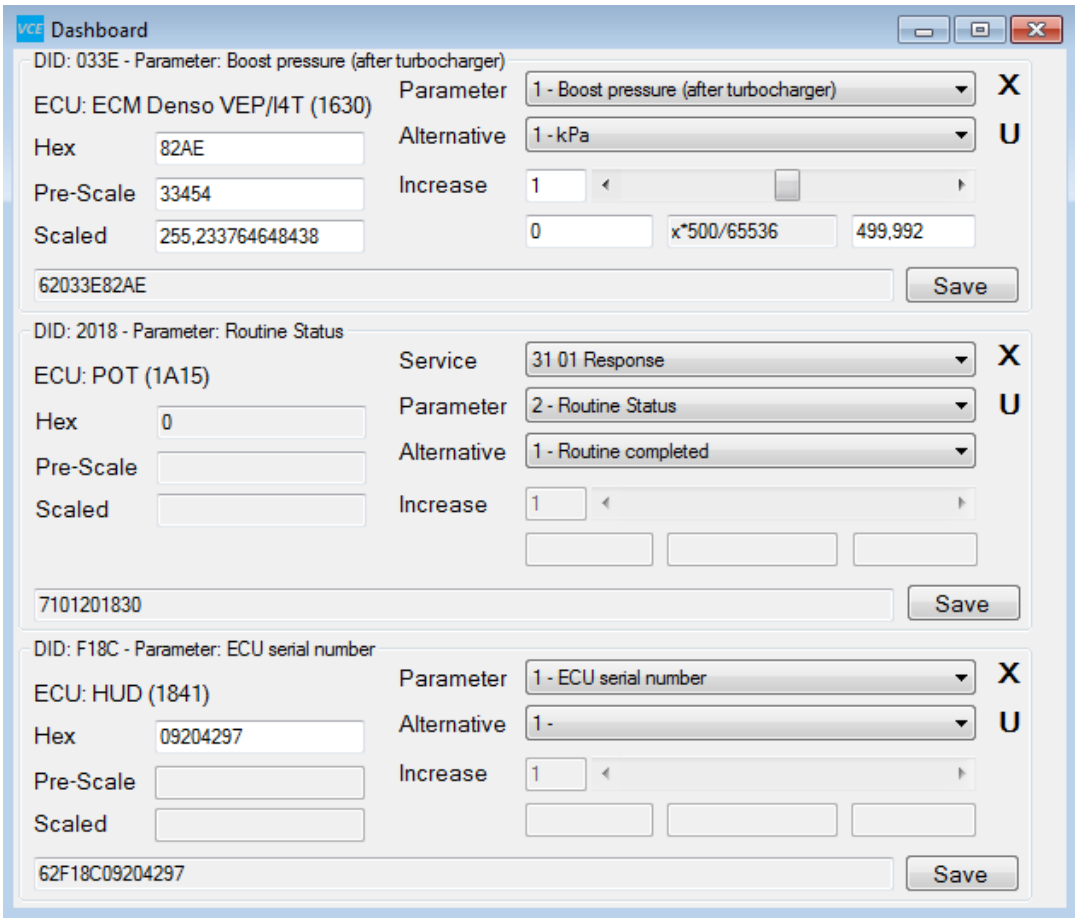
CHALMERS



Virtual Car Editor

Examensarbete inom Högskoleingenjörsprogrammet i Data- och Informationsteknik

ALBIN HELLQVIST

TASDIKUL HUDA

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2017

EXAMENSARBETE 2017

Virtual Car Editor

ALBIN HELLQVIST

TASDIKUL HUDA

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2017

Virtual Car Editor
ALBIN HELLQVIST
TASDIKUL HUDA

Examinator: Peter Lundin

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola / Göteborgs Universitet
412 96 Göteborg, Sverige
Telefon: 031-772 1000

Omslag:
Modifieringen av en parameter i en virtuell bil.

Institutionen för Data- och Informationsteknik
Göteborg 2017

Virtual Car Editor

ALBIN HELLQVIST
TASDIKUL HUDA
Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola / Göteborgs Universitet

Sammanfattning

Volvo Cars är en av världens mest ledande biltillverkare som producerar flera hundratusentals bilar per år. De bilarna som produceras behöver bland annat felsökas och diagnostiseras. I Göteborg, Sverige ansvarar gruppen Diagnostics & dependability för att leverera felsökning samt diagnostik av bilar, till verkstäder över hela världen och det är i samarbete med denna grupp som detta projektarbete har genomförts.

För att kunna felsöka bilar utvecklar gruppen testskript som används för att skicka förfrågningar till en provbil samt logga dess respons. Exempel på förfrågningar kan vara om bilens nuvarande motorvarvtal eller totalt färdad sträcka. De skripten som utvecklas kan även prövas mot en virtuell bil, som i detta sammanhang är en XML-fil bestående av svar som motsvarar en riktig bil vars respons sparats vid ett visst tillfälle.

I dagsläget kan det vara väldigt omständligt att hantera virtuella bilar eftersom det kräver att man måste handskas med XML-filer och utföra allt arbete i dessa för hand. Denna metod blir alltmer tidskrävande i samband med att innehållet i XML-filen växer i storlek. För att underlätta det vardagliga arbetet för Diagnostics & dependability blir syftet med detta projekt att göra hantering av virtuella bilar mer kraftfull och smidig.

Resultatet av detta projekt är en applikation utvecklad i Visual Studio med programmeringsspråket C#. Denna applikation innehåller tre huvudfunktioner avsedda till att generera, sammanfoga samt modifiera virtuella bilar. Applikationen som har utvecklats är viktig i det avseendet att man inte längre behöver handskas med dessa XML-filer i samma utsträckning som tidigare.

Nyckelord: DID, DTC, ECU, kontrollrutin, SPA-plattformen, test-skript, virtuella bilar, Volvo Cars

Abstract

Volvo Cars is one of the most leading car manufacturer in the world and they produce over several hundreds of thousands cars a year. The cars produced need among other things troubleshooting and diagnostics. This project has been executed in cooperation with Diagnostics & dependability, which is a section within Volvo Cars, responsible for delivering car diagnostics and troubleshooting information to workshops all over the world.

In order to troubleshoot the cars, the group develop scripts used to send requests to a test car and log the obtained response. Examples of requests can be the car's engine's current revolutions per minute or total distance travelled. The scripts developed can also be tested against a virtual car which in this context is an XML file consisting of a set of requests and responses, corresponding to a car's requests and responses.

Managing virtual cars today can be very tedious since it requires that you have to deal with XML-files by hand in order to perform necessary work. This method is not only inefficient but also very time consuming, especially if the content of a XML file is large. To facilitate the working process for Diagnostics & dependability, this project aimed to develop a more efficient method for managing virtual cars.

The result of this project is an application developed in Visual Studio using C# as the programming language. This application consists of three main functions that allow a user to generate, merge and modify virtual cars. The application developed is important in the matter that it is no longer required to deal with XML files in the same extension as before.

Keywords: Control routine, DID, DTC, ECU, SPA platform, test script, virtual cars, Volvo Cars

Förord

Det fanns ett behov för de på gruppen Diagnostics & dependability inom Volvo Cars att göra hanteringen av virtuella bilar enklare. En i gruppen programmerade en applikation för att underlätta detta men det saknades tid och motivation för att göra den fullständig, dessutom var den inte kopplad mot Volvos databas. En nyutveckling av denna applikation föreslogs då av John Hellqvist (bror till Albin Hellqvist) som examensarbete. Med hjälp från gruppens chef Ulf Edvardsson, John Hellqvist, Fredrik Fridmar samt Erik Skoog så kunde ett beslut tas vilket gjorde detta möjligt.

Examensarbetet är utfört på Volvo Cars i samarbete med gruppen Diagnostics & dependability. För att bli examinerad inom programmet Datateknik 180 hp vid Chalmers tekniska högskola är examensarbetet ett obligatoriskt moment som omfattar 15 högskolepoäng inom Data- och Informationsteknik.

Vi vill tacka John Hellqvist som kommit med förslaget till examensarbetet, Fredrik Fridmar som varit vår handledare inom Volvo Cars, Erik Skoog som kommit på idén för applikationen, Ulf Edvardsson som var gruppens förra chef, Henric Jansson som är gruppens nuvarande chef samt Jan Jonsson som varit vår handledare på Chalmers och som hjälpt till med den akademiska biten. Vi vill även tacka ovanstående samt resten av de på Diagnostics & dependability för deras tekniska support och synpunkter.

Innehållsförteckning

Förkortningslista

DID - Data Identifier

DTC - Diagnostic Trouble Code

ECU - Electronic Control Unit

NRC - Negative Response Code

SOP - Start of production

SPA - Volvo Scalable Product Architecture

SPASE - SPA Script Editor

VIDA - Vehicle Information and Diagnostics for Aftersales

VIN - Vehicle Identification Number

1. INLEDNING

Volvo Cars är en av världens mest ledande biltillverkare med inte bara tillverkning i Sverige, utan även i flera andra länder såsom Belgien, Kina och Malaysia. Idag ägs dock inte Volvo Cars av Aktiebolaget Volvo utan ägs av det kinesiska företaget Zhejiang Geely Holding Group. Huvudkvarteret ligger fortfarande i Göteborg i Sverige, och det är även här som detta projekt har genomförts i samarbete med gruppen Diagnostics & dependability.[1]

1.1. Bakgrund

Gruppen Diagnostics & dependability ansvarar för att leverera heltäckande stöd för felsökning och diagnostik till verkstäderna världen över. Detta så att mekanikerna kan både felsöka samt åtgärda symptom för kundernas bilar på ett tids- och kostnadseffektivt sätt.

En stor del av felsökningarna kräver testskript, vilka gruppen Diagnostics & dependability utvecklar. Dessa skript skickar förfrågningar till en provbil samt loggar dess respons. Skriptet kan till exempel fråga bilen om dess nuvarande varvtal eller däcktryck. Dessa skript behöver dock inte enbart prövas mot en riktig bil, utan kan även prövas mot en virtuell bil. En virtuell bil är en XML-fil som innehåller färdiga förfrågningar samt svar som motsvarar en riktig bils förfrågningar samt svar.[2] En riktig bils svar ändras dock under tiden eftersom den är i drift medan den virtuella bilen har statiska svar.

1.2. Syfte

Syftet med projektet är att göra hanteringen av virtuella bilar mer kraftfull och användbar så att de verifieringarna som utförs i nuläget, kan genomföras med en bättre effektivitet och tidsmarginal. Som tidigare nämnt har den virtuella bilen statiska svar, och i nuläget måste de på Diagnostics & dependability ändra i XML-filen för hand, vilket kan vara tidskrävande. Gruppen vill att detta skall ersättas med en dynamisk lösning som underlättar deras vardagliga arbete.

[1] http://assets.volvocars.com/se/~/media/shared-assets/downloads/this-is-volvo/volvo-in-brief2016.pdf
[2] Virtuell bil förklaras mer ingående i "Teknisk Bakgrund"

1.3. Mål

Det övergripande målet med detta projekt är att utveckla en applikation, bestående av följande tre huvudfunktioner:

- Generera virtuella bilar
- Sammanfoga virtuella bilar
- Modifiera virtuella bilar

1.3.1. Generera virtuella bilar

Generering av virtuella bilar skall ge användaren möjlighet till att skapa virtuella bilar i form av XML-filer, utefter den dokumenterade diagnosspecifikationen för ett specifikt bilprojekt. Dessa specifikationer skall kunna hämtas automatiskt från CarComs databas som beskrivs i "Teknisk bakgrund". Funktionen skall dessutom ge möjligheten att lägga till eller ta bort förfrågningar i befintliga virtuella bilar. Till denna del kan det tillkomma ytterligare funktioner under förutsättningen att tid finnes. Exempel på en sådan funktion är att de genererade bilarna skall fungera i Volvos program VIDA[3]. Implementation för generering av virtuella bilar beskrivs i avsnitt "4.3.1. Generering".

1.3.2. Sammanfoga virtuella bilar

Sammanfogning av virtuella bilar skall kunna göras på två olika sätt. En användare skall få möjligheten att antingen välja förfrågningar med svar från existerande virtuella bilar och sammanfoga dessa till en ny bil, eller låta programmet automatiskt välja alla unika förfrågningar med tillhörande svar. Implementeringen för denna funktion förklaras i avsnitt "4.3.2. Sammanfogning".

1.3.3. Modifiera virtuella bilar

En virtuell bil skall kunna modifieras på olika sätt. När en virtuell bil har laddats in så kommer programmet att läsa av vilka förfrågningar som finns i bilen samt respons för respektive förfrågan. En användare skall därifrån kunna modifiera responsen för de förfrågningarna som existerar i den virtuella bilen, välja parametrar som förfrågningarna består av och justera värden för dessa parametrar. Modifiering av virtuella bilar förklaras med ingående i avsnitt "4.3.3. Modifiering".

[3] VIDA förklaras under avsnittet "Teknisk bakgrund".

1.4. Precisering av frågeställning

Detta avsnitt specificerar de övergripande frågorna som skall besvaras under detta arbete.

- Hur kan hantering av virtuella bilar göras mer kraftfull och smidig? Detta besvaras i samband med resultaten i avsnitt 5.
- På vilket sätt kommer applikationen att underlätta det vardagliga arbetet för dem på gruppen Diagnostics & dependability? Detta besvaras i samband med diskussion och slutsatser i avsnitt 6.
- Kommer denna applikation att kunna tillämpas av andra aktörer än enbart Diagnostics & dependability? Detta besvaras i samband med diskussion och slutsatser i avsnitt 6.

1.5. Avgränsningar

Detta projekt kommer att vara avgränsat till programmering i Visual Studio och applikationen skall utvecklas med programmeringsspråket C#. Anledning till detta val av programmeringsspråk är att andra program som de i Diagnostics & dependability använder är utvecklade med C#, vilket gör applikationen lättare att vidareutveckla.

Eftersom det finns många olika bilplattformar så kommer detta projekt att avgränsas till den plattform som Diagnostics & dependability arbetar med, det vill säga SPA-plattformen. Exempel på några bilar som denna plattform innefattar är Volvo S90, V90, XC60 och XC90.

1.6. Rapportens disposition

I avsnitt 2 förklaras de begreppen som är nödvändiga för att förstå virtuella bilars struktur och de programmen som har använts under detta projekt. I avsnitt 3 beskrivs arbetets uppdelning och tillvägagångssätt. I avsnitt 4 förklaras databashantering, implementerade funktioner och dess samhörighet med det grafiska gränssnittet. I avsnitt 5 sammanställs de resultaten som har uppnåtts under detta projekt och i avsnitt 6 diskuteras vilka slutsatser som kan dras av dessa resultat. I avsnitt 7 ges förslag på förbättringar och ytterligare funktioner som kan implementeras inför framtida utveckling av applikationen.

2. TEKNISK BAKGRUND

Detta examensarbete har genomförts på företaget Volvo Cars i samarbete med gruppen Diagnostics & dependability. Gruppen arbetar mestadels med programmen CarCom II, SPASE och VIDA men innan dessa beskrivs ges en kort förklaring av en bils struktur ur ett perspektiv som är relevant för denna rapport. Sedan kommer en förklaring om hur virtuella bilar är uppbyggda.

2.1. En bils diagnostiska utläsning

Detta avsnitt är avsett till att förklara en bils diagnostiska utläsning.

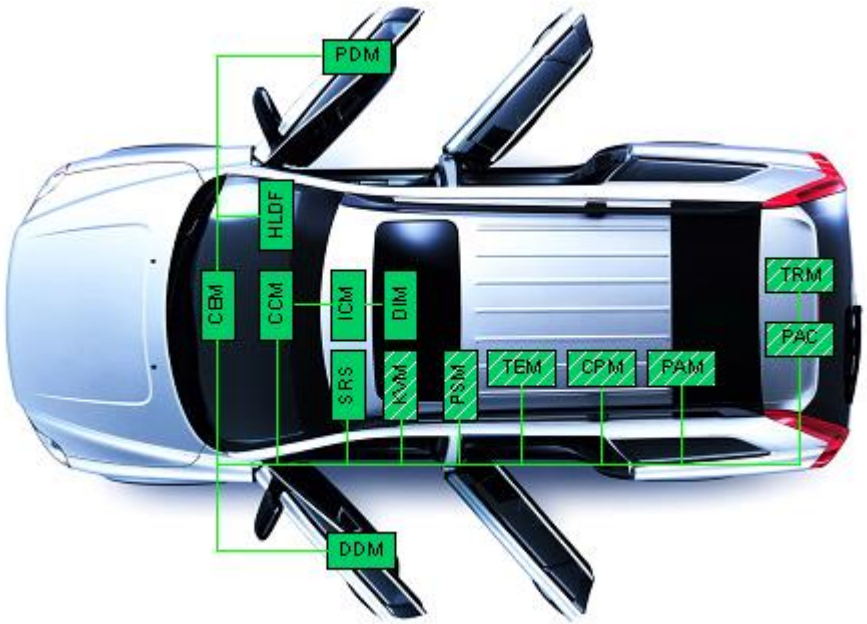


Bild 1: Illustration av ECU:ers placering i en äldre bil.

En modern bil kan innehålla över 40 stycken styrenheter (vilket i fortsättningen kommer att benämnas som ECU). Varje ECU har ett diagnosnummer kopplat till sig och fungerar som ett unikt id. Varje ECU har hand om ett eller flera områden, till exempel finns en ECU, Supplemental Restraint System (SRS), som hanterar passiv säkerhet i bilen såsom airbags och bältesspänning vid krock. ECU:erna samlar information från olika givare och mätpunkter till det som kallas DID:s. Dessa DID:s innehåller således svar med ett visst antal bitar, allt från en bit till över tusen bitar och innehåller information om till exempel bilens varvtal eller bromstryck. En DID kan ha flera parametrar och dessa motsvarar en viss del av en DID:s svar. Det kan exempelvis finnas en DID med två parametrar, där parameter ett mäter bromstrycket till det främre vänstra hjulet medan parameter två mäter bromstrycket till det främre högra hjulet.

En ECU innehåller även DTC:er och kontrollrutiner (Control Routines). DTC:er hanterar felkoder och fungerar som en slags "flagga" som kan sättas beroende på fel som uppkommit. Vissa DTC:er har även hantering för något som kallas "snapshots" vilket är att den sparar värden från några specifika DID:s när felet uppkom. Det finns också vissa DTC:er som har stöd för något som kallas "extended data" och detta är en respons som innehåller räknare för att monitorera hur många gånger felet har uppkommit. Kontrollrutiner är diverse inbyggda tester som kan köras för vissa ECU:er och det kan vara test som till exempel prövar att ventilen för bränslelocket öppnas och stängs korrekt.

För att läsa ut samt använda DID:s, DTC:er och kontrollrutiner i diagnostiskt syfte från ECU:erna använder man sig utav "services", även kallat tjänster. Varje ECU stödjer ett visst antal tjänster och de tjänster som är relevanta för detta projekt beskrivs i Tabell 1 nedan.

Tabell 1: Förklaring av de tjänster som är relevanta för detta projekt.[4]

Tjänst	Namn	Beskrivning
19 02	Report DTC By Status Mask	Används tillsammans med medskickad hexadecimal mask för att efterfråga alla DTC:er vars statusflagga matchar den givna maskningen.
19 03	Report DTC Snapshot Identification	Används för att efterfråga alla DTC:er som använder sig utav tjänst 19 04.
19 04	Report DTC Snapshot Record By DTC Number	Används för att få en specifik DTCs snapshots DID:s.
19 06	Report DTC Extended Data Record By DTC Number	Används för att få en specifik DTC:s extended data.
22	Read Data By Identifier	Används för att efterfråga svar från specificerad DID.

[4] https://automotive.softing.com/fileadmin/sof-files/pdf/de/ae/poster/UDS_Faltposter_softing2016.pdf

31 01	Routine Control Start Routine	Används för att starta en kontrollrutin och ibland skickas det även med information som till exempel säger hur länge kontrollrutinen ska köras.
31 02	Routine Control Stop Routine	Används för att stänga av en pågående kontrollrutin.
31 03	Routine Control Request Routine Results	Används för att få resultat från en kontrollrutin, till exempel om denne lyckades med sitt test.

När en tjänst används så behöver den tillfrågade ECU:n nödvändigtvis inte svara med det som efterfrågades, utan kan ibland svara med en Negative Response Code (NRC) istället. En NRC är en slags kod som säger anledningen till varför ECU:n inte svarade på efterfrågan. Ett exempel på detta är NRC nummer 11 som säger att den använda tjänsten inte stöds av den ECU:n.[5]

2.2. Programvara

I detta avsnitt beskrivs de programmen som har varit relevanta för detta projekt.

CarCom II är ett program som används för att läsa och editera dess databas. Databasen innehåller information för att få diagnosnummer (ECU:er) för valt projekt, se tillhörande DID:s, DTC:er, kontrollrutiner etcetera. Databasen innehåller även information om inom vilka ramar svaren förväntas vara vid förfrågningar från SPASE och utan databasen hade inte kommunikationen med bilar varit möjlig.

SPASE används för att skapa skript som skickar förfrågningar till en specifik bil samt loggar dess respons. Detta kan till exempel vara att den efterfrågar om nuvarande varvtal eller däcktryck för den valda bilen. SPASE har även en möjlighet som det utvecklade programmet har, att man kan välja "default car config". Detta är ett sätt att få ut alla diagnosnummer (ECU:er) från en vald biltyp såsom Volvo V90 -17 istället för att få ut detta från ett valt projekt.

[5] https://automotive.softing.com/fileadmin/sof-files/pdf/de/ae/poster/UDS_Faltposter_softing2016.pdf

VIDA är ett program som används utav verkstäder och återförsäljare över hela världen. Programmet innehåller allt från priser på Volvos produkter och verkstadsmanualer till hantering av mjukvaruuppdateringar. För detta projekt är VIDA relevant i det avseendet att VIDA tillåter utläsning av såväl riktiga som virtuella bilar. VIDA är alltså ett bra verktyg för att verifiera ifall den virtuella bilen fungerar korrekt.

2.3. Struktur för virtuell bil

Detta avsnitt beskriver strukturen för en virtuell bil samt hur innehållet i en virtuell bil avläses.

```
<ecus>
  <ecu address="1212" name="AUD">
    <request message="190220">
      <response>5902FF9A021220</response>
    </request>
    <request message="1903">
      <response>59039A021220</response>
    </request>
    <request message="19049A021220">
      <response>59049A0212202005800000000D13400</response>
    </request>
    <request message="22D134">
      <response>62D13400</response>
    </request>
    <request message="3101600B">
      <response>7101600B00000000</response>
    </request>
  </ecu>
</ecus>
```

Bild 2: Virtuell bil i XML-format.

Bild 2 ovan illustrerar strukturen för en virtuell bil i XML-format. En virtuell bil består utav en uppsättning av ECU:er och varje ECU innehåller en eller flera förfrågningar. En förfrågan representerar den information som har efterfrågats för en specifik ECU, samt vilken tjänst den efterfrågade informationen har. Det kan exempelvis vara information om den totala sträckan som en bil har färdats, felkod som har uppkommit i något tillfälle eller information om en viss kontrollrutin som finns i ECU:n.

De första tecknen i förfrågningen indikerar på vilken tjänst som används för att få den efterfrågade informationen. De tjänster som hanteras i detta projekt består utav två till fyra tecken. Om en tjänst används för att efterfråga en DID eller kontrollrutin (tjänst 22 respektive 31 01 i detta exempel) brukar de tecken som kommer efter tjänsten representera identifikationen för efterfrågad information. För att avgöra identifikationen för DTC:er däremot så behöver programmet avläsa responsen ifall den använda tjänsten är 19 02 eller 19 03. Om tjänsten istället är 19 04 eller 19 06 kan en DTC identifieras utifrån de sex tecknen som kommer efter tjänsten i förfrågningen.

För varje förfrågan finns det även en tillhörande respons. Värdet som finns i de två första tecknen i responsen är alltid värdet på två första tecknen för tjänsten adderat med hexadecimalt 40. Sedan följer responsen av identifieraren för den efterfrågade informationen och slutligen står själva svaret.

3. METOD

Detta avsnitt är avsett till att först beskriva hur arbetet för detta projekt är uppdelat. Sedan förklaras hur hantering av virtuella bilar utförs i dagsläget och hur applikationen som skall utvecklas under detta projekt kan bidra till att denna hantering blir mer kraftfull.

3.1. Arbetets upplägg

Generering och modifiering av virtuella bilar är två av tre huvudfunktioner som skall implementeras i applikationen. För att implementera dessa funktioner är det nödvändigt att inhämta olika typer av information från CarComs databas. Således anses det vara relevant att påbörja arbetet med databashantering, samt skriva SQL-skript som kommer att användas till generering- samt modifieringsdelen.

Programmet som utvecklas under detta projekt skall ge användaren möjligheten till att ladda in och spara virtuella bilar. För detta ändamål är det väsentligt att implementera funktioner som kan hantera inläsning och skrivning av XML-filer eftersom virtuella bilar är utav detta format.

Applikationen som skall utvecklas skall låta en användare att hämta information för den virtuella bilen som laddas in i programmet. En virtuell bil som enbart består av adresser, förfrågningar och svar har ingen koppling till databasen. Således är det nödvändigt att implementera funktioner som kan matcha en virtuell bil mot ett projekt, så att information om den virtuella bilen kan inhämtas.

När ovanstående arbete är avklarat skall ett enkelt grafiskt gränssnitt byggas så att de tre huvudfunktionerna (sammanfoga, generera och modifiera virtuella bilar) som skall finnas med i applikationen kan testas under utvecklingsfasen.

Avslutningsvis skall det grafiska gränssnittet förbättras så att den utvecklade applikationen blir mer användarvänlig. Med användarvänlig i detta fall menas att det ska vara smidigt för en användare att navigera mellan de olika delarna i programmet.

3.2. Hantering av virtuella bilar

Hantering av virtuella bilar i dagsläget förutsätter att man behöver handskas mycket med XML-filer. Detta kan vara omständligt och tidskrävande, särskilt om XML-filen består av många styrenheter som i sin tur har mängder med förfrågningar. Detta avsnitt är avsett till att förklara hur virtuella bilar hanteras idag och hur denna hantering kan göras mer kraftfull med den applikationen som skall utvecklas under detta projekt.

För att exempelvis modifiera en viss respons på en DID så behöver man först söka igenom XML-filen tills rätt DID har hittats, och sedan ändra responsen för hand. Detta är både omständligt och tidskrävande, speciellt om svaret på en DID består av flera hundra eller till och med tusentals bitar. Det är ännu mer omständligt om enbart en del av DID:ens svar skall ändras eftersom man då behöver söka igenom svaret på DID:en för att hitta till rätt position. Det kan t.ex. vara så att svaret består av 100 tecken och man måste söka sig till position 72, och sedan ändra svaret i den positionen. Denna metod till att modifiera virtuella bilar blir mer och mer omständligt i samband med att antalet svar som skall modifieras ökar.

Funktionen som skall implementeras för att modifiera en virtuell bil kommer att underlätta arbetet som är beskriven ovan. För att modifiera respons på en DID behöver användaren bara ladda in en XML-fil och välja de DID:ar vars respons skall modifieras. Då kommer programmet att hämta responsen som finns för de valda DID:arna och låta användaren att modifiera svaret. En användare kommer även att kunna välja parametrar för en DID och justera parametervärden. De valda parametrarna motsvarar en del av DID:ens svar och detta innebär att man inte längre behöver slösa tid på att söka igenom svaret för en viss DID i en XML-fil för att hitta till rätt position.

För att sammanfoga två virtuella bilar behöver man även här söka igenom XML-filer, kopiera utvalda styrenheter med förfrågningar från dessa filer och sedan klistra in det i den nya filen. Med programmet som skall utvecklas kommer en användare ha möjligheten att bara ladda in XML-filer, välja godtyckligt antal styrenheter och förfrågningar som finns i filerna och slutligen generera en ny bil, som är en sammanfogning av innehållet från de två existerande bilarna.

För att generera en ny virtuell bil behöver man på liknande sätt som sammanfogningen söka igenom en XML-fil och välja ut vilka delar som skall finnas med i den nya bilen, eller fylla i innehållet för den nya bilen manuellt.

Med applikationen som skall utvecklas kommer en ny virtuell bil att kunna genereras på olika sätt. En användare kommer ha möjligheten att antingen välja ett projekt eller en default car config-bil. När någondera är vald så kommer programmet att hämta alla ECU:er, DTC:er, DID:ar och kontrollrutiner som finns i det specificerade projektet eller default car config-bilen. Därifrån kommer användaren att kunna välja de enheter som skall läggas till i den nya bilen. Det kommer också vara möjligt för en användare att generera en ny virtuell bil genom att ladda in en befintlig virtuell bil till programmet och sedan modifiera innehållet i den.

4. IMPLEMENTERING

Detta kapitel är avsett till att först beskriva samhörigheten mellan det applikationens funktioner och det grafiska gränssnittet, databashantering och hantering av virtuella bilar. Sedan beskrivs implementation av de funktioner som finns i applikationen.

4.1. Samhörighet mellan GUI och funktioner

Detta avsnitt skall förklara samhörigheten mellan det grafiska gränssnittet och de funktioner som har implementerats i applikationen.

Om en användare laddar in en virtuell bil behöver programmet identifiera innehållet i den virtuella bilen. Om användaren däremot specificerar ett projekt eller default car config-bil behöver programmet hämta information om projektet eller default car config-bilen från databasen. När innehållet i bilen har identifierats eller information från databasen har inhämtats visas detta upp i applikationen.

Den första lösningen som implementerades för att visa alla styrenheter i en virtuell bil, projekt eller default car config-bil förutsatte inte att de ECU:er som visas i applikationen skulle vara associerad med motsvarande ECU i den virtuella bilen, projektet eller default car config-bilen. Med denna lösning kunde programmet alltså inte direkt identifiera de DID:ar, DTC:er, kontrollrutiner och förfrågningar som tillhörde den ECU:n som har blivit vald av en användare. För att kunna identifiera innehållet var programmet tvunget att först läsa av adressen eller diagnosnumret för den valda ECU:n och sedan anropa andra funktioner som med hjälp av det avlästa värdet kunde identifiera innehållet.

Om en användare exempelvis skall generera en virtuell bil med en uppsättning av DID:ar i ett specificerat projekt så skall även ECU:er för de valda DID:arna läggas till i bilen. Med den första lösningen kunde programmet inte direkt identifiera vilken ECU som de valda DID:arna tillhörde. För att lägga till DID:ar till den nya bilen skapade programmet först en ECU och sedan lades de valda DID:arna till DID-listan för ECU:n.

Den nya lösningen blev att associera varje ECU som visas i applikationen med dess motsvarande ECU i den virtuella bilen, projektet eller default car config-bilen beroende på vad användaren har valt. Följaktligen kan programmet direkt identifiera innehållet för den ECU:n som väljs av en användare och visa upp det direkt i applikationen. Fördelen med denna lösning jämfört med den första är att programmet inte längre behöver utföra onödigt

arbete såsom genomsökning av innehållet i den virtuella bilen, valda projektet eller default car config-bilen för att kunna identifiera information.

4.2. Databashantering

Detta avsnitt förklarar hur databasen hanteras samt varför det är relevant med databashantering för detta projekt.

För att kunna generera och modifiera en virtuell bil kräver programmet mycket information om framförallt olika bilprojekt, styrenheter och dylikt. Informationen hämtas från CarComs databas och anropen görs antingen med SQL-skript som vi har skrivit för specifika ändamål, eller färdigskrivna procedurer som redan finns tillgängliga på databasen.

Exempel på ett specifikt SQL-skript är den som hämtar en bils default car config, det vill säga med specificerad bilmodell, årtal, motor och växellåda så hämtas alla tillhörande styrenheter för denna konfiguration. Exempel på en färdig procedur som vi använder är den som hämtar styrenheter som matchar ett visst bilprojekt, t.ex. SPA-plattformen. Man får dock alla olika styrenheter för t.ex. alla motorer som finns för den plattformen när man egentligen bara behöver en av dessa för den virtuella bilen.

Att hämta en bils default car config var inte enkelt. Om en användare inte är intresserad av att välja vilket årtal, motor eller växellåda en bilmodell ska ha så bör ändå motor- och växellådsstyrenheter visas. Detta löstes med att programmet själv alltid väljer de första alternativen inom årtal, motor och växellåda, såvida någon av dessa inte har specificerats. Detta skript skickar en stor SQL-förfrågan men detta är inget problem eftersom skriptet enbart körs när användaren väljer sin default car config-bil.

Användaren har möjligheten att välja om databashanteringen ska försöka köra så mycket procedurer den kan eller om den enbart ska köra sparade/egentillverkade SQL-skript. Procedurerna är avsevärt snabbare i förhållande till skripten men samtidigt så kräver dessa att en användare har en viss access till sitt användarlogin, något som SQL-skripten inte kräver i samma utsträckning.

4.3. Hantering av virtuella bilar

Detta avsnitt är avsett till att förklara hur en virtuell bil hanteras i programmet, mer specifikt hur en virtuell bil är uppbyggd, hur den läses in och sparas samt hur den matchas mot ett projekt i databasen.

När en XML-fil laddas in i programmet anropas funktionen som hanterar inläsning av XML-filer. Den virtuella bilen i programmet har en lista för att spara alla ECU:er som finns i XML-filen och varje ECU i sin tur har en lista för att spara dess tillhörande förfrågningar. Funktionen som läser in virtuella bilar kommer att söka igenom hela XML-filen som laddas in och lägga till alla ECU:er med dess tillhörande förfrågningar till den virtuella bilen i programmet. Denna funktion tar hänsyn till att det inte lagras dubbletter av ECU:er eller förfrågningar i den virtuella bilen i programmet.

För att kunna använda vissa funktioner såsom modifiera en virtuell bil behöver användaren olika typer av information från den virtuella bilen som laddas in i programmet. Exempel på sådan information kan vara status på en viss DTC, min- och maxvärde för parametrar samt vilka alternativ som finns för en viss parameter. För att identifiera sådan information behöver den virtuella bilen matchas mot ett projekt i databasen. Resterande del av detta avsnitt kommer att beskriva hur denna matchning sker.

I den virtuella bilen identifieras en ECU av en fysisk adress men för ett projekt däremot identifieras ECU:n av ett diagnosnummer. Således behöver varje ECU-adress i den virtuella bilen matchas mot dess tillhörande diagnosnummer. Det diagnosnummer som en ECU har kan identifieras av innehållet i DID:en ”EDA0”, förutsatt att en ECU i den virtuella bilen innehåller denna DID.

För att identifiera vilka DID:ar, DTC:er och kontrollrutiner som finns inom en viss ECU behöver programmet först identifiera vilken tjänst som har använts för förfrågningarna som finns i respektive ECU. Om den använda tjänsten för en förfrågan är 22, dvs. förfrågningen i den virtuella bilen inleds med “22” innebär det att en DID har efterfrågats. Programmet kan därmed identifiera den efterfrågade DID:en genom att läsa av de fyra tecknen som kommer efter tjänsten i förfrågningen.

Om tjänsten är 31 01, 31 02 eller 31 03 i förfrågningen indikerar detta att en kontrollrutin har efterfrågats. Programmet identifierar den efterfrågade kontrollrutinen genom att läsa av de tecknen som kommer efter tjänsten i förfrågningen.

Det kan dock vara lite mer omständligt att identifiera en DTC eftersom det inte alltid räcker med att enbart titta på förfrågningen. Om den använda tjänsten är 1902 kommer svaret för denna förfrågan att bland annat innehålla identifikationen på DTC:er vars status matchar den efterfrågade statusen. Programmet behöver därmed läsa av en viss position i responsen för att identifiera de DTC:er som finns för en viss ECU. Om tjänsten som används i förfrågningen är 1903 kommer alla DTC:er som använder tjänst 1904 att finnas med i responsen. Således behöver programmet även för denna tjänst läsa av responsen för att kunna identifiera DTC:er. Om tjänsten som används i förfrågningen är 1904 så innebär det att en DTC har efterfrågats för dess snapshots-DIDs. I detta fall behöver programmet inte identifiera DTC:n utifrån innehållet i responsen utan den efterfrågade DTC:n identifieras av de sex tecknen som kommer efter tjänsten i förfrågningen. Om tjänsten som används i förfrågningen är 1906 efterfrågas en specifik DTC:s extended data och den efterfrågade DTC:n kan identifieras av de sex tecknen som kommer efter tjänsten.

4.3.1. Generering

I detta avsnitt beskrivs hur en användare kan generera en virtuell bil eller modifiera innehållet i en befintlig virtuell bil. Detta avsnitt är även avsett till att beskriva implementering av de funktioner som möjliggör dessa tjänster.

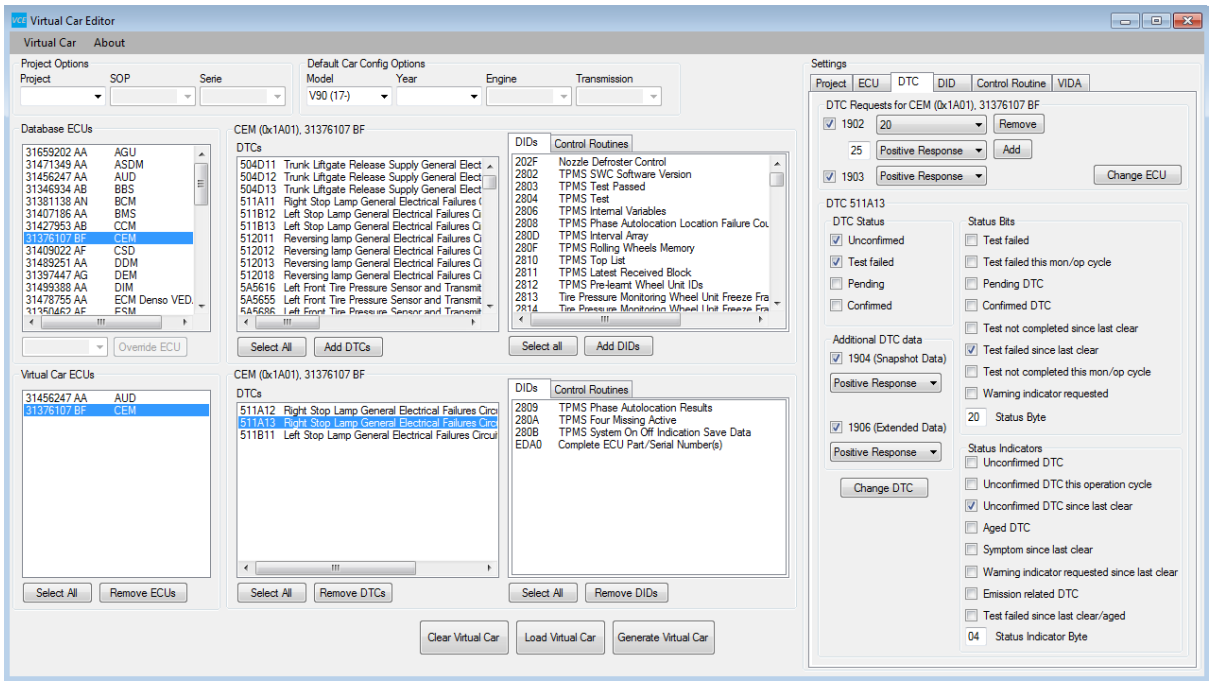


Bild 3: Bild som illustrerar genererings-vyn.

Generering av en helt ny virtuell bil kan göras på två olika sätt, antingen genom att välja ett projekt eller välja en default car config-bil. Vi börjar med att beskriva hur en virtuell bil kan genereras utifrån ett valt projekt.

När användaren trycker på "project" så laddas en lista över bilprojekt in från databasen, och när ett projekt har valts så laddas dess tillhörande SOP in från databasen. Utifrån valt projekt samt SOP hämtas serie-listan och när även detta är valt så hämtas alla ECU:er för den valda specifikationen. Denna applikation är främst gjord för att fungera med SPA-bilar och projektnamnen för dessa bilar brukar oftast bestå av fyra tecken. Exempel på ett sådant projektnamn är "517A" där 5:an indikerar på att bilen är utav SPA-plattformen, 17 indikerar på produktionsår 2017 och A indikerar på vilken vecka i produktionsåret som bilen börjar tillverkas.

Om användaren istället väljer att få ECU:er utifrån default car config-funktionen så räcker det att denne väljer modell. Eftersom årtal, motortyp och växellåda inte är valda så väljer programmet automatiskt tillhörande ECU:er som egentligen skulle kräva att dessa var valda åt användaren. Användaren kan givetvis välja årtal, motortyp eller växellåda också om det önskas men det kan i vissa fall saknas ECU:er ifall något av alternativen saknar tillhörande ECU. Modell-listan i default car config är väldigt lång då den även innehåller alla äldre bilmodeller. Applikationen tillåter användaren att maska bort alla icke-SPA genom att använda en sparad lista på modeller som inte skall visas. Det är dessutom möjligt att sortera bort ECU:er som har namn som "PBL" och "SBL" då dessa inte brukar användas.

När ECU:erna är hämtade (kan ses i bild 3 under "Database ECUs") så kan användaren välja en ECU och när detta gjorts så hämtas ECU:ns DTC:er, DID:s, kontrollrutiner samt annan nödvändig information såsom stöd för tjänst 10 01, 10 02 etcetera från databasen. Användaren får nu möjlighet till att ändra om tjänst 10 01, 10 02, 10 03 samt 11 01 skall finnas i bilen och kan fritt välja vilka DTCs, DIDs samt kontrollrutiner som ska läggas in till den genererade bilen.

Användaren kan även välja att ändra vald ECU till någon annan av samma typ genom att köra "override". Till exempel, om ECU:n "SRS" är vald så fås en lista över alla ECU:er med namnet SRS fast med andra diagnosnummer som användaren kan välja mellan för att byta ut original ECU:n. Om den valda ECU:n byts ut så hämtas information för den nya ECU:n precis som för den tidigare.

När en användare väljer en eller flera DTC:er så kan följande utföras:

Tjänst 19 02: Välja att lägga till eller ta bort olika "hex"-masker för ECU:n

Tjänst 19 03: Välja om tjänst 19 03 skall användas eller inte för ECU:n

Tjänst 19 04: Välja om snapshot skall användas och i så fall laddas snapshot-data in från databasen när DTC:n har lagts till i den genererade bilen.

Tjänst 19 06: Välja om extended data skall användas och i så fall så kommer "status indicator" användas här i. Likt 19 04 så är även denna tjänst DTC-specifik och inte ECU-specifik som tjänst 19 02 och 19 03.

För samtliga ovanstående tjänster kan användaren välja om det ska vara NRC-respons eller inte. Dessutom så kan status byte och status indicator ändras på ett smidigt sätt för vald/valda

DTC:er. Om användaren kryssar i något alternativ under “DTC Status” så ändrar den automatiskt motsvarigheten i status byte och status indicator. Kryssar man i något alternativ under de sistnämnda grupperna så ändras kryssrutorna under DTC Status och de tillhörande textrutor ändras också automatiskt efter detta. Samma princip gäller om man ändrar textrutorna direkt.

När det gäller DID:ar så kan en användare lägga till ett godtyckligt antal sådana till den nya bilen samt välja om det skall vara med NRC-respons eller inte. När det gäller kontrollrutiner så kan användaren välja om tjänst 31 01, 31 02, 31 03 skall vara med, samt om NRC-respons för dessa tjänster skall vara med eller inte.

Genererings-delen är den mest omfattande delen i detta examensarbete och det är den delen som har fått mest ändringar eftersom förslag på funktioner kom till denna del vid olika tillfällen från personalen på gruppen.

4.3.2. Sammanfogning

Detta avsnitt är avsett till att beskriva de funktioner som tillåter en användare att sammanfoga två virtuella bilar.

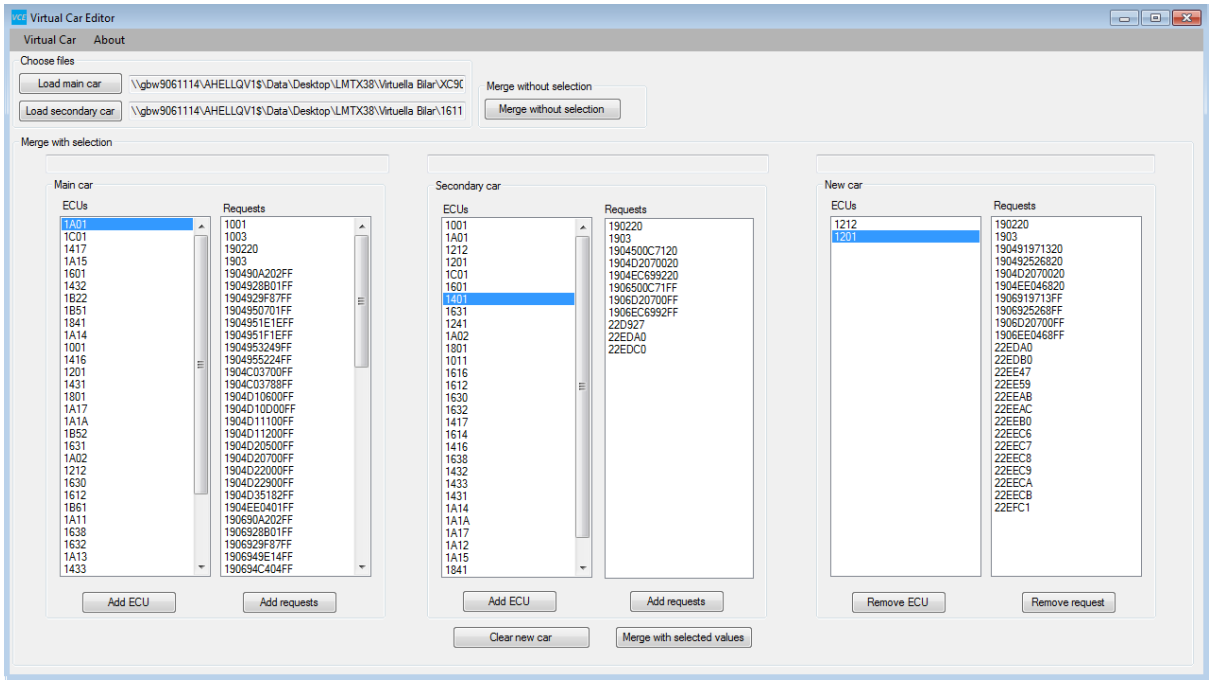


Bild 4: Gränssnitt för sammanfogning av virtuella bilar.

Bild 4 ovan visar det grafiska gränssnittet för sammanfogning av virtuella bilar. För att kunna sammanfoga virtuella bilar behöver användaren givetvis börja med att ladda in existerande XML-filer. När XML-filer laddas in i applikationen kommer programmet att hantera de på det sättet som är beskriven under "4.3. Hantering av virtuella bilar".

Applikationen erbjuder två olika funktioner för att sammanfoga virtuella bilar. Den första funktionen är "Merge without selection" och denna funktion tillåter en användare att sammanfoga två bilar så mycket som det går. Mer specifikt så kommer denna funktion att ta enheter med tillhörande förfrågningar från båda bilarna och sammanföra dessa till en ny virtuell bil. Samtidigt som knappen trycks så kommer en ruta komma upp som låter användaren välja vart han vill spara sin nya virtuella bil.

Den andra funktionen "Merge with selected values" ger användaren möjligheten att sammanfoga en virtuell bil med utvalda enheter från de bilarna som har laddats in. När en användare laddar in en virtuell bil kommer applikationen att visa upp alla ECU:er från den virtuella bilen. Därifrån kan en användare välja en ECU och då kommer applikationen att visa upp alla förfrågningar som finns i ECU:n. Utifrån de förfrågningar som visas i

programmet kan användaren lägga till ett godtyckligt antal sådana med dess tillhörande respons, till den bilen som skall sammanfogas. En användare kan även lägga till en eller flera ECU:er samtidigt till den nya bilen vilket skulle innebära att alla förfrågningar med dess tillhörande respons läggs till. Innan den nya bilen genereras har användaren möjligheten att ta bort förfrågningar, ECU:er eller rentav ta bort allt som har lagts till i den nya bilen.

4.3.3. Modifiering

I detta avsnitt beskrivs hur en användare kan modifiera en virtuell bil samt implementering av de funktioner som möjliggör denna tjänst.

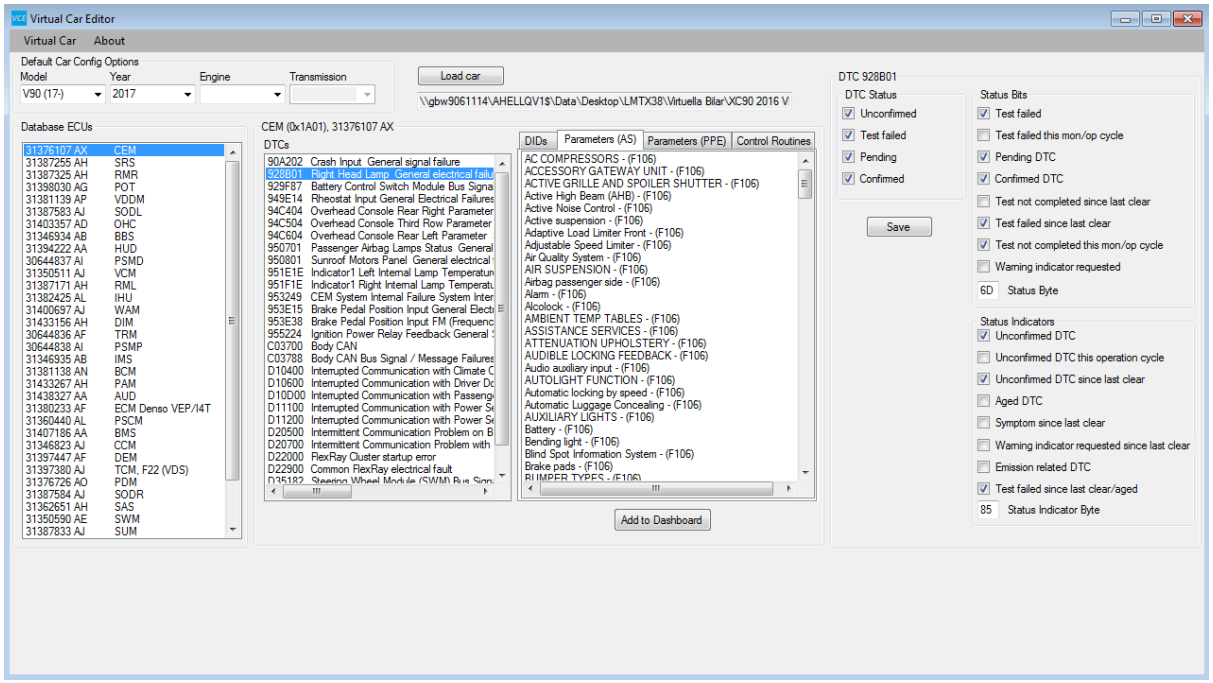


Bild 5: Första vyn för modifierings-delen.

Bild 5 ovan visar den första sidan för modifiering av virtuella bilar. Modifiering av virtuella bilar låter en användare att ändra respons för parametrar inom en DID eller kontrollrutin, justera värdet på de olika alternativen som finns för en viss parameter och ändra status på DTC:er. En virtuell bil kan dock inte identifiera information såsom parametervärden, vilka alternativ som finns för en viss parameter och vilka värden som finns i dessa alternativ. Således behöver den virtuella bilen matchas till ett projekt så att all nödvändig information som behövs för att modifiera bilen kan hämtas från CarComs databas. Hur en virtuell bil hanteras av programmet när den har laddats in och hur den matchas mot ett projekt har förklarats i avsnitt ”4.3. Hantering av virtuella bilar”.

När den virtuella bilen har gjorts om till ett projekt kommer alla ECU:er att listas upp med dess diagnosnummer (t.ex. 31376107 AX) och förkortning (t.ex. CEM). En användare kan därmed välja en ECU och då kommer programmet att anropa olika SQL-skript för att hämta all nödvändig information för den ECU:n. Ett av de skripten används för att hämta alla DID:ar som finns i den virtuella bilen för den valda ECU:n, ett annat skript för inhämtning av

DTC:er i den virtuella bilen, ett tredje skript för att hämta kontrollrutinerna i bilen och ett fjärde för inhämtning av alla parametrar som finns i de hämtade DID:arna.

För att hämta alla parametrar och dess alternativ använde vi till en början ett enda SQL-skript. Detta resulterade dock i en stor fördröjning när programmet skulle hämta parametrar med många alternativ. För att minska denna fördröjning delades skriptet upp i två delar. Det första skriptet anropas i samband med att alla DID:ar för en vald ECU har blivit hämtade och detta skript skickar förfrågan till databasen om alla parametrar som finns i de hämtade DID:arna. Det andra skriptet anropas när en användare har valt ett visst antal parametrar att modifiera och skriptet hämtar alla alternativ som finns för de valda parametrarna.

När alla DID:ar, DTC:er, och kontrollrutiner som finns i den virtuella bilen för den valda ECU:n har hämtats från databasen kommer dessa att visas upp i det grafiska gränssnittet. Därifrån kan användaren modifiera en DTC genom att t.ex. ändra dess status. Dessutom kommer användaren att kunna välja ett godtyckligt antal parametrar från DID:ar eller kontrollrutiner som visas i applikationen. Alla valda parametrar kommer att visas upp i en ny ruta och utseendet för denna ruta visas i bilden nedan:

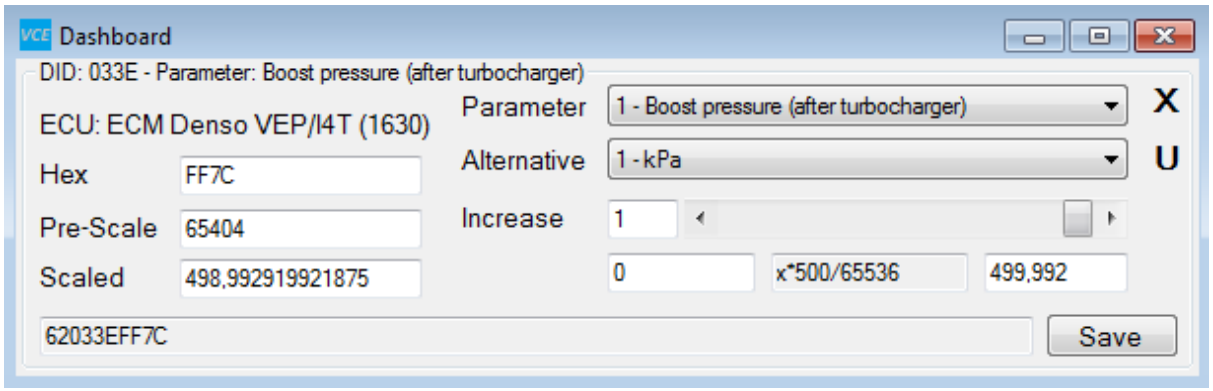


Bild 6: Vyn när en användare har valt att modifiera DID eller parametrar.

I bilden ovan har parametern "Boost pressure (after turbocharger") blivit vald av användaren och i bilden framgår det även att denna parameter tillhör DID "033E" som finns i ECU "ECM Denso VEP/l4T" med adressen "1630".

Varje parameter har ett eller flera alternativ och ett av de alternativen visas av programmet när användaren väljer att modifiera en parameter. Huruvida ett alternativ är valt för en parameter eller inte kan avgöras utifrån responsen för den DID som den valda parametern tillhör. Den delen av DID:ens svar som motsvarar värdet för parametern räknas ut med hjälp av parameterns offset och bitlängd. Det värdet som finns i denna del jämförs sedan med de

värdena som alternativen för en parameter har. Om värdet för parametern är lika med värdet för ett av parameterns alternativ innebär att det alternativet blir vald för parametern. Om inget alternativ motsvarar den valda parameterns värde väljer programmet det sista alternativet som finns för parametern. När användaren justerar värdet på en parameter kommer den del av DID:ens respons som motsvarar parametern också att ändras.

Modifiering av en parameter kan göras på ett smidigt sätt med den scrollbaren som har implementerats. Om användaren vill öka eller minska värdet för varje tryck med ett visst antal steg kan detta anges i textfältet som står till vänster om scrollbaren. Under scrollbaren finns det tre textfält där den första och sista visar min- respektive max-värde för parametern medan fältet i mitten visar ekvationen för hur parametervärdet skall skalas.

Förutom scrollbaren kan en användare även modifiera en parameter genom att ange ett värde i textrutan för "Hex", "Pre-scale" eller "Scaled". Om användaren ändrar hex-värdet så ändras även värdet för "Pre-scale". Samma sak för om "Pre-scale" ändras fast då ändras "Hex" och "Scaled". Ändras "Scaled" vilket är värdet som är beräknat med ekvationen så körs en algoritm som tar fram inversen för denna om det är en förstagradsekvation, således kan "Pre-scale" ändras.

Om en användare trycker på listan för parametrar kommer alla parametrar inom samma DID att visas. Därmed kan användaren välja en annan parameter och modifiera dess värde. När användaren har modifierat klart en parameter kan denne trycka på save och då kommer det ändrade värdet att sparas i den virtuella bilen som har laddats in.

5. RESULTAT

Detta avsnitt kommer att beskriva de uppnådda resultaten i detta projekt.

Resultatet av detta arbete är en utvecklad applikation som innehåller tre huvudfunktioner. Den första och mest omfattande huvudfunktionen som har implementerats i applikationen är avsedd till att generera virtuella bilar vilket beskrivs nedan.

För att generera en virtuell bil behöver en användare först specificera ett projekt eller default car config-bil. När detta har gjorts anropar programmet olika SQL-skript för att hämta all nödvändig information såsom ECU:er, DID:ar, DTC:er och kontrollrutiner för det specificerade projektet eller default car config-bilen. Därifrån kan en användare välja ett godtyckligt antal enheter, ta bort valda enheter, välja olika alternativ för tjänster t.ex. huruvida NRC-respons skall vara med eller inte, välja att ersätta en ECU med en annan som har samma namn fast olika diagnosnummer, lägga till stöd för VIDA och slutligen generera en virtuell bil utifrån de valen som har gjorts.

En virtuell bil kan även genereras genom att ladda in en befintlig virtuell bil till programmet. När den befintliga bilen har laddats kommer en användare att kunna modifiera innehållet i bilen, lägga till samt ta bort enheter från den befintliga bilen och generera en ny virtuell bil utifrån detta.

Den andra huvudfunktionen som har implementerats i applikationen låter en användare att sammanfoga två virtuella bilar och det finns två alternativ för denna funktion. Det första alternativet låter en användare att sammanfoga två bilar genom att ta alla enheter med dess tillhörande förfrågningar från båda bilarna och sammanföra dessa till en ny virtuell. Detta alternativ tar hänsyn till att de enheter och förfrågningar som läggs in till den nya bilen är unika. Det andra alternativet ger användaren möjligheten att själv välja vilka enheter och förfrågningar från de befintliga bilarna som skall finnas med i den sammanfogade bilen.

Den tredje huvudfunktionen som har implementerats i applikationen hanterar modifiering av virtuella bilar. När en virtuell bil har laddats in i programmet kan en användare modifiera den enligt följande:

- Välja en DTC och ändra dess statusar.
- Välja en DID eller kontrollrutin och modifiera dess parametrar.
- Välja ett godtyckligt antal parametrar för olika DID:ar och modifiera dess värde på ett smidigt sätt.

5.1. Applikationens påverkan

För att göra en bedömning av resultatet kan det konstateras att den utvecklade applikationen har bidragit till en mer effektiv metod för hantering av virtuella bilar. Med de funktionerna som är implementerade i applikationen behöver man inte längre hantera virtuella bilar genom att handskas med XML-filer i samma utsträckning som tidigare.

Om man exempelvis skall generera en ny virtuell bil eller sammanfoga virtuella bilar med applikationen så behöver man inte söka igenom en XML-fil för hand och välja de styrenheter, DID:ar, DTC:er och kontrollrutiner som skall finnas med i den nya bilen. Det enda som krävs för att generera en virtuell bil med applikationen är att användaren specificerar ett projekt, default car config-bil eller laddar in en befintlig bil. När detta har gjorts kommer programmet att identifiera all information som är nödvändigt för att generera en ny virtuell bil. För sammanfogning av virtuella bilar behöver användaren enbart ladda in existerande virtuella bilar och då kommer programmet att visa upp alla enheter med tillhörande förfrågningar. Därifrån kan användaren själv välja vad som skall ingå i den sammanfogade bilen.

Om man skulle vilja ändra på en parameter för en viss DID utan vår applikation skulle man först behöva ta reda på offset och bitlängd för parametern. När detta är gjort skulle man vara tvungen att räkna ut den positionen inom DID:ens respons som motsvarar parametervärdet. Sedan skulle man behöva söka sig fram till den positionen och slutligen modifiera värdet. Detta tillvägagångssätt är väldigt opraktiskt, speciellt om man skall ändra responsen för flera DID:ar och därmed behöver repetera dessa steg för varje modifiering. Med den applikation som vi har utvecklat räcker det med att en användare väljer en parameter. Då kommer programmet att räkna ut den del av DID:ens respons som motsvarar den valda parametern och ändra värdet i responsen enligt det som användaren har specificerat.

6. DISKUSSION OCH SLUTSATSER

I detta avsnitt diskuteras de slutsatser som kan dras från de uppnådda resultaten och detta görs genom att svara på de två sista frågorna som ställdes under avsnittet "1.3. Precisering av frågeställningar".

I samband med en mer kraftfull hantering av virtuella bilar kommer den utvecklade applikationen att ha en stor betydelse för gruppen Diagnostics & dependability. Enligt vår handledare på gruppen[6] kommer applikationen att minska på trycket med avseende på tillgång till riktiga provbilar, särskilt i tidiga faser i projekten. Med några enkla knapptryckningar kommer man att kunna generera användbara virtuella bilar längs projektets gång. Detta ger de i gruppen en bra möjlighet att komma igång med utveckling samt verifiering av skript, tidigare än vad som är möjligt utan applikationen.

Applikationen bidrar till att de i gruppen kan generera bilar med olika slags specifikationer, växla mellan utrustningsnivåer och på så vis verifiera logik i de skripten som utvecklas. Gruppen kommer med lätthet kunna försätta en virtuell bil i samma läge som för en specifik kundbil i verkstad, exempelvis genom att sätta samma statusar på felkoder. Därmed kan de "simulera" det som mekanikern upplever i verkstaden med dennes kundbil.

Med applikationen kan man välja att låta bilen svara med negativ respons istället för med positiv respons. Detta medför att gruppen kan utveckla robusta skript med bra felhantering, vilket höjer kvaliteten på deras leveranser.

Gruppen kommer på ett bättre sätt kunna verifiera stora skript där flera parametrars värden påverkar beteendet i skriptet. Genom att enkelt kunna välja ut ett antal parametrar och få in dem i en överskådlig vy för att kunna justera dem, ökar möjligheterna att utföra verifieringen virtuellt även för relativt komplexa funktioner som de i gruppen annars hade fått pröva mot en provbil.

Sammanfattningsvis kan man säga att om gruppen lär sig använda applikationen på rätt sätt så kommer de att spara väldigt många timmar i förhållande till om motsvarande arbete hade utförts utan applikationen.

[6] Fredrik Fridmar, metodingenjör, Volvo Cars, 2017, demonstration av applikationen, 31 maj

Förutom gruppen Diagnostics & dependability kommer applikationen även att vara användbar för andra aktörer. Workshop Information Electrical som är gruppens redaktörer, ansvarar för att språkgranska gruppens leveranser. Redaktörerna kommer med applikationen ha möjligheten till att enkelt anpassa en bils specifikation, utrustningsnivå, sätta värden på parametrar samt felkoder och på så sätt uppfylla villkor för att ta vägval till skriptets alla delar. Applikationen bidrar därmed till att den språkgranskningen som görs av redaktörerna kan utföras på ett mer effektivt och snabbare sätt.

Diagnoskonstruktörer som oftast är intresserade av att veta vilken åtgärd som ska tas vid en viss felkod eller kundsymptom som lett mekanikern, kommer också att ha nytta av applikationen. Genom att använda applikationen och försätta bilen i önskat läge kan flödet av felsökning kontrolleras. Det blir mer kraftfullt jämfört med dagens metod där andra interna system behöver användas för att försöka få en överblick över hur felsökningsflödet kommer att se ut. Med applikationen kan flödet tydligt visualiseras i den slutgiltiga eftermarknadsapplikationen VIDA.

CMQ (Current Model Quality) och övriga inom kvalitetsorganisationen kan också ha nytta av applikationen. De har egentligen samma behov som diagnoskonstruktörerna för att förstå problem och beteenden som uppstår i verkstan när mekanikern ställs inför att felsöka en bil som lämnar viss typ av data ifrån sig.

Customer Service samt deras applikationsutvecklare av SPASE och VIDA kan ha nytta av applikationen för att skapa och modifiera bilar med innehåll som gör att applikationernas olika delar kan verifieras.

6.1. Hållbar utveckling

Arbetsmiljön för de i Diagnostics & dependability som använder applikationen har blivit bättre eftersom de mer sällan kommer behöva boka fysiska provbilar för att utföra vissa uppgifter. Ur ett rent miljöperspektiv så sparas dessutom bränsle/el-kostnader eftersom vissa testskript skall pröva långsamma förändringar i bilen, vilket nu i många fall kan göras vid skrivbordet istället med applikationen. Applikationen i sin helhet bidrar till att såväl tid som resurser sparas för både de anställda och Volvo Cars.

7. FÖRSLAG TILL FORTSATT ARBETE

I detta avsnitt anges förslag på förbättringar av nuvarande funktioner samt implementation av ytterligare funktioner för framtida utveckling av applikationen.

För genererings-delen så skulle det kunna skapas ett verktyg för att välja bilmodell, motor etc. för VIN-numret så att användaren enbart behöver skriva in chassinummer istället för hela VIN-numret. Nackdelen med nuvarande metod är alltså att användaren redan måste ha färdiga VIN-nummer och veta tillhörande bilmodell, årtal, motor, växellåda för dessa nummer. Det skulle även kunna göras så att genererings-delen kan hantera mer ovanliga tjänster. Detta skulle dock enbart hjälpa till i fall där man använder dessa tjänster vilka är nästintill obefintliga.

Efter applikationens demonstration på Volvo så föreslogs ytterligare funktioner som skulle kunna underlätta arbetsmiljön i genererings-delen. En av dessa är att användaren bör kunna välja flera ECU:er i den skapade bilen och se samtliga DID:ar, DTC:er och kontrollrutiner för dessa. Ett annat förslag är att användaren bör kunna få möjligheten till att ändra tillbaka till original-ECU:n om denna blivit ersatt via override-funktionen.

Angående modifierings-delen så kom det förslag såsom att användaren bör kunna ändra respons för DID-svaren i DTC:ers snapshot-data samt ändra respons för räknare-värden i DTC:ers extended data. Modifierings-delen saknar stöd för hantering av NRCs, det finns dock stöd för detta i genererings-delen men det kan vara omständligt att behöva byta mellan dessa två vyer även om denna funktion är något som inte kommer att användas ofta. Det kom även förslag om att det bör finnas en knapp för att spara allting som finns i dashboard-fönstret på samma gång och att användaren även borde få möjligheten till att låta programmet kontinuerligt spara vid ett givet tidsintervall.

Rent allmänt för applikationen så skulle det kunna skapas någon sorts förloppsindikator som visar hur många procent som är klar i fall där inläsning från databasen tar väldigt lång tid. Avslutningsvis så skulle man kunna göra om alla egenskapade SQL-skript till procedurer i databasen vilket skulle göra anropen betydligt kortare och därmed snabbare.

Referenser

[1] Volvo Cars, 2017. VOLVO CARS IN BRIEF. Volvo Cars. http://assets.volvocars.com/se/~/media/shared-assets/downloads/this-is-volvo/volvo-in-brief2016.pdf (Hämtad 2017-05-20).

[2] Softing, 2016. Unified Diagnostic Services - ISO 14229, https://automotive.softing.com/fileadmin/sof-files/pdf/de/ae/poster/UDS_Faltposter_softing2016.pdf (Hämtad 2017-05-30).