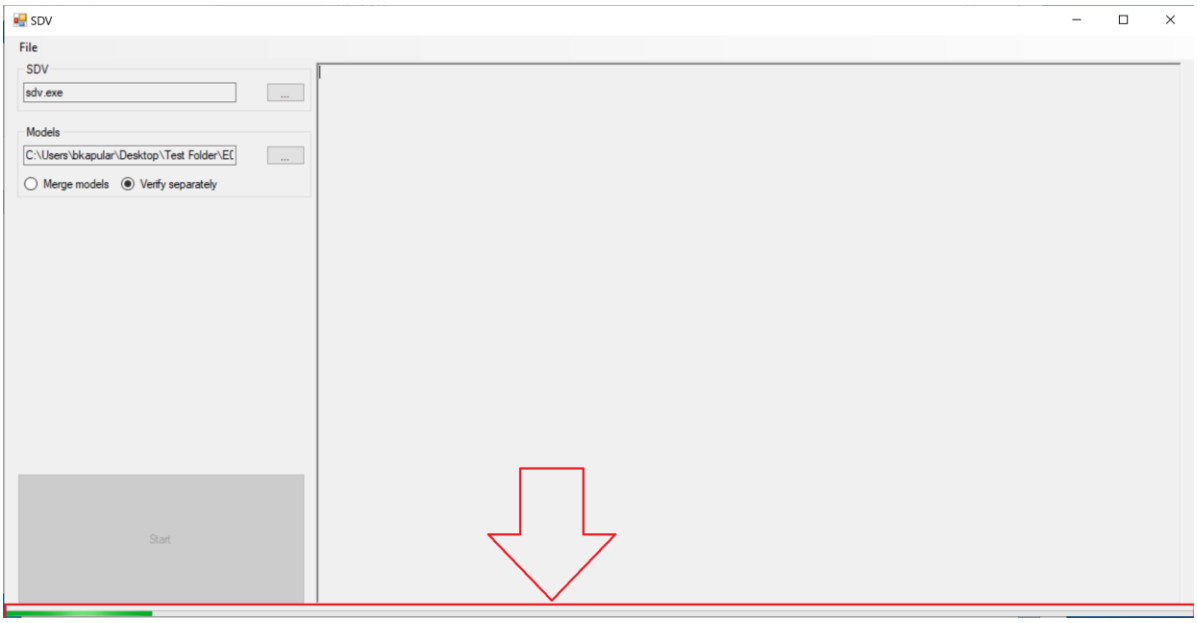


Slika 3.18 Prikaz grafičkog sučelja s istaknutim gumbom za pronalazak posljednjeg generiranog izvješća

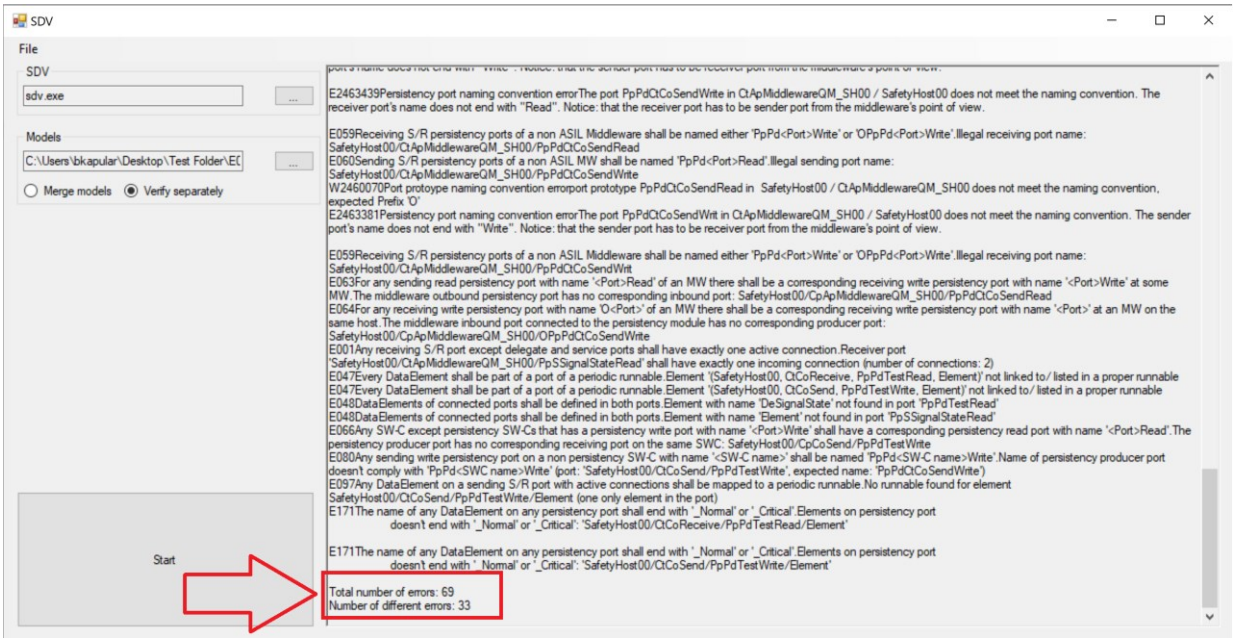
3.4.2. *Verify separately* opcija

Kao što je već navedeno, u sklopu grafičkog sučelja dodana je opcija za odvojenu provjeru svih odabranih modela. Za razliku od prve opcije, gdje se svi modeli spajaju u jedan i time tehnički svi odjednom provjeravaju, u ovom načinu rada modeli se provjeravaju jedan po jedan. Kako sekvencijalna obrada velikog broja modela može značajno potrajati, za vrijeme rada pojavljuje se traka koja pokazuje trenutni napredak na dnu prikaza (slika 3.19). Ista traka se prikazuje i pri korištenju *Merge models* opcije, no zbog činjenice da taj način rada puno brže radi, traka je od puno manjeg značaja.

S obzirom da svaka obrada stvara zaseban izvještaj, kako bi se izbjegla potreba za otvaranjem svakog od njih radi pojedinačne analize, u sklopu rada sučelja stvara se jedno zajedničko izvješće (slika 3.20). U sklopu stvaranja zajedničkog izvješća, ubačene su još neke informacije korisne za analizu rezultata. Dodana je i informacija o ukupnom broju grešaka koje su se pojavile, ali i informacija o broju jedinstvenih grešaka, koje se vide na dnu izvješća na slici 3.20. Jedinstvenom greškom se smatra jedan tip greške.



Slika 3.19 Prikaz grafičkog sučelja s naznačenom trakom napretka za vrijeme pojedinačnog ispitivanja modela



Slika 3.20 SDV s otvorenim zajedničkim izvješćem i ispisanim brojem grešaka

U slučajevima kada se jedan tip greške više puta ponavlja, neovisno o tome jesu li u potpunosti identične (npr. ako se događaju na različitim komponentama), višak grešaka se uklanja. Razlog za moguće ponavljanje grešaka opisan je u podpoglavlju 3.2.

Prema tome, podatak o broju jedinstvenih grešaka je potencijalno informativniji za korisnika, pogotovo u slučaju da korisnik izrađuje testne skupove, kao što je situacija i u ovom diplomskom radu.

Način na koji grafičko sučelje dolazi do navedenih podataka je pomoću XML izvještaja koje SDV stvara. Već spomenuti u podpoglavlju 2.2, XML izvještaji, za razliku od običnih tekstualnih izvještaja, jasno su strukturirani prema standardnim pravilima XML jezika. To znači da se iz njih mogu izdvojiti željene informacije puno jednostavnije nego iz običnih tekstualnih izvještaja. Za izdvajanje grešaka iz XML datoteke korišteni su LINQ upiti (slika 3.21).

```
IEnumerable<string> errorIDs = from item in xmlFile.Descendants()
                               where item.Name.LocalName == "error-id"
                               select item.Value;


IEnumerable<XElement> errors = from item in xmlFile.Descendants()
                               where item.Name.LocalName == "failed-assert"
                               select item;
```

Slika 3.21 LINQ upiti koje grafičko sučelje koristi za izdvajajnje grešaka i njihovih identifikacijskih brojeva

Učitane greške se spremaju u listu *IEnumerable* tipa podataka. Pomoću te liste se onda zapisuju u zajednički izvještaj (slika 3.22). *IEnumerable* sam po sebi sadrži funkciju *Count()* koja prebrojava broj elemenata koje sadrži, što nam daje broj grešaka.

```
File.AppendAllText("General Report.txt", "Total number of errors: " + indivErrors.Count().ToString() + "\n");

File.AppendAllText("General Report.txt", "Number of different errors: " + indivErrors.Distinct().Count().ToString() + "\n");
```

Slika 3.22 Prikaz funkcije prema kojoj se greške prebrojane funkcijama *Count()* i *Distinct()* zapisuju u izvještaj

Iz iste liste se prebrojava i ukupni broj grešaka koji je učitan. Do broja jedinstvenih grešaka se dolazi jednako jednostavno. Koristeći ugrađenu funkciju *Distinct()* zajedno sa funkcijom *Count()* su uklonjeni ponavljajući elementi liste. Cijeli navedeni proces je moguć jer svaki tip greške ima svoj zasebni identifikacijski broj greške (engl. *error ID*). Zbog toga je moguće tretirati greške koje su možda izazvane na različitim komponentama pa sadrže nešto drukčiji opis, kao više instanci jedne greške umjesto više različitih.

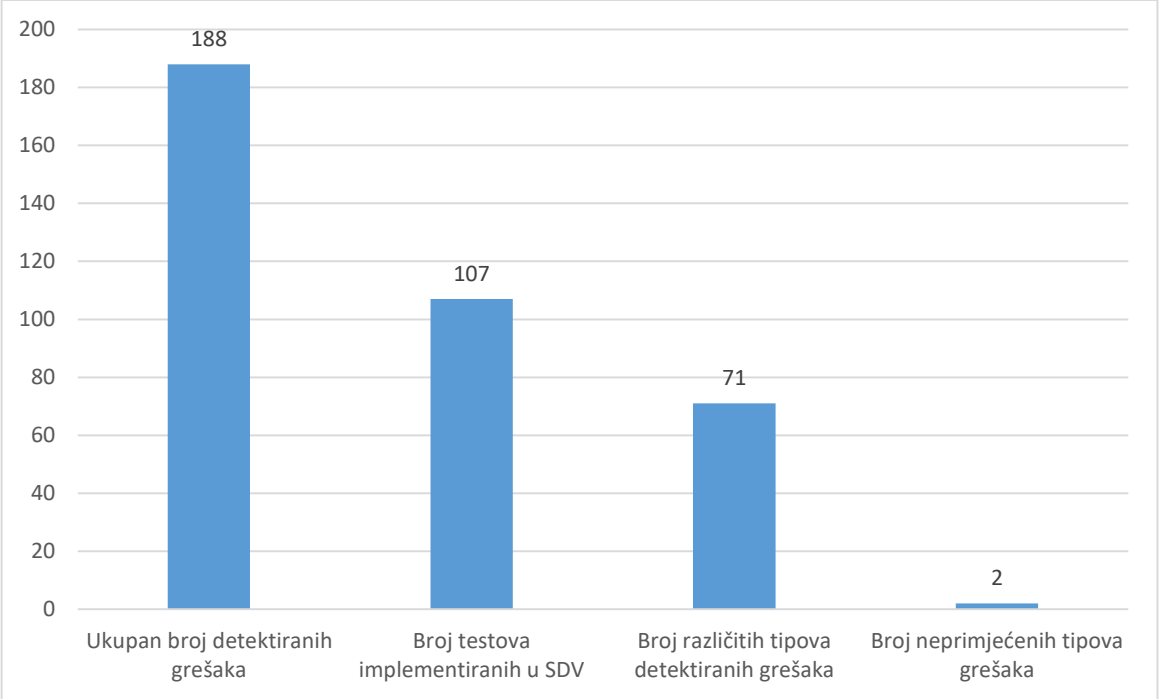
4. VERIFIKACIJA SDV-A UZ POMOĆ IZRAĐENOG TESTNOG SKUPA ALATA

U ovom su poglavlju opisani rezultati testiranja SDV-a na skupu modela izrađenom za njegovo testiranje. Kao što je već opisano u podpoglavlju 3.2, modeli su rađeni za testiranje verzije **0.9.6** SDV-a. Kako je nekoliko novijih verzija SDV-a napravljeno za vrijeme trajanja izrade ovog diplomskog rada, modeli su pokrenuti i na tim verzijama kako bi se utvrdila njihova primjenjivost na buduće verzije alata.

Proces validacije alata započeo je pri samoj izradi testnog skupa modela. Za vrijeme izrade modela, vodilo se računa o svim ciljanim i slučajnim greškama koje su se pojavljivale, kao i o greškama koje su se možda trebali pojaviti, ali nisu.

4.1. Provjera svih modela testnog skupa pomoću SDV-a

Kako bi se utvrdila uspješnost prepoznavanja grešaka prisutnih u modelima, provedena je statistička analiza rezultata. Za praćenje ukupnog broja detektiranih grešaka i broja različitih grešaka, zbog jednostavnosti je korišteno grafičko sučelje izrađeno u sklopu ovog diplomskog rada. Na slici 4.1 mogu se vidjeti rezultati dobiveni provjerom 44 izrađena modela na verziji **0.9.6** SDV-a.



Slika 4.1 Prikaz rezultata pokretanja svih modela testnog skupa na verziji **0.9.6** SDV-a

Na slici 4.1 se mogu vidjeti četiri različita stupca. Prvi stupac se odnosi na ukupan broj grešaka koji je SDV detektirao u testnom skupu modela. Drugi stupac prikazuje trenutni broj testova implementiranih u SDV. To znači da se u verziji **0.9.6** provjerava 107 različitih zahtjeva opisanih u SDG-u, tj. da postoji 107 različitih grešaka koje se mogu pojaviti. Zadnja dva stupca prikazuju ukupan broj različitih tipova detektiranih grešaka i broj tipova grešaka koje nisu detektirane.

S obzirom da su modeli rađeni ciljano za ispitivanje određenog podskupa mogućih grešaka, broj različitih grešaka prisutnih u modelima je unaprijed poznat. U modelima su prisutna 73 različita tipa grešaka, od kojih se neke neizbježno ponavljaju, što znači da testni skup pokriva **68,22%** (73/107) mogućih grešaka. Zbog tog ponavljanja se ukupan broj grešaka značajno razlikuje od broja mogućih tipova greški. Iz rezultata se može vidjeti da nisu sva 73 tipa greške uspješno detektirana. Uspješno je detektiran 71 tip greške od moguća 73, što znači da je detektirano **97,26%** različitih tipova grešaka u testnom skupu modela.

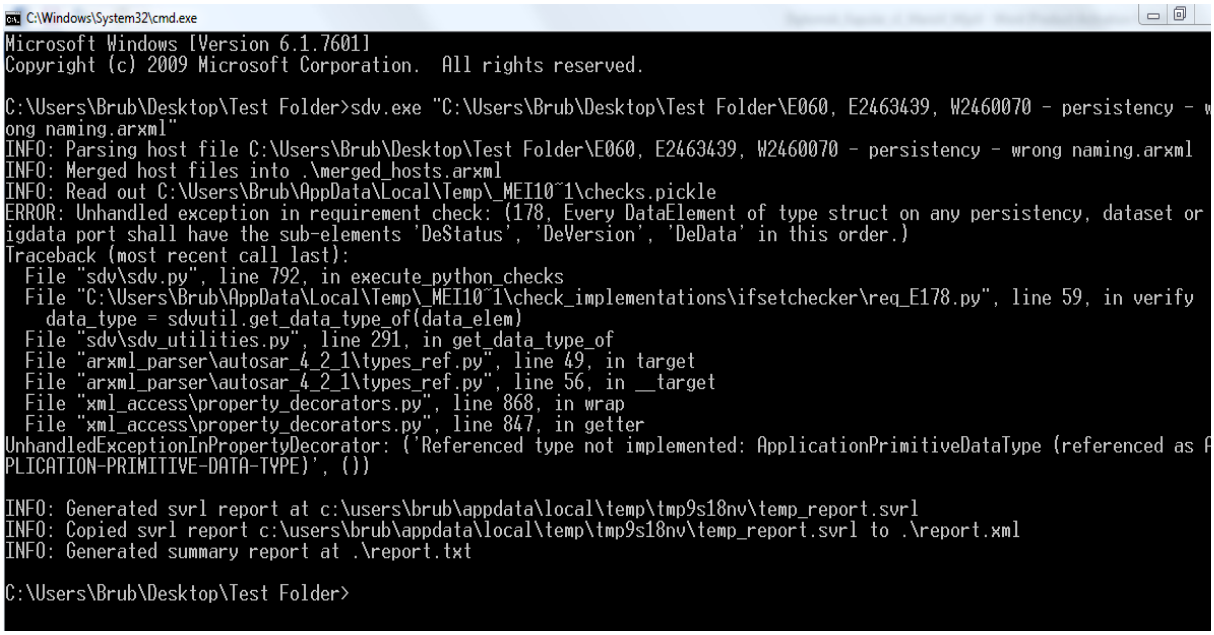
Ta brojka se treba uzeti s oprezom jer su te greške ujedno i jedne od rijetkih koje su zahtijevale direktnu manipulaciju ARXML datotekom kako bi se izazvale i moguće je da iste nisu dovoljno dobro izmijenjene da SDV primjeti grešku. Razlog za to je činjenica da DaVinci Developer ne dopušta da se različite varijable, konstante i sl. imenuju jednako zbog čega je došlo do potrebe za direktnom izmjenom ARXML-a. Modeli koji sadrže spomenute greške su modeli broj 20 i 22 u tablici 3.1.

Ponavljanjem ispitivanja nisu primijećene razlike u dobivenim rezultatima, što je prema očekivanjima s obzirom da SDV ne bi trebao modificirati modele koji mu se predaju. Jedna iznimka je ako mu se preda model koji se zove *merged_hosts*. To je inače ime pod kojim SDV sprema model koji je rezultat spajanja svih predanih modela. U slučaju da mu se preda samo *merged_hosts* model, SDV mu briše sav sadržaj, što ne bi trebao inače biti problem ako korisnik promjeni ime modela. Međutim, bilo bi potencijalno korisno ograničiti sposobnost SDV-a da se pokrene u tom slučaju, kako bi se izbjegao potencijalan gubitak podataka.

4.2. Analiza iznimki koje su se pojavile pri provjeri određenih modela

Uz prepoznate greške, pažnja je obraćena i na iznimke koje su se pojavljivale pri pokretanju alata (slika 4.2). Primjećeno je da se iznimke većinom pojavljuju u situacijama kada bi SDV trebao prepoznati neku grešku, ali iz nekog razloga u tome ne uspijeva. Budući da SDV pokreće svaki test zasebno, SDV će normalno odraditi ostatak provjere, čak i kada se pojavi iznimka. SDV u

većini slučajeva pravilno prepoznaje koje testove treba pokrenuti, ali iz nekog razloga se testovi ne izvedu pravilno.



Slika 4.2 Prikaz ispisa pri provjeri modela 14, u kojem se zbog neispravnog rada testa javlja iznimka za provjeru greške E178

4.2.1. Iznimka testa za grešku E178

U ovom specifičnom slučaju vidljivom na slici 4.1, primjećeno je da do iznimke dolazi jer test samo provjerava je li svaki podatak u skupu pravilno imenovan, a ne i nalazi li se dobar broj podataka u skupu. Modeli na kojem se navedena iznimka pojavljuje, u skupu koji test provjerava, sadrže samo jedan element, umjesto očekivana tri. No nije da sami test uvijek ne funkcionira. Kada su druga dva elementa dodana, test se počeo pravilno izvoditi. U slučaju da imena sva tri elementa nisu pravilno imenovana i poredana, SDV ispisuje grešku u izvještaj kako i treba.

4.2.2. Iznimka testa za grešku E332

Iznimka testa za grešku E332 pojavljivala se u nešto drukčijoj situaciji naspram prošle opisane iznimke. Iznimka za spomenutu grešku javlja se u modelima koji ne sadrže specifičnu komponentu. Javlja se u svim modelima koji ne sadrže komponentu *CtApComQM*. Greška E332 se javlja u slučaju nepravilne strukture priključaka navedene komponente. To nije jedina situacija da se provjerava struktura nekog priključka, međutim je jedina koja stvara problem. Kako bi se spriječilo pojavljivanje navedene iznimke, u većinu modela je dodana navedena komponenta.

4.2.3. Iznimka testa za grešku E203

Iznimka vezana uz grešku E203 je primjećena samo u jednoj situaciji. Navedena iznimka pojavila se samo u jednom modelu testnog skupa. Model broj 41 iz tablice 3.1 u kojem je došlo do spomenute iznimke je napravljen za provjeru greške E2426155. E203 se javlja u slučaju nepravilnog pristupa elementima testnog priključka komponente. S obzirom da se greška E2426155 javlja u slučaju da u modelu nedostaje oznaka sustava (engl. *System tag*) i ne postoji direktna povezanost između navedenih grešaka, teško je doći do zaključka zašto do nje dolazi.

Problem kod iznimke na slici 4.1 i drugih iznimaka je činjenica da iznimke nisu zapisane u izvještaju koji SDV generira. To otežava analizu pravilnog rada alata, jer ih je moguće propustiti pri analizi većeg broja modela. Također ih je teže pratiti jer zahtijevaju pojedinačnu analizu ispisa u upravljačkoj konzoli za dotične modele. Na spomenute iznimke je obraćena pažnja i pri usporedbi s novijim verzijama SDV-a, o čemu će biti riječi kasnije u ovom poglavlju.

Zbog lakše usporedbe s budućim verzijama SDV-a, sačuvani su svi izvještaji generirani za konačne verzije modela. Uspoređeni su s izvještajima koje su generirale novije verzije SDV-a, kako bi se utvrdilo pojavljuju li se još uvijek sve greške koje bi se trebale pojavljivati i pojavljuju li se neke nove.

4.3. Provjera spojenih modela testnog skupa pomoću SDV-a

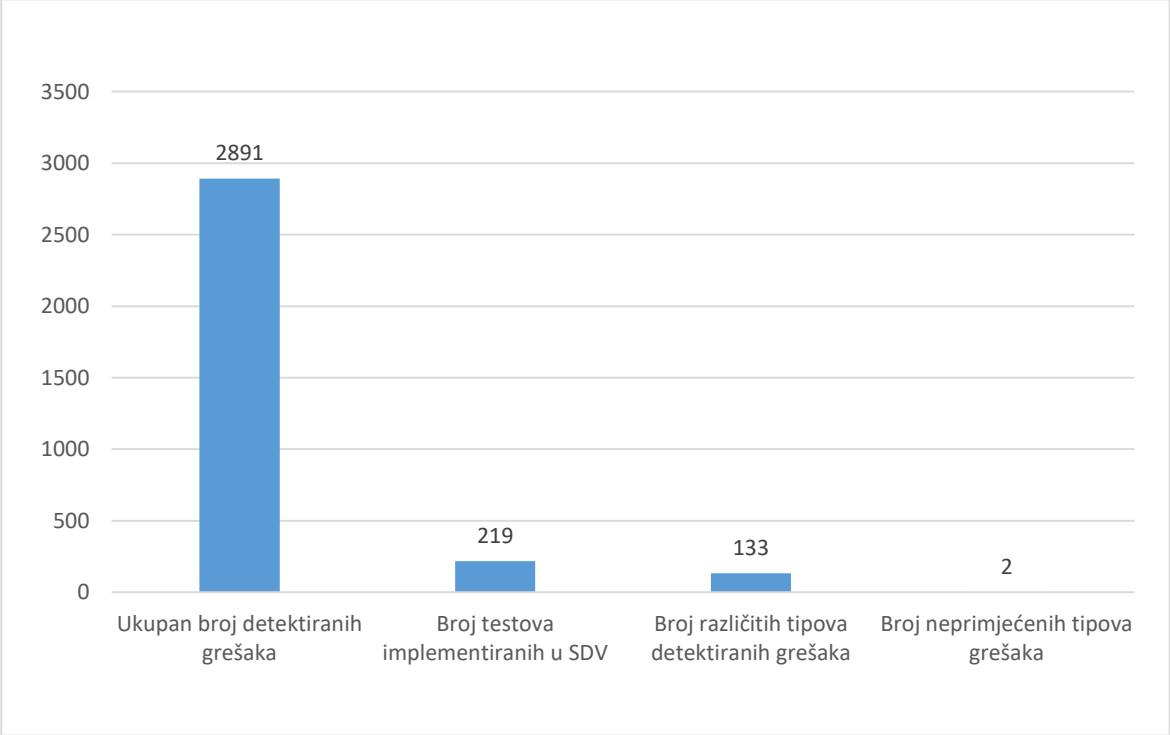
Kao što je spomenuto o prošlom potpoglavlju, DaVinci Developer ne dopušta dodavanje više varijabli, konstanti i dr. istog imena. To otežava izradu modela koji služe za provjeru mogućih grešaka jer se mora direktno manipulirati ARXML datotekom. Alternativan način provjere ispravnosti vezanih testova je spajanjem modela koji sadrže iste varijable, konstante i dr. Spajanjem takvih modela SDV je uspješno prepoznao greške. Iako je u sklopu ovog diplomskog rada spajanje modela korišteno za izazivanje grešaka za koje je, zbog DaVinci Developera, teško napraviti jedan specifičan model, spajanjem modela se može dobiti i u potpunosti točan model, jer se svaka komponenta može definirati kao zaseban model.

4.4. Testiranje testnog skupa modela na novijim verzijama SDV-a

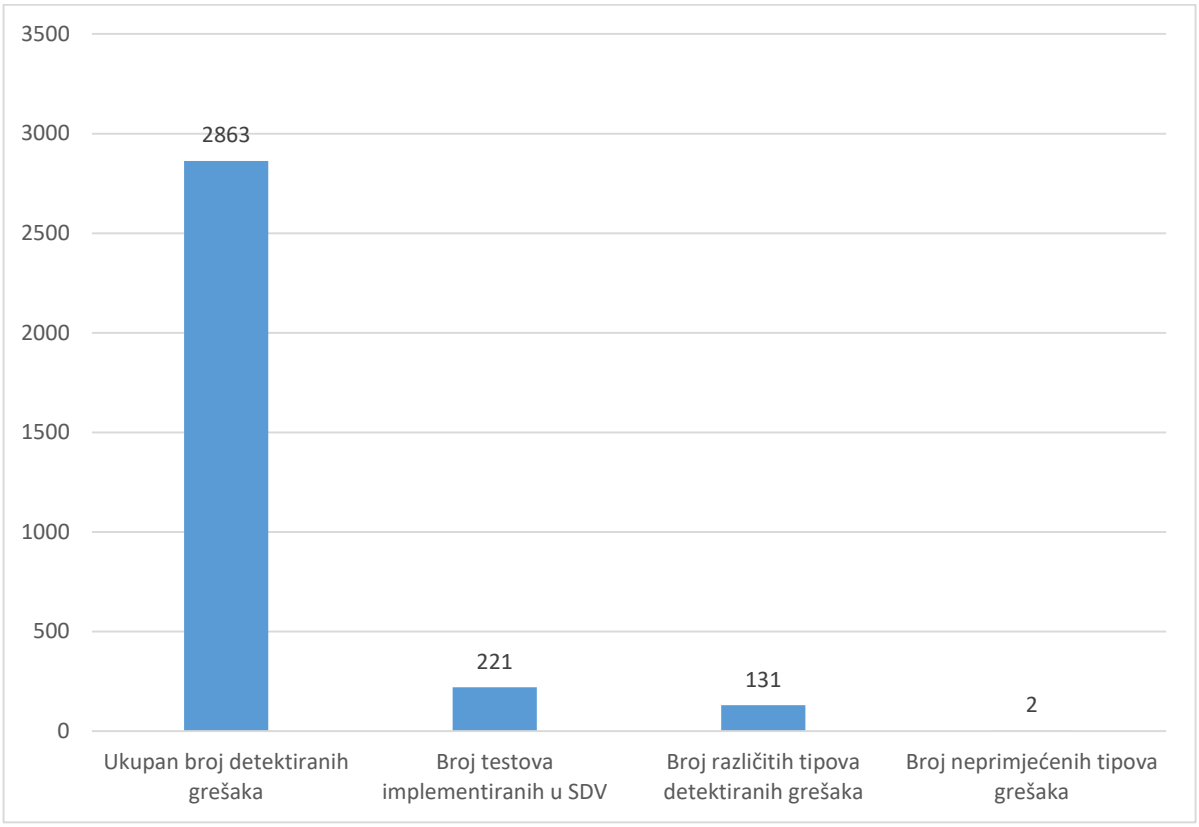
Kako bi se utvrdila korisnost izrađenog testnog skupa modela za ispitivanje novijih verzija SDV-a, modeli su iskorišteni za ispitivanje triju najnovijih verzija SDV-a. U vrijeme faze testiranja modela u sklopu ovog diplomskog rada, to su bile verzije **0.10.4, 0.10.5** i **0.10.6**. Provođenje testiranja na novijim verzijama dovelo je do značajne razlike u rezultatima (grafovi na slikama

4.3, 4.4 i 4.5), u usporedbi sa verzijom **0.9.6**. Primijećena je velika razlika u broju pojedinačnih grešaka, kao i još veća razliku u ukupnom broju grešaka.

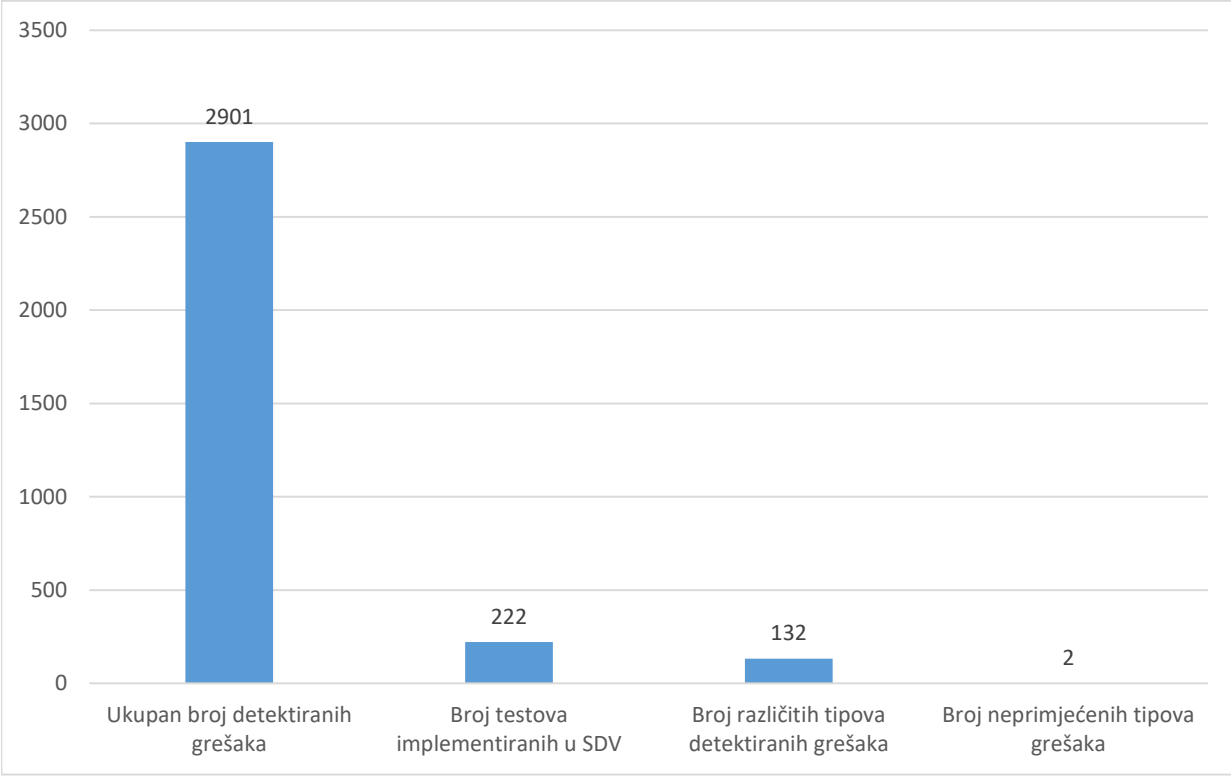
Obje razlike potječu iz značajno većeg broja testova koje provode novije verzije SDV-a, naspram verzije za koju su modeli rađeni. Čak i bez detaljnije analize, dalo bi se zaključiti da se broj grešaka značajno povećao zbog dodataka mnogih testova za zahtjeve kojih bi se svaki model trebao pridržavati, a za koje nisu postojali testovi u verziji **0.9.6**, za koju su modeli originalno rađeni. Treba uzeti u obzir i činjenicu da je verzija **1.3.0** SDG-a koju koristi verzija **0.9.6,** izmijenjena do verzije **1.6.0,** tako da zastarjelost modela može proizaći i iz promjene zahtjeva na SDV. Za usporedbu, dodatno su prikazani podatci na istom grafu (slika 4.6). Na slici 4.6 se može lakše vidjeti razlika u provedenim testovima na svakoj verziji SDV-a. Sve tri novije verzije su dale slične rezultate, uz manju varijaciju. Ono što treba spomenuti je da, ako se pretpostavi da su sve greške detektirane u verziji **0.9.6** prisutne i u novijim verzijama, onda testni skup pokriva **31,98%** (73/222) mogućih grešaka u najnovijoj verziji SDV-a (verzija **0.10.6**). Iako puno manji postotak od onog za verziju za koju su modeli rađeni, svejedno smanjuju potrebu za izradom modela za testiranje gotovo jedne trećine implementiranih testova. Detaljnijom analizom podataka može se utvrditi jesu li greške u izvještajima za verziju **0.9.6** još uvijek prisutne u izvještajima novijih verzija. Također se daju usporediti i pojave iznimki pri pokretanju, i time provjeriti jesu li još uvijek prisutne. Kako bi se navedene tvrdnje potvrdile, provedena je dodatna analiza.



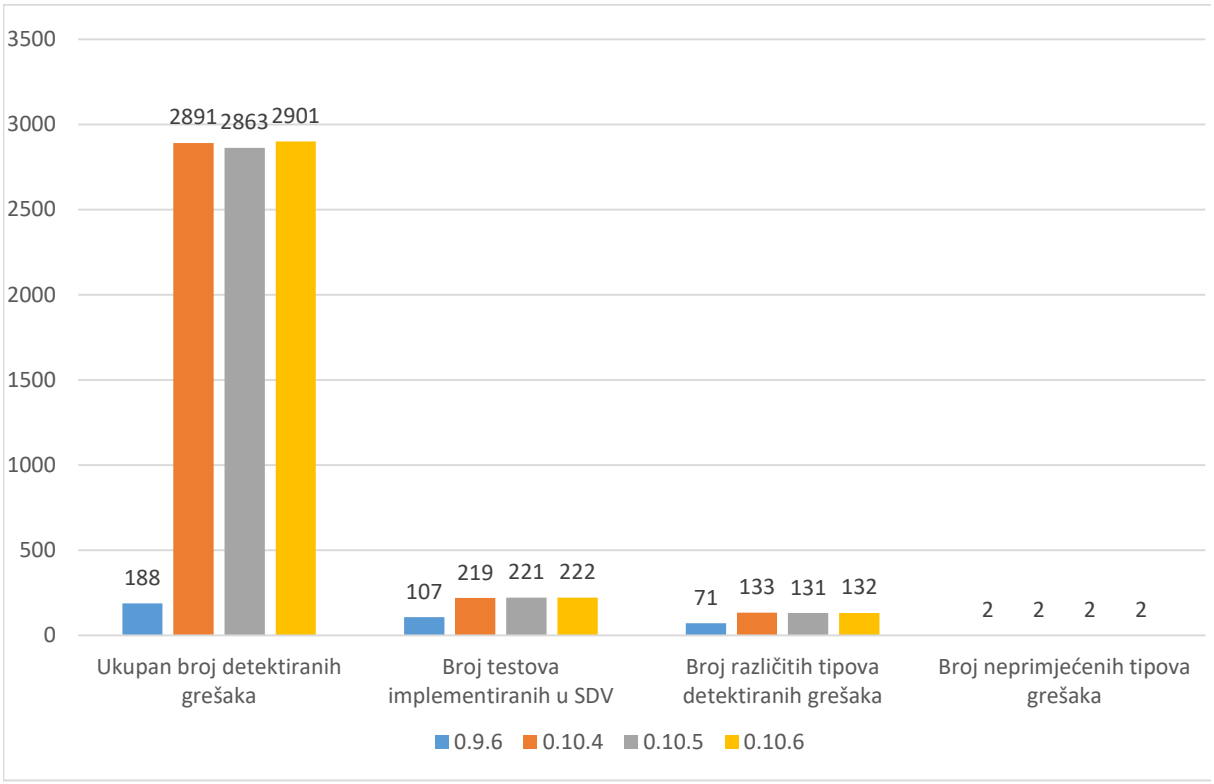
Slika 4.1 Prikaz rezultata pokretanja svih modela testnog skupa na verziji **0.10.4** SDV-a



Slika 4.2 Prikaz rezultata pokretanja svih modela testnog skupa na verziji **0.10.5** SDV-a



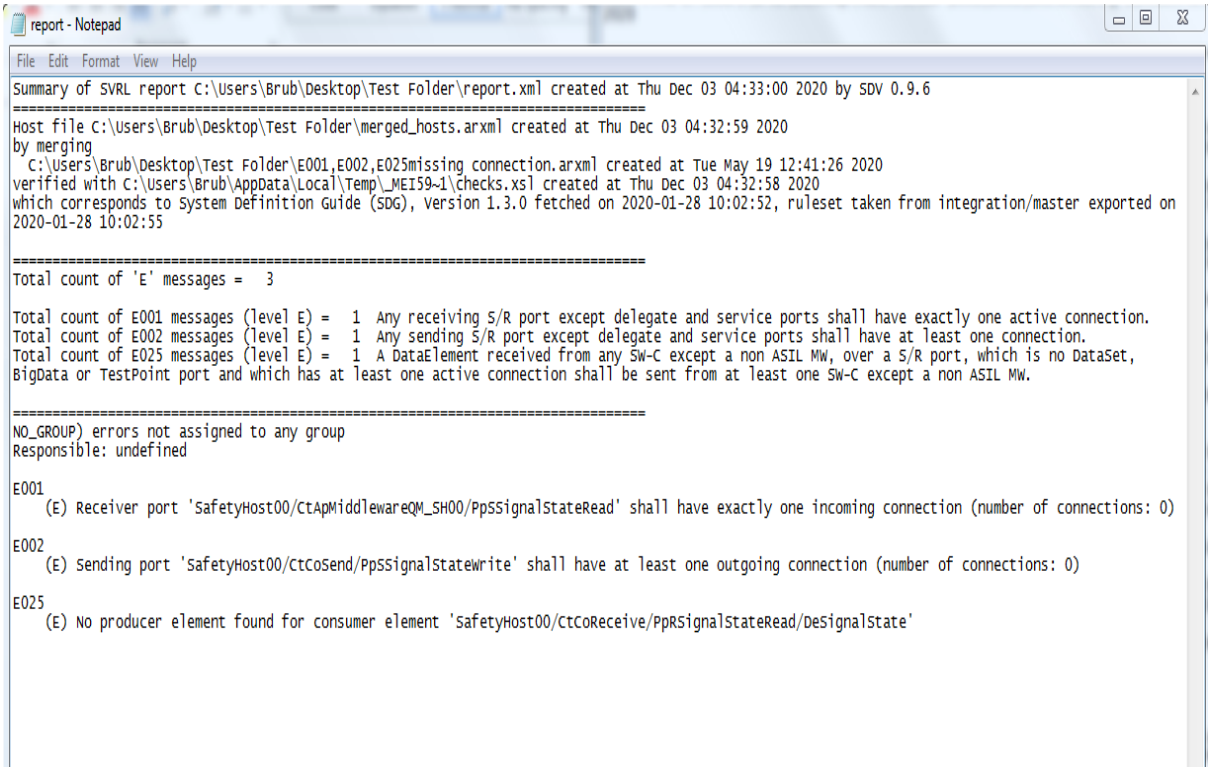
Slika 4.3 Prikaz rezultata pokretanja svih modela testnog skupa na verziji **0.10.6** SDV-a



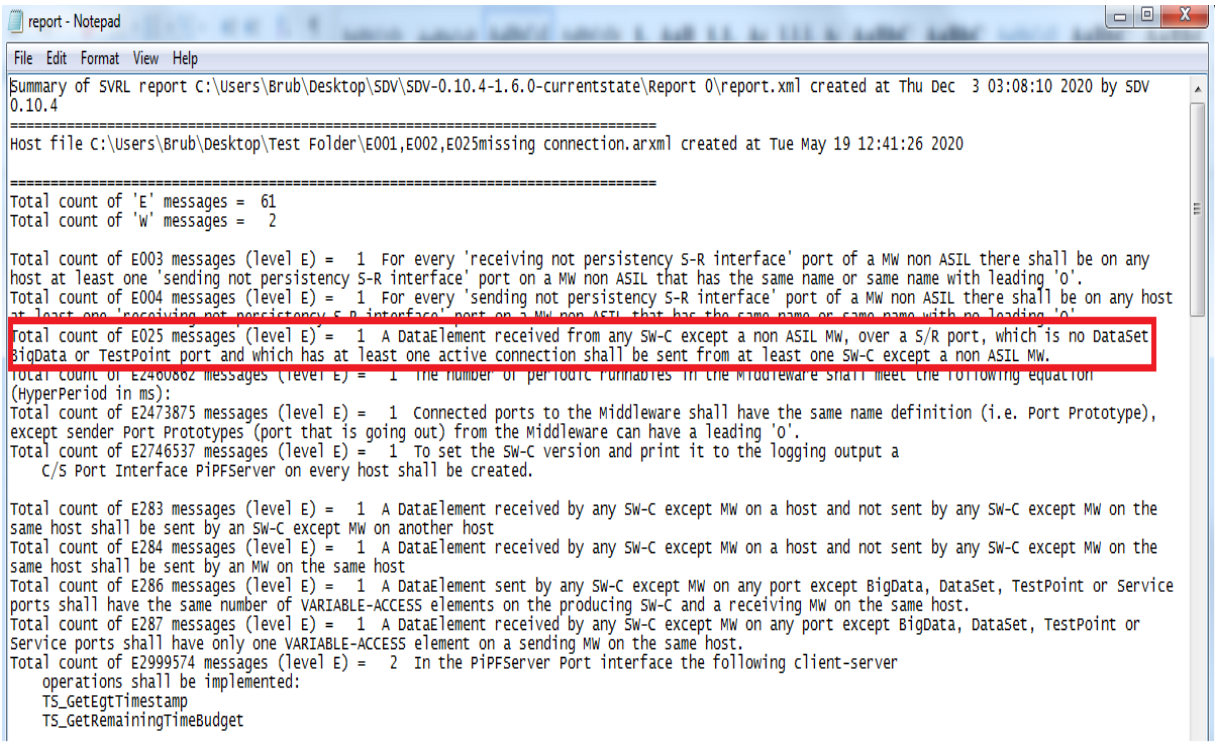
Slika 4.4 Usporedni prikaz rezultata provedenih testova nad testnim skupom uz pomoć verzija **0.9.6**, **0.10.4**, **0.10.5** i **0.10.6** SDV-a

4.5. Usporedba izvještaja generiranih od strane verzija 0.9.6, 0.10.4, 0.10.5 i 0.10.6

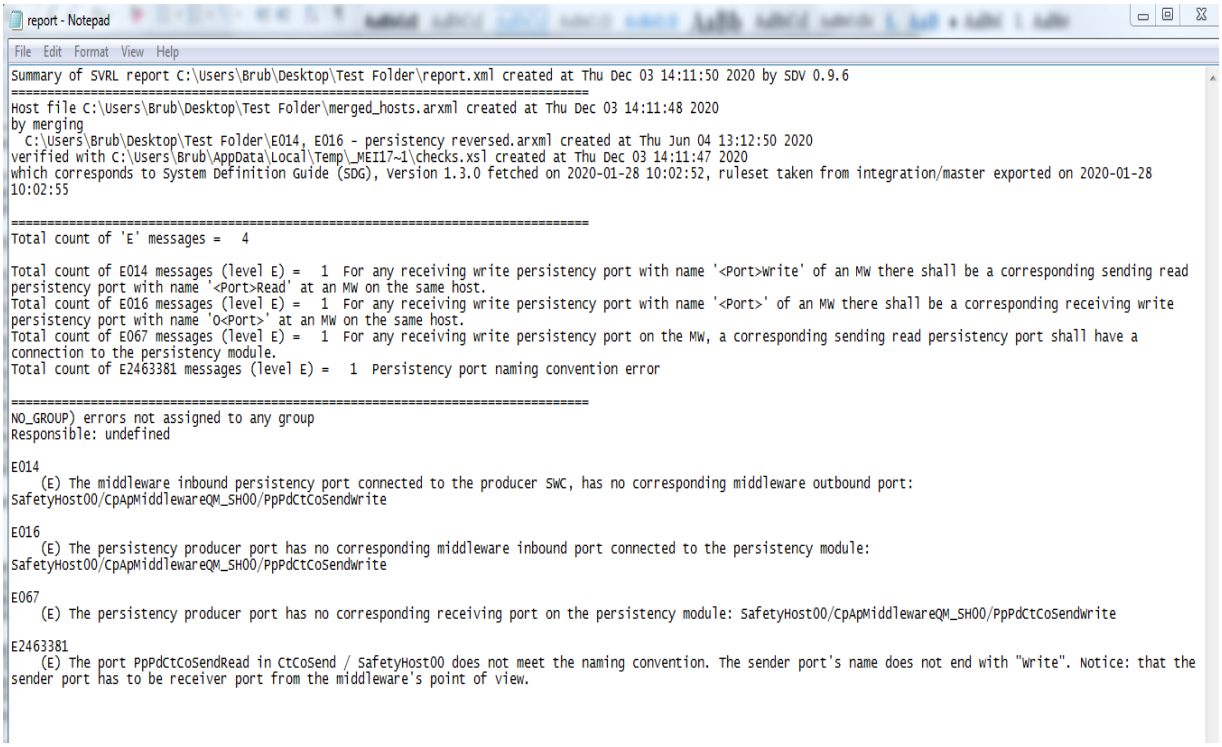
Uspoređeni su izvještaji svih triju novijih verzija SDG-a s izvještajima verzije **0.9.6**, kako bi se provjerilo pojavljuju li se uistinu iste greške. U nastavku su kao dokaz priložena tri primjera. Na prvi pogled nisu sve greške bile prisutne u svim izvještajima. Na slikama 4.7 i 4.8 može se vidjeti da se usporedbom izvještaja analize modela u oba pojavljuje jedino greška E025, dok greške E01 i E02 nisu prisutne. Međutim, detaljnijom analizom se ispostavilo da su samo oznake nekih grešaka izmijenjene. Tako je greška E001 pretvorena u grešku E3609000, a greška E002 u grešku E3609002. Izvještaji verzija **0.10.6** i **0.10.5** su identični verziji **0.10.4** pri ispitivanju istog modela. Kao dodatni primjeri, priloženi su izvještaji na slikama 4.9-4.12, gdje su testirana još dva modela i gdje se može vidjeti da se sve greške na izvještajima starije verzije mogu pronaći i na izvještaju novije verzije. Iste se greške kao i u prvom provjerenom modelu javljaju u verzijama **0.10.5** i **0.10.6**. Iznimke koje su se pojavljivale u nekim modelima se nisu pojavile u verzijama **0.10.5** i **0.10.6.** U verziji **0.10.4** je iznimka za E178 još uvjek prisutna u relevantnim modelima, međutim druge dvije nisu.



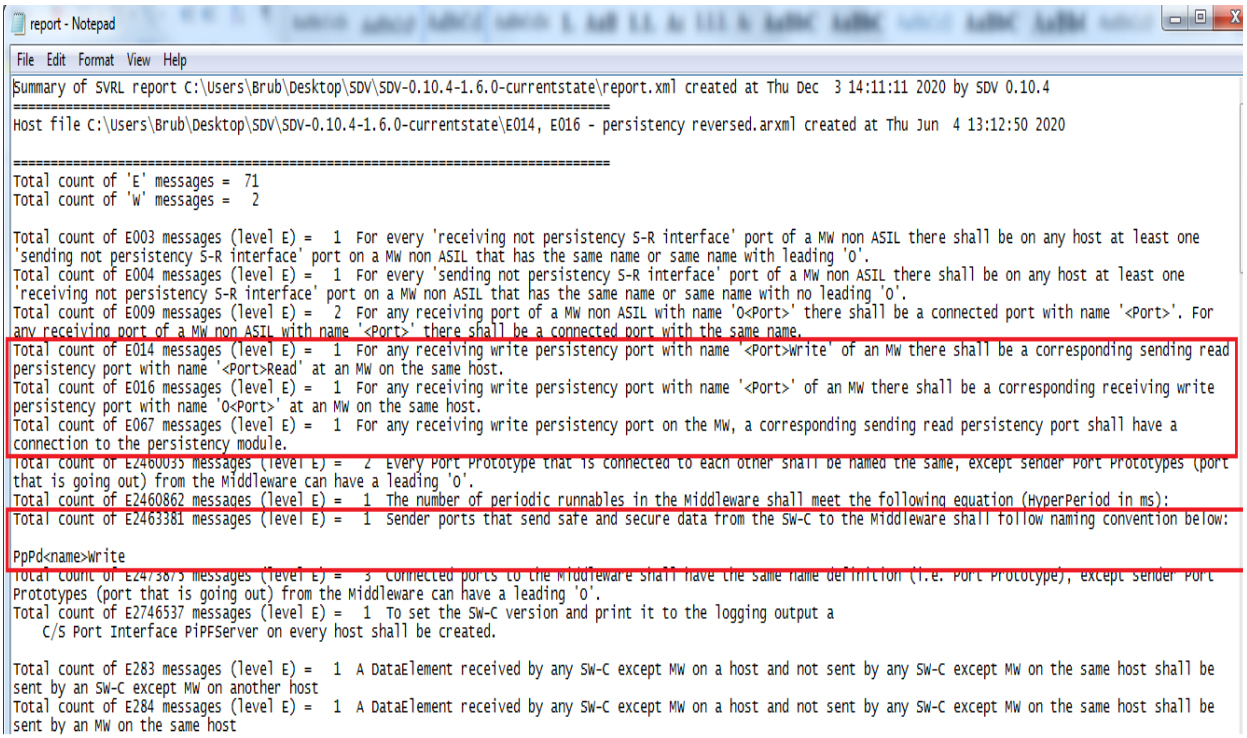
Slika 4.5 Izvještaj generiran ispitivanjem modela koji izaziva greške E001, E002 i E025 koristeći verziju 0.9.6 SDV-a



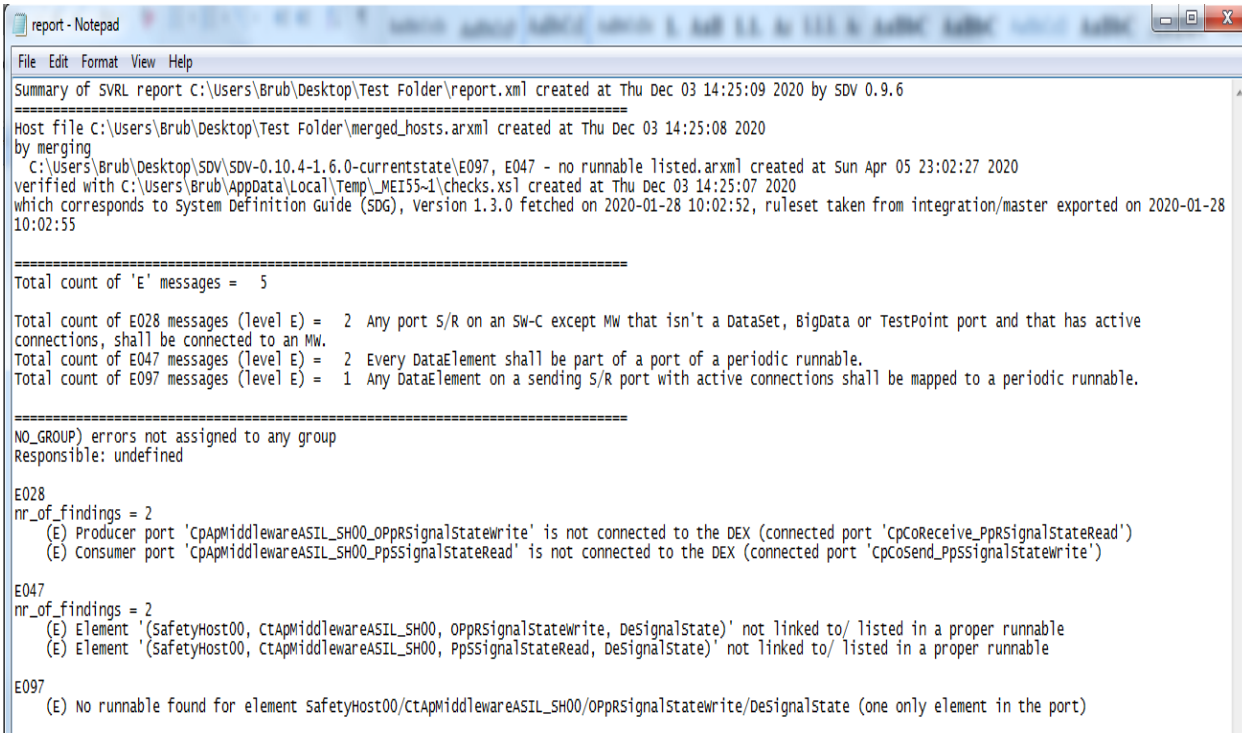
Slika 4.6 Isječak izvještaja generiranog ispitivanjem identičnog modela sa slike 4.7 na verziji 0.10.4 SDV-a sa naznačenom greškom E025



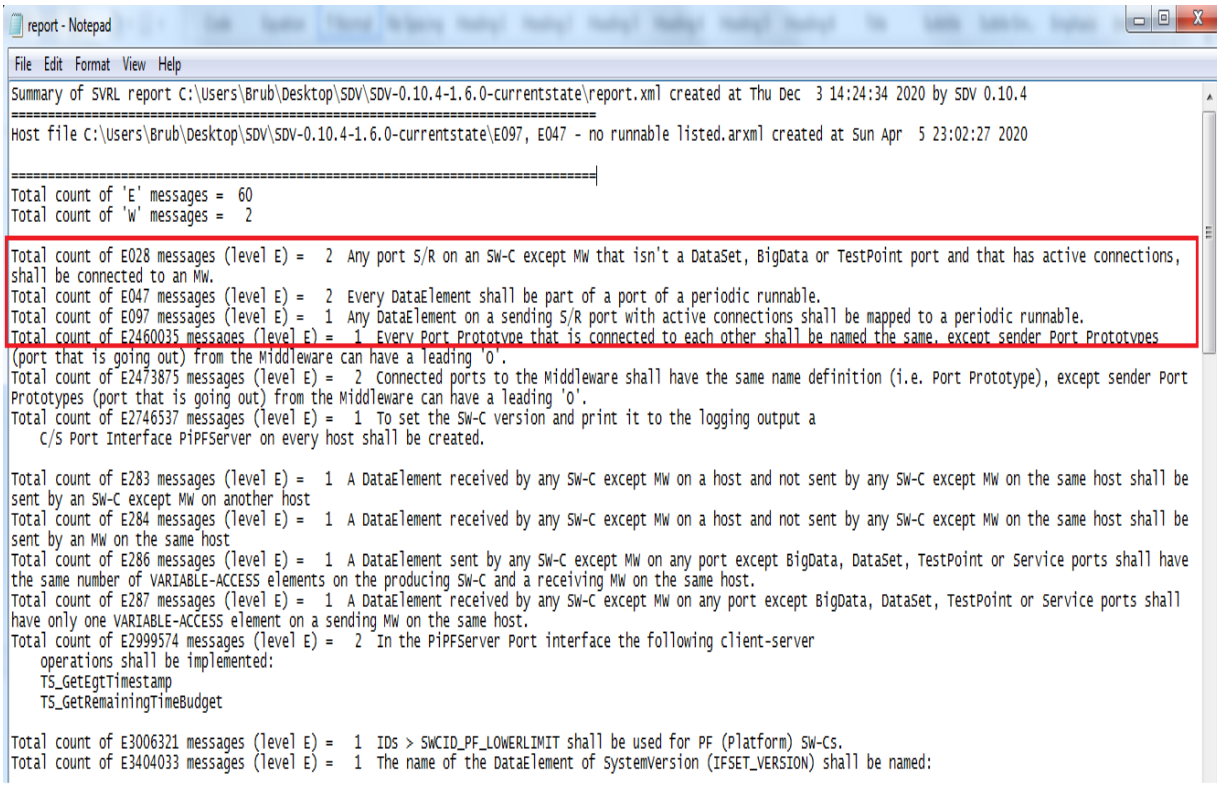
Slika 4.7 Izvještaj generiran ispitivanjem modela koji izaziva greške E2463381, E014, E016 i E067 koristeći verziju **0.9.6** SDV-a



Slika 4.8 Isječak izvještaja generiranog ispitivanjem identičnog modela sa slike 4.9 na verziji **0.10.4** SDV-a sa naznačenim greškama E2463381, E014, E016 i E067



Slika 4.9 Izvještaj generiran ispitivanjem modela koji izaziva greške E028, E047 i E097 koristeći verziju 0.9.6 SDV-a



Slika 4.10 Isječak izvještaja generiranog ispitivanjem identičnog modela sa slike 4.11 na verziji 0.10.4 SDV-a sa naznačenim greškama E028, E047 i E097

4.6. Problemi pri testiranju i moguća rješenja uočenih problema

Iako su izrađeni modeli iskoristivi za provjeru novijih verzija SDV-a, to ne mijenja činjenicu da razlika u provedenim testovima otežava posao. Kako model koji ima svega tri greške u veziji **9.6.0** prelazi u model sa više od 60 kad sa testira s novijim verzijama SDV-a, usporedba izvještaja postaje iscrpna. Prema tome, modele bi potencijalno trebalo nadograditi. Doduše, problem kod takvog pristupa testiranju je to što bi sa svakom novom verzijom SDV-a trebalo nadograditi sve modele. S obzirom da se skup modela napravljen u svrhu ovog diplomskog rada sastoji od 44 modela koji pokrivaju samo dio mogućih testova, nadogradnja svakog modela bila bi mukotrpan posao i nadilazi okvire ovog rada. Također, s obzirom na učestalost izlaska novih verzija alata, potencijalno bi bilo i nemoguće držati korak s izmjenama.

Alternativno rješenje je izraditi opće modele koji ne testiraju neku određenu verziju SDV-ili nadogradnja već postojećih na tu razinu. Takvi modeli bi trebali unaprijed se pridržavati svih zahtjeva opisanih u SDG-u, osim onih čije testovi se provjeravaju. Na taj način bi se izbjegao navedeni problem, gdje se modeli ne pridržavaju nečega što se nije uopće ispitivalo. Iako potencijalno najbolji pristup, problem kod njega je to što je teško znati ispravnost svakog modela bez visoke razine stručnosti. Osima toga, ni ova metoda ne izbjegava nužno potrebu za nadogradnjom modela. Kao i SDV, i SDG se povremeno ažurira, što znači da moguća potreba za nadogradnjom modela ne nestaje.

Treće rješenje ne dolazi sa strane izrade modela, već iz samog načina provođenja testova. Ograničavanje testova koje SDV izvršava na one koji nas zanimaju, značajno bi olakšalo provjeru njihove ispravnosti. Ograničavanjem testova na testove koji su npr. bili prisutni u verziji **0.9.6** SDV-a, omogućava direktnu usporedbu s prijašnjim rezultatima i time znatno ubrzava postupak, jer korisnik unaprijed zna koje bi mu greške alat trebao javiti.

Jedan dodatan problem koji treba razmotriti u sklopu testiranja alata je i činjenica da su svi modeli rađeni isključivo za testiranje i ne odražavaju nužno izgled stvarnih modela. Zbog toga bi uz ovo testiranje bilo poželjno odraditi slično testiranje na već gotovim modelima koji se koriste u nekim proizvodima. Izmjenom spomenutih modela na način da se više ne pridržavaju opisanih zahtjeva, dao bi se bolji uvid u performanse alata, u kontekstu u kojem bi se trebao koristiti, jer bi kompleksnost stvarnih modela također mogla imati utjecaj na funkcionalnost alata.

5. ZAKLJUČAK

Automatska verifikacija modela, zbog kompleksnosti modernih sustava, ima sve veću ulogu u modernom svijetu. U sklopu ovog diplomskog rada je izrađen testni skup modela, čija je svrha bila ispitati jedan takav alat za automatsku verifikaciju. Alat koji je ispitan u ovom radu zove se System Model Verifier i njegova je svrha provjeriti ispravnost modela rađenih po AUTOSAR standardu.

Ispitivanje je provedeno provjerom izrađenih modela na četiri različite verzije SDV-a. Svaki je model rađen tako da sadrži grešku ili greške koje bi SDV trebao registrirati i ispisati ih u izvještaju. Izrađenim modelima pokriveno je **68,22%** mogućih grešaka verzije **0.9.6** SDV-a. Statističkom analizom rezultata dobivenih provjerom modela na verziji **0.9.6** SDV-a može se vidjeti da je detektirao prisutne greške s preciznošću od **97,26%**. Provjerom istih modela na novijim verzijama SDV-a, može se doći do istog zaključka jer su se iste greške pojavile i i na njima, međutim izrađeni modeli nisu u potpunosti prikladni za korištenje uz novije verzije. Testni skup modela pokriva **31,98%** mogućih grešaka u verziji **0.10.6.** Uz sve navedeno i uzimajući u obzir da SDV nije trenutno u svojoj konačnoj verziji, može se zaključiti da SDV dovoljno dobro obavlja provjeru u svojoj trenutnoj verziji, iako ne savršeno.

Kao odgovor na dodatni zahtjev u sklopu ovog diplomskog rada, izrađeno je i grafičko sučelje za SDV, zbog činjenice da u trenutnoj verziji ono ne postoji. Grafičko je sučelje izrađeno tako da korisniku olakša služenje alatom i uključuje dodatnu mogućnost sekvencijalne provjere većeg broja modela, što sami SDV ne podržava. S obzirom da je grafičko sučelje uvelike ubrzalo analizu rezultata svojom sposobnošću praćenja ukupnog broja grešaka, može se reći da je zahtjev na sučelje da olakša služenje alatom također zadovoljen.

LITERATURA

[1] AUTOSAR, »AUTOSAR FAQ,« [Mrežno]. Available: https://www.autosar.org/faq/. [Pokušaj pristupa 23. listopad 2020].

[2] »Autosar Tutorials,« [Mrežno]. Available: https://autosartutorials.com/what-is-autosar/. [Pokušaj pristupa 23. listopad 2020].

[3] AUTOSAR, »AUTOSAR XML Schema production rules,« [Mrežno]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_TPS_XMLSchemaProductionRules.pdf. [Pokušaj pristupa 23. listopad 2020].

[4] dSpace, »dSPACE System architecture,« [Mrežno]. Available: https://www.dspace.com/en/inc/home/products/systems/system_architecture/sd_casestudy_ar-validation.cfm#143_30198. [Pokušaj pristupa 22. listopad 2020].

[5] NATIONAL INSTRUMENTS CORP., »NI,« 5. Ožujak 2019. [Mrežno]. Available: https://www.ni.com/en-rs/innovations/white-papers/11/what-is-the-iso-26262-functional-safety-standard-.html#toc2. [Pokušaj pristupa 27. Listopad 2020].

[6] M. Gros, »BTC embedded systems,« BTC-ES, 2. Svibanj 2019. [Mrežno]. Available: https://www.btc-es.de/en/blog/when-and-how-to-qualify-tools-according-to-iso-26262.html. [Pokušaj pristupa 22. Listopad 2020].

[7] Vector Informatik GmbH, »Vector,« Vector Informatik GmbH, [Mrežno]. Available: https://www.vector.com/int/en/. [Pokušaj pristupa 2 12 2020].

[8] W3Schools, »Introduction to XML,« [Mrežno]. Available: https://www.w3schools.com/xml/xml_whatis.asp. [Pokušaj pristupa 15. 11. 2020.].

[9] W3Schools, »W3Schools.com,« [Mrežno]. Available: https://www.w3schools.com/cs/cs_intro.asp. [Pokušaj pristupa 2 11 2020].

[10] Microsoft, »A tour of the C# language,« [Mrežno]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Pokušaj pristupa 4 11 2020].

[11] Microsoft, »Language Integrated Query (LINQ),« 11 30 2016. [Mrežno]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/linq/. [Pokušaj pristupa 4 11 2020].

[12] Microsoft, »Desktop Guide (Windows Forms .NET),« 10 26 2020. [Mrežno]. Available: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-5.0. [Pokušaj pristupa 4 11 2020].

[13] C. R. Prause, R. Gerlich i R. Gerlich, »Evaluating Automated Software Verification Tools,« u *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, Vasteras, Švedska, 2018.

SAŽETAK

U svrhu testiranja verifikatora modela za provjeru programske podrške u automotiv industriji, definirana je potreba za izradom testnog skupa modela za provjeru System model verifier (SDV) alata. Uz izradu modela i testiranje alata na istima, definirana je i potreba za izradom grafičkog sučelja za alat. Izrađeni modeli su iskorišteni za testiranje nekoliko verzija SDV -a kako bi se utvrdila njihova korisnost i ispravnost samog alata. Izrađeno grafičko sučelje je iskorišteno za ubrzanje analize alata čime je također dokazana i njegova korisnost.

Ključne riječi: AUTOSAR, DaVinci Developer, grafičko sučelje, testiranje, verifikator modela

TESTING OF MODEL VERIFIER WITHIN PROGRAM DEVELOPMENT IN AUTOMOTIVE INDUSTRY

ABSTRACT

For the purpose of testing a model verifier for checking software in the automotive industry a need for the creation of a group of test models for validating the System model verifier tool (SDV) has been defined. Together with the creation of the models and the testing of the tool, a need for the creation of a graphical user interface has been defined as well. The created models were used to test several versions of the SDV for confirming the usefulness and correctness of the tool. The created graphical user interface was used to speed up the analysis of the tool, therefore proving its own usefulness.

Key words: AUTOSAR, DaVinci Developer, graphical user interface, testing, model verifier

ŽIVOTOPIS

Bruno Kapular rođen je 21.1.1997. u Vinkovcima. Pohađao je osnovnu školu u Starim Jankovcima. Nakon osnovne škole je pohađao opći smjer u „Gimnaziji Matije Antuna Reljkovića“ u Vinkovcima. Završio je preddiplomski smjer računarstva na „Fakultetu elektrotehnike, računarstva i informacijskih tehnologija“ u Osijeku koji je u sklopu Sveučilišta Josipa Jurja Strossmayera. Nakon završetka preddiplomskog studija, upisao je diplomski studij računarstva na istom sveučilištu.