Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Eclipse környezetbe integrált járműbusz vizualizáció

Diplomaterv

Készítette
Faragó Dániel

Belső konzulens
Dr. Sujbert László
BME MIT

Külső konzulens
Dr. Pintér Gergely
ThyssenKrupp Presta Hungary Kft.

Budapest, 2014

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Faragó Dániel*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2014. december 18.

Faragó Dániel
hallgató

Kivonat

Napjaink modern járműveinek összetett funkcióit egymással kapcsolatban álló elektronikus vezérlőegységek (ECU) valósítják meg. Az így létrejövő bonyolult elosztott rendszer tervezése és tesztelése nehéz mérnöki feladat. Az autóiparban a komponensek kimerítő tesztelése elengedhetetlen a fejlesztés korai fázisától kezdve.

Az AUTOSAR szabvány által definiált modellezési nyelv lehetővé teszi autóipari rendszerek működésének és a rendszerben zajló kommunikációnak szabványos leírását. Ezen modellek elegendő információt tartalmaznak egy a vezérlőegységek között zajló kommunikáció megfigyelésére alkalmas eszköz konfigurációjának előállítására.

A vezérlőegységek működésének tesztelése során megkerülhetetlen azok kommunikációs jeleinek megfigyelése. Erre a célra a piacon számos eszköz és hozzájuk kapcsolódó PC oldali szoftver érhető el. Ezen megoldások, hátránya, hogy noha gazdag vizualizációs és gyakran szimulációs eszközkészlettel rendelkeznek, a hálózaton utazó keretek szerkezetét (vagyis az általuk hordozott, a felhasználó számára jelentéssel bíró adatok formátumát) gyártó és/vagy hálózatspecifikus leírásmódban várják (CAN DB, Fibex). Kézenfekvőnek tűnik egy olyan megoldást fejleszteni, amely az ECU-k kommunikációjának leírására használt AUTOSAR modellt tudja referenciaként használni, köztes leírásmódok kikényszerítése nélkül.

A megvalósított Trace szoftver lehetőséget biztosít AUTOSAR alapú vezérlőegységek kommunikációjának megfigyelésére akár üzenetek, PDU-k vagy szignálok szintjén. A kommunikáció megjeleníthető táblázatos és grafikus formában, valamint a kiválasztott szignálokhoz külön indikátorok is rendelhetők. A megfigyelési napló igény esetén CSV fájlba menthető az adatok későbbi feldolgozására. A megjelenítő képes a szignálok értékének alapján különböző statisztikai paraméterek számítására is.

Az eszköz a megjelenítendő üzenetek, PDU-k, szignálok valamint a kommunikációs kapcsolatot biztosító csatornák paramétereinek konfigurációját egy adat modellben tárolja. A konfigurációs modell könnyedén származtatható az AUTOSAR szabványos System Description modelljeiből. A konfigurációs folyamatban – melynek során az üzenetek és a hozzájuk kapcsolódó megjelenítési beállítások megadhatók – a felhasználót egy varázsló segíti.

A szoftver a nyílt forráskódú Eclipse keretrendszerre és hozzá kapcsolódó technológiákra épül.

Abstract

Nowadays the complex functionalities of modern vehicles are realized by electronic control units (ECU) connected with each other via different vehicle buses. The resulting distributed system is hard to be designed and tested. Therefore in the automotive industry the exhaustive testing of components is indispensable from the early phase of the development process.

The AUTOSAR standard offers the methodology of describing a whole system, and the communication between control units. This description is based on standardized data models, and contains all information for the configuration of tools observing communication over the vehicle bus.

To test the proper operation of an ECU, the observation of the related communication signals is a basic requirement. For this purpose different communication devices and PC side software are available on the market. The disatvantages of these solutions – although they provide rich visualization and often simulation services – are that the structure of the frames traveling on the network (the format of the user relevant data delivered by them) needs to be described in a manufacturer and/or protocol specific way (like CAN DB or Fibex). It seems obvious to develop a solution that can use AUTOSAR models as a description of inter-ECU communication, without enforce using intermediate description methods.

The implemented Trace software offers a convenient way to visualize the frames, PDUs and signals participating in the communication between AUTOSAR based control units. The communication can be observed in a table and also in a graph view and selected signals can be displayed on custom indicators. The log recorded by the Trace can be saved into CSV files to support further processing. Statistical parameters can be also calculated and displayed by the tool.

The configuration of the implemented software includes frames, PDUs and signals. Parameters of communication channels to be used to connect to the vehicle bus network are also stored. The entire configuration description described by a data model. This model can be easily derived from AUTOSAR System Description models. The configuration process is supported by a wizard which helps the user to choose the required frames and their display settings.

The software is based on the open source Eclipse framework and related technologies.

Bevezető

Napjaink autóipara rohamosan fejlődik. A biztonságra valamint környezetvédelemre irányuló kormányzati előírások teljesítése a fogyasztók által elvárt komfort és teljesítmény megtartása mellett csak fejlett elektronikai megoldások alkalmazásával lehetséges. Egy mai prémiumkategóriás személyautó akár száz vezérlőegységet (ECU) is tartalmazhat. A vezérlőegységek feladataikat számos mért és számított paraméter alapján végzik. Az adatok egy részét saját maguk állítják elő, míg a másik részét egymással kommunikálva, a környezetükben elhelyezkedő többi ECU-tól kapják. Az így kialakuló bonyolult kommunikációs hálózat és elosztott rendszer tervezése, valamint megbízható működésének garantálása nem kis kihívás az autógyártók számára. A különböző ECU-k különböző beszállítóktól származnak, mégis biztosítani kell azok konzisztens együttműködését.

Diplomatervezés feladatomat a ThyssenKrupp Presta Hungary Kft.-nél végeztem. A vállalat elektronikus kormányrendszerek fejlesztésével és gyártásával foglalkozik. A fejlesztés korai fázisától kezdve elengedhetetlen a vezérlőegységet körülvevő kommunikációs környezet szimulálása (restbus szimuláció[1]) valamint a kommunikációs folyamatok megfigyelése, naplózása (trace). Erre a célra különböző kommunikációs eszközöket, átjárókat (gateway), valamint hozzájuk kapcsolódó számítógépes keretrendszereket alkalmaznak.

A piacon kapható megoldások (például a Vector eszközei) számos megjelenítési és vizualizációs funkcióval rendelkeznek, rugalmasak és jól konfigurálhatók, áruk azonban magas. A másik sokkal fontosabb szempont, hogy az üzenetek értelmezéséhez különböző gyártó- és protokollfüggő leírókat várnak (például Fibex vagy CAN DB), az AUTOSAR szabványos leírók támogatása még nem elterjedt. A vállalatnál felmerült az igény egy saját készítésű, alacsony költségű, AUTOSAR alapú megoldás létrehozására.

Az előző félévekben elkészült a hardver, valamint az MSc Önálló Laboratórium tárgy keretein belül elkészítettem az eszközön futó beágyazott (C nyelven írt) szoftvert valamint a használatához szükséges PC oldali (Java nyelven írt) drivert és API-t. Diplomafeladatom egy – a driver szolgáltatásaira támaszkodó – trace szoftver elkészítése, melynek segítségével a Gatewayen keresztül megfigyelhető és elemezhető az autóipari buszrendszereken zajló kommunikáció. Ezzel párhuzamosan készül egy restbus szimulációs felület is. A két eszköz egymást kiegészítve, egymás szolgáltatásaira támaszkodva egy teljes értékű járműkommunikáció szimulációs és vizualizációs megoldást kínál.

[1]Remaining Bus Simulation: Autóipari vezérlőegységek fejlesztése során a vezérlőegységet körülvevő többi vezérlőegység kommunikációs jeleinek szimulálása.

Megvalósítandó követelmények

A szoftver leendő felhasználóit képező csoportok vezetőivel való megbeszélések során összegyűjtöttük az eszközzel kapcsolatos elvárásokat. Bemutatták, hogy jelenleg milyen programokat és azoknak milyen szolgáltatásait használják mindennapi munkájuk során. Ezek alapján lefektettük a Diplomatervezés, valamint a későbbiek során megvalósítandó követelményeket:

- A vállalatnál a fejlesztés az autóiparban egyre széleskörűbben alkalmazott AUTOSAR szabvány alapján történik. A szabvány lehetővé teszi az autóipari rendszerek magas szintű modellezését a szoftverkomponensektől kezdve a vezérlőegységek közti kommunikációig. Az készítendő szoftvernek támogatnia kell az AUTOSAR modellek feldolgozását a konfigurációs folyamat leegyszerűsítése érdekében. A modellek kezelésére, szerkesztésére valamint az azok alapján történő kódgenerálásra egy Eclipse alapú, saját fejlesztésű eszközt, az AUTOSAR Architectet használjuk. Megoldásunknak ezen eszköz szolgáltatásait kell kiegészítenie, így az hosszútávon szoftvertervezési és -integrálási feladatokon kívül tesztelési és verifikálási feladatok ellátására is képes lehet.

- A szoftvernek az AUTOSAR kommunikáció mindhárom szintjét (szignálok, PDU-k, keretek) támogatnia kell, restbus szimulációs és trace oldalról is. A kommunikációs logot, különböző nézetekben kell tudnia megjelenítenie, arra különböző szűrőket kell tudnia alkalmaznia. Támogatnia kell kiválasztott szignálokhoz indikátorok valamint kontrollok megjelenítését, azok értékeinek futásidőben történő nyomon követésére és változtatására. A kijelölt szignálok értékét képesnek kell lennie grafikonon is megjelenítenie, valamint azokkal kapcsolatosan alapvető statisztikai paramétereket kiszámítania.

- Képesnek kell lennie a hosszabb, valamint a diagnosztikai üzenetek átvitelére alkalmazott különböző szállítási rétegbeli protokollok értelmezésére (FlexRay és CAN Transport Layer).

- A restbus szimuláció szekvenciáinak leírását egy – a Vector CAPL nyelvéhez hasonló – script nyelvvel kell támogatnia.

- Az elkészítendő eszköznek a Gateway szolgáltatásaira támaszkodva támogatnia kell mindhárom elterjedt autóipari kommunikációs protokollt (CAN, FlexRay, LIN).

Összefoglalva tehát a diplomaterv célja egy olyan desktop alkalmazás fejlesztése, amely lehetővé teszi autóipari buszokon zajló kommunikáció felhasználóbarát vizualizációját ECU-k fejlesztésének és tesztelésének támogatása céljából. A PC és a beágyazott buszok közötti híd szerepét a hardware gateway eszköz játssza, amelynek a PC oldali driverét a korábbi tanulmányaim során készítettem el. A most megvalósítandó programnak illeszkednie kell az AUTOSAR alapú fejlesztési környezetbe azáltal, hogy a kommunikáció leírását

közvetlenül a rendszer modellből (ECU Extract) kell kinyernie. A munka során a fő kihívást jelentő területek (i) a kommunikációt leíró gazdag rendszermodell megértése és feldolgozása, (ii) a vizualizációhoz szükséges PC oldali adatmodell megtervezése, (iii) a desktop vizualizációs funkciók tervezése és megvalósítása és (iv) együttműködés a gateway eszközzel.

A szoftver a fent említett funkciókra támaszkodva a jövőben kiegészülhet számos magas szintű szolgáltatással. Ilyenek például a különböző diagnosztikai funkciók (UDS), kalibrációs protokollok (CCP, XCP), valamint a hálózati menedzsment magas szintű támogatása.

Dolgozatomban a szoftver felépítését és működését a CAN protokollon keresztül mutatom be. A FlexRay valamint LIN protokollok felett való működés analóg a CAN esetében ismertetettel.

Modellvezérelt szoftverfejlesztés

A mérnöki területek napjainkra jellemző összetett problémái egyre összetettebb alkalmazások készítését teszik szükségessé. A felhasználóknak ellenben megbízható és egyszerűen testreszabható eszközökre van szükségük. A szoftverfejlesztés modellalapú megközelítése egyre népszerűbb paradigma, mivel modellek alkalmazásával a megoldandó problémák leírása az emberi gondolkodáshoz közelebb álló, magasabb absztrakciós szintre emelhető. A modell megalkotása után, a szoftver azt feldolgozva már képes a felhasználó szándékai szerinti működés megvalósítására.

Az eszközt használók munkáját megkönnyítendő, célszerű az általános modellezési nyelvek (például UML) helyett egy az adott szakterületre specializált, saját modellezési nyelvet alkotni és használni. Az így testreszabott nyelvet nevezik szakterület-specifikus modellező nyelvnek, röviden DSML-nek[1] [1].

Az AUTOSAR által definiált nyelv is egy ilyen, kifejezetten autóipari vezérlőegységek működésének modellezésére szolgáló, szakterület-specifikus nyelv. Feladatom megoldása során, az AUTOSAR modellező nyelvét használtam fel az elkészítendő szoftver konfigurációjának leírására és tárolására.

Az általam megvalósított szoftver működésének hátterében több adatmodell is áll. A felhasználó egy varázsló segítségével egy szabványos AUTOSAR rendszerleíró modellből hozza létre a szoftver működéséhez szükséges (szintén AUTOSAR nyelven megfogalmazott) adatmodellt. A szoftver indulásakor feldolgozza a hivatkozott modelleket, és annak alapján építi fel saját futásidejű objektumait.

Eclipse

Az Eclipse egy nyílt forráskódú, platformfüggetlen szoftverkeretrendszer, melyre támaszkodva hatékonyan és gyorsan fejleszthetők különböző célú kliens alkalmazások. Az Eclipse és a köré épülő szolgáltatások elsősorban integrált fejlesztőkörnyezetek (IDE) és model-

[1] Domain Specific Modeling Language

lezőeszközök fejlesztését célozzák meg, de egyre népszerűbb platformként szolgál egyéb, általános célú szoftverek fejlesztéséhez is. [2]

Az Eclipse-re épülő alkalmazások a platformra telepíthető pluginekből állnak. A komponens alapú megközelítésnek köszönhetően a funkciók egymástól függetlenül, rugalmasan fejleszthetők, és integrálhatók a környezetbe. Az általam fejlesztett eszköz is lényegében Eclipse pluginek egy halmazát jelenti, melyek egymásra, valamint az AUTOSAR Architect pluginjai által nyújtott szolgáltatásokra támaszkodva valósítják meg a kívánt működést.

Eclipse Modeling Framework

Az Eclipse Modeling Framework, vagy EMF, egy olyan Eclipse alapú keretrendszer, melynek segítségével szakterület-specifikus modellezési nyelvek és hozzájuk kapcsolódó modellező eszközök fejleszthetők. [3]

Az EMF szolgáltatásai:

- Lehetőséget biztosít a nyelv meta-modelljének definiálására (Ecore).
- A definiált meta-modell alapján képes a modellt futásidőben leíró Java osztályok generálására.
- Képes a modell szerkesztésére szolgáló, egyszerű editor plugin generálására.
- Számos olyan szolgáltatást nyújt, melyek a modellek Eclipse-be integrált kezelését segítik.

SWT és Nebula

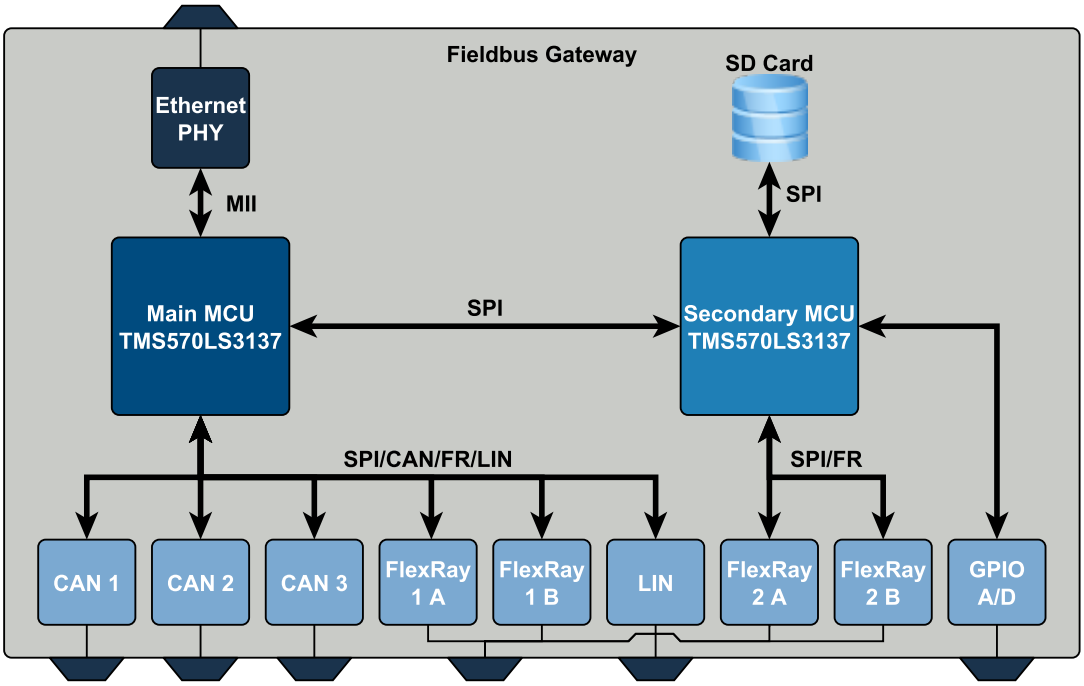
Az Eclipse alapú alkalmazások grafikus felhasználói felülete (GUI) az SWT (Standard Widget Toolkit) komponensgyűjteményre épül. Szemben a Java nyelv két elterjedt, grafikus felhasználói felületek létrehozására szolgáló eszköztárával – az AWT-vel és a Swinggel –, az SWT nem része a szabvány Java API-nak. Kifejezetten Eclipse alapú alkalmazások felhasználói felületeinek létrehozására fejlesztették ki. Az komponensgyűjtemény az operációs rendszer elemeit használja fel, így az ezt használó alkalmazások kinézete minden platformon igazodik az adott ablakkezelő rendszerhez.

A Trace szoftver felhasználói felületének megvalósítása során számos olyan GUI elemre volt szükségem, mely nem volt megtalálható a szabványos SWT komponensek között. Ilyenek például a szignálok megjelenítésére, vagy vezérlésére szolgáló indikátorok és kontrollok, valamint a jelek időbeli változásának vizualizációjára szolgáló grafikus elemek. A Nebula Project az SWT komponenseinek kiegészítését célozza meg különböző az SWT szolgáltatásaira épülő, de nem szabványos GUI elemek létrehozásával.

Fieldbus Gateway

A kommunikációs interfészt biztosító hardver – úgynevezett Fieldbus Gateway – felépítése *(1. ábra)* és az eszköz szolgáltatásai:

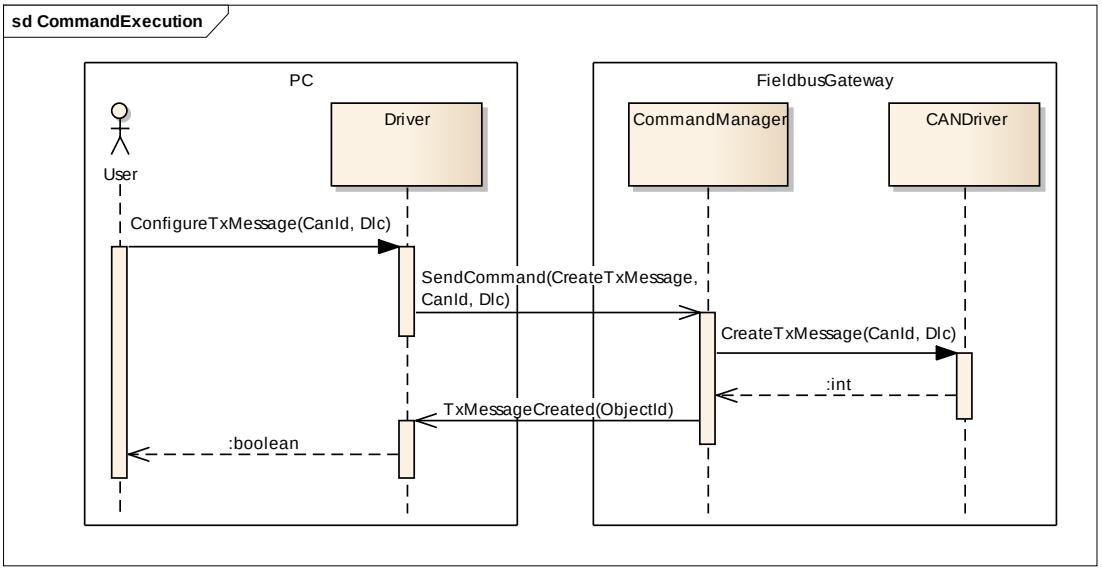
- Ethernet kapcsolat a PC-vel való kommunikációhoz
- 3 darab független CAN csatorna
- 2 darab (2 x 2) FlexRay csatorna belül összekötve, így képes az eszköz önmagában is működtetni egy FlexRay hálózatot
- 1 darab LIN csatorna
- Számos GPIO és A/D port
- SD kártya, mely felhasználható a konfiguráció tárolására, valamint logolásra



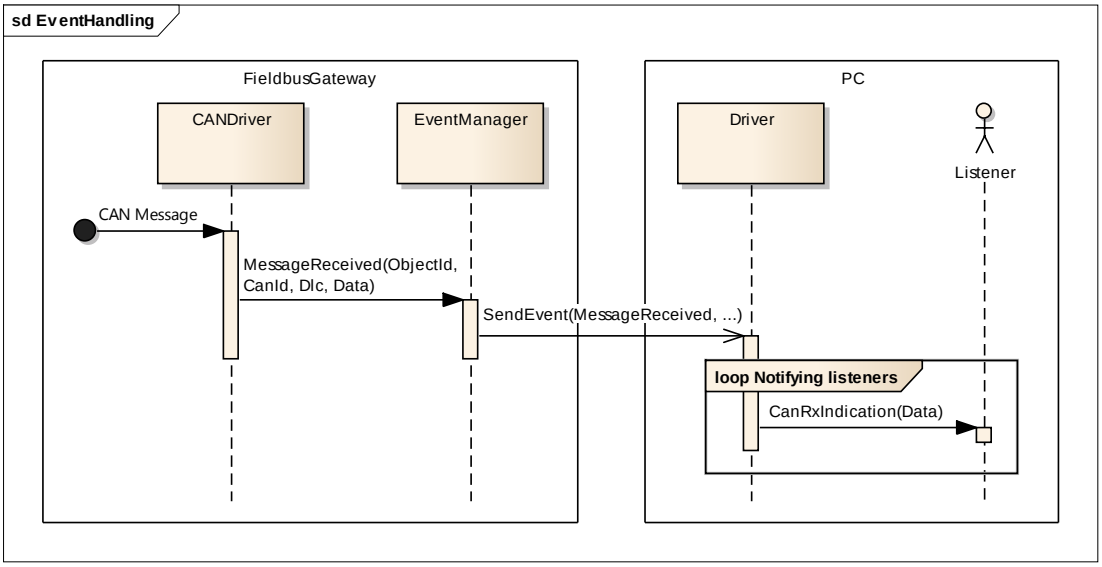
1. ábra. *A Fieldbus Gateway felépítése*

A Gateway-jel a kommunikáció Etherneten keresztül, UDP/IP protokollon lehetséges. A driver által nyújtott Java nyelvű API segítségével különböző parancsok adhatók ki a Gateway számára. A Java API metódusai addig nem térnek vissza, amíg a Gatewayen végrehajtandó parancs futása be nem fejeződött. Miután ez megtörtént, az eszköz egy válaszüzenetben tájékoztatja erről a drivert, elküldve neki a parancs esetleges visszatérési értékét. A meghívott Java metódus ezt követően visszatér, átadva a felhasználónak a kapott értékeket *(2. ábra)*.

Parancsok végrehajtásán kívül a Gateway képes a PC-n futó szoftvert különböző eseményekről is tájékoztatni. Egy esemény bekövetkeztekor a Gateway egy időbélyeggel ellátott üzenetben tájékoztatja erről a drivert, majd a driver értesíti az adott eseményre feliratkozott objektumokat *(3. ábra)*.



2. ábra. *Egy parancs végrehajtásának folyamata*



3. ábra. *Egy esemény jelzésének folyamata*

1. fejezet

Az AUTOSAR mint modellezési nyelv

Az autóipar robbanásszerű fejlődésével a hangsúly a hardverről egyre inkább áttevődött a szoftverre. A vezérlőegységeken futó szoftverek tesztelése bonyolult és időigényes feladat – különösen biztonságkritikus alkalmazások esetén – miközben egyre élesebb verseny folyik a beszállítók között a piacra kerülési idő tekintetében. A különböző autógyártóknál és beszállítóknál alkalmazott fejlesztési és tervezési folyamatok, valamint adattárolási formátumok eltérőek, az esetlegesen ebből fakadó félreértések és kompatibilitási problémák megoldása pedig szintén idő- és költségigényes feladat. Nem ritka, hogy egy-egy gyártó, több hardver platformra is fejleszt mely szintén jelentős többletmunkát igényel.

Az AUTOSAR egy szabványgyűjtemény mely a fenti problémákra kínál megoldást. A név egy betűszó (AUTomotive Open System ARchitecture), melynek magyar feloldása: Autóipari Nyílt Rendszerarchitektúra. A szabványt – mely számos korábban elterjedt ISO és ASAM szabványt is átvett – az autóipari vállalatokból létrejött AUTOSAR konzorcium fejleszti és publikálja 2003 óta. Célja a fenti problémák megoldása a fejlesztési folyamat termékeinek és eszközeinek minél több ponton való újrafelhasználhatóságával.

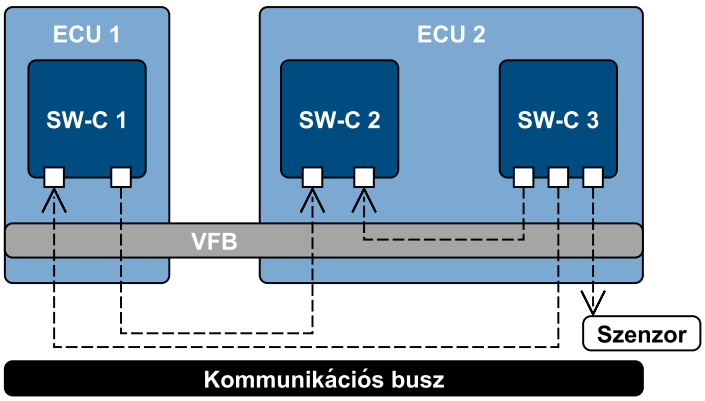
A szabványrendszer alapvetően három részre osztható:

- Definiál egy fejlesztési módszertant, valamint előírja a fejlesztés során alkalmazandó betartandó folyamatokat.
- Definiál egy szigorúan rétegzett szoftverarchitektúrát mely a szoftvert hardverfüggő és hardverfüggetlen rétegekre osztja, így lehetségessé válik a hardverfüggetlen komponensek újrafelhasználása különböző platformokon. Ezzel a V modell implementációs és tesztelési fázisának időtartama is jelentősen lerövidülhet.
- Definiál egy modellezési nyelvet, melynek segítségével a vezérlőegységeken futó szoftver leírható a magas szintű funkcióktól kezdve, az alacsony, hardverspecifikus rétegekig. Egységes adattárolási formátumot ír elő a modellek tárolására és megosztására, ezzel megkönnyítve a megrendelők és beszállítók közti kommunikációt.

Diplomatervezés feladatom elméleti hátterét főként a szabvány harmadik, modellezéshez kapcsolódó része szolgáltatta. Az autógyártók általában a beszállítók rendelkezésére bocsájtják az adott vezérlőegységre vonatkozó információkat tartalmazó modellt, az úgynevezett ECU Extractot. Ezen modell tartalmaz minden információt ahhoz, hogy egy a kommunikáció megfigyelésére alkalmas eszköz a felhasználó minimális beavatkozásával felkonfigurálható legyen.

1.1. Az AUTOSAR rétegzett szoftverarchitektúrája

Az AUTOSAR alapkoncepciója, hogy az ECU funkcionalitását a hardvertől független, atomi szoftverkomponensek (SW-C) valósítják meg. A komponensek egymással és a különböző, hardverszinten megvalósított szenzorokkal és beavatkozókkal szabványos AUTOSAR interfészeken keresztül kommunikálnak. A kommunikáció az úgynevezett Virtuális Függvénybuszon (VFB) történik, melynek feladata a szoftverkomponensek összekötése és a rendszer fizikai határainak elrejtése. Így akár két különböző ECU-n futó szoftverkomponens is teljesen transzparens módon kommunikálhat egymással *(1.1. ábra)*.



1.1. ábra. *Szoftverkomponensek kommunikációja*

A szoftverkomponensek hardverfüggetlenségét az AUTOSAR egy rétegzett, szigorúan definiált szoftverarchitektúrával segíti elő *(1.2. ábra)*. Az architektúra három fő rétegre tagolódik:

Alkalmazás Réteg - Application Layer

Ebben a rétegben kapnak helyet a már ismertetett, az ECU funkcionalitását megvalósító szoftverkomponensek. A komponensek szabványos AUTOSAR interfészeken, az RTE-n keresztül kommunikálnak egymással.

Futtató Környezet - Runtime Environment (RTE)

Ez a réteg egy az ECU-t leíró modell alapján generált „glue” kódréteg. Az RTE valósítja meg a fent látható Virtuális Függvénybuszt, melynek feladata a komponensek közti transzparens és interferenciamentes kommunikáció biztosítása. Amennyiben egy

1.2. ábra. *Az AUTOSAR rétegzett szoftverarchitektúrája*

komponens egy másik ECU-n található komponenssel kíván kommunikálni, az RTE továbbítja a kérést a BSW réteg felé.

Alapszoftver Réteg - Basic Software Layer (BSW)

A réteg feladata, hogy a hardvertől és az ECU felépítésétől független interfészeket, és magas szintű szolgáltatásokat nyújtson az RTE, valamint a szoftverkomponensek számára. A BSW réteg számos szoftvermodulból áll, melyek további három rétegbe rendeződnek:

Mikrokontroller Absztrakciós Réteg - Microcontroller Abstraction Layer

A BSW legalsó rétege. Ebben a rétegben kapnak helyet az adott mikrokontroller különböző perifériáit kezelő driverek. Feladata a mikrokontroller-specifikus funkciók elfedése és szabványos – a konkrét vezérlőtől független – interfész biztosítása az ECU Absztrakciós Réteg számára. A réteg megvalósítása erősen kötődik az adott vezérlő típusához.

ECU Absztrakciós Réteg - ECU Abstraction Layer

Egységes felületet nyújt a mikrovezérlő belső és külső perifériáihoz való hozzáféréshez függetlenül az ECU-n belüli elhelyezkedésüktől és a vezérlőegységhez való kapcsolódásuk módjától. Ebben a rétegben kapnak helyet a külső perifériákhoz tartozó driverek is. Az ECU Absztrakció Réteg megvalósítása már független az alkalmazott mikrovezérlőtől, de erősen függ a vezérlőegység hardverkialakításától.

Szolgáltatás Réteg - Services Layer

A BSW legfelső rétege, mely magas szintű (kommunikációs-, diagnosztikai-, hálózati-menedzsment-, memória-) szolgáltatásokat nyújt az RTE és a szoftverkomponensek számára.

1.2. Az AUTOSAR tervezési módszertana

A rendszer tervezése az autógyártóknál, a szoftverkomponensek és -kompozíciók, valamint a köztük lévő kapcsolatok definiálásával kezdődik *(1.3. ábra)*. A kompozíciók több atomi komponensből illetve esetlegesen további kompozíciókból állhatnak, így a tervezés hierarchikusan történhet, a funkciók egyre alacsonyabb szintre való lebontásával. A rendszert alkotó elemek összeköttetéseinek, valamint belső viselkedésének modellezése a Software Component Template [4] alapján történik. Ezen a szinten még nem látszanak az ECU-k határai, a leírás csupán az elemek közötti kapcsolatokat és azoknak – a rendszer tervezésének szempontjából lényeges – tulajdonságait tartalmazza. Az autógyártó nem tervezi meg a szoftver részleteit, csupán a fontosabb funkciók interfészeit, ahol egy-egy funkció tipikusan egy-egy ECU által lesz megvalósítva. Az ECU-kon futó szoftver részletes megtervezése a beszállítók feladata.

A következő lépésben a komponenseket konkrét erőforrásokhoz, vezérlőegységekhez rendelik *(1.3. ábra)*. A tervezés ezen fázisa a System Description Template [5] alapján történik. Először definiálják a vezérlőegységeket és azok kommunikációs interfészeit, majd hozzájuk rendelik az azokon futó komponenseket. Ezt az összes vezérlőegységet tartalmazó modellt hívják System Description-nek, vagy System Model-nek.

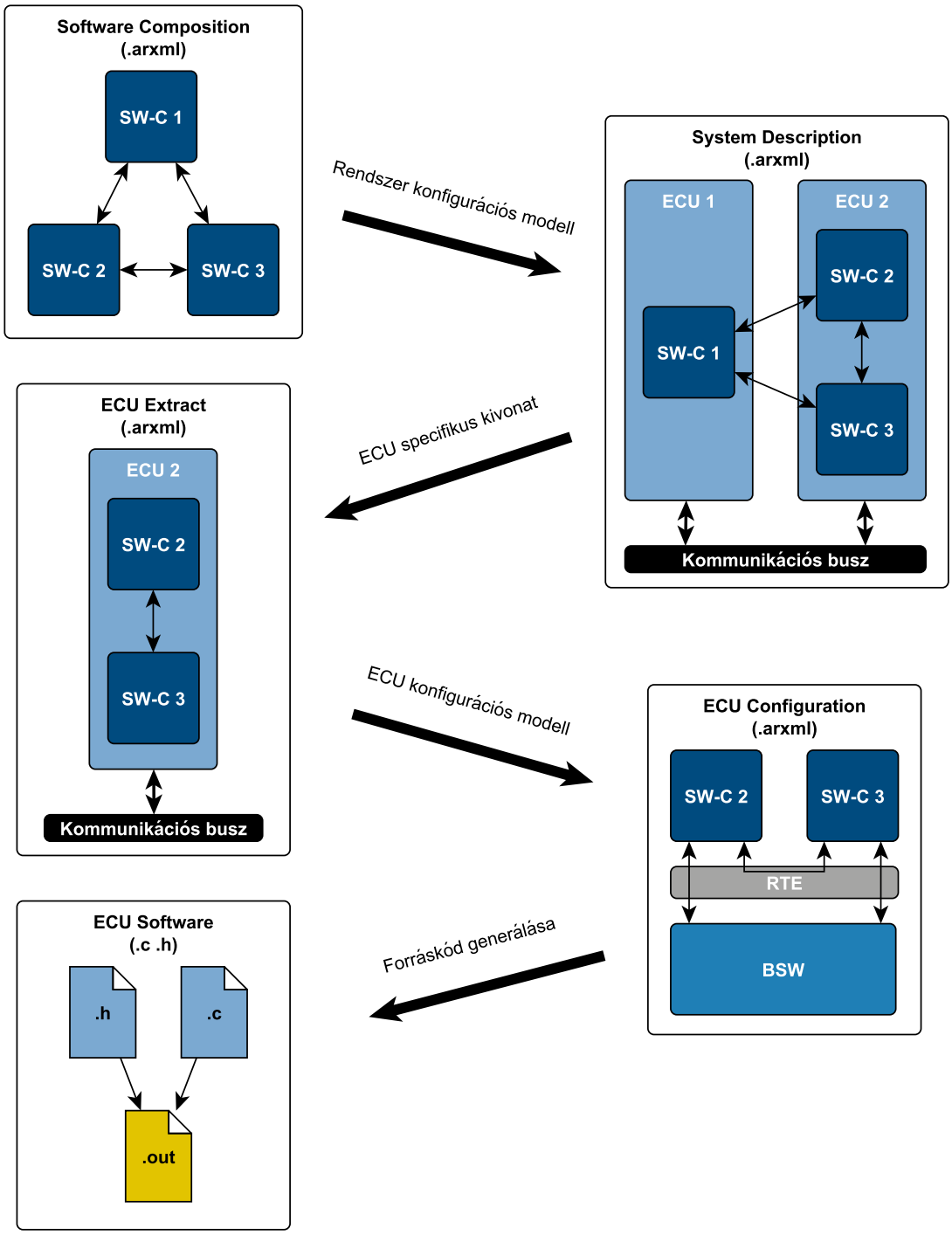
A beszállítóknak saját termékük fejlesztéséhez nincs szüksége – a legtöbb esetben nem is nyilvános – teljes rendszert leíró modellre, csupán a saját vezérlőegységüket és annak környezetét interfész szinten leíró kivonatra. Ezt a kivonatot, mely már csak egyetlen ECU-t, valamint annak kommunikációs csatornáit és az azon közlekedő adatokat tartalmazza nevezik ECU Extractnak. Az ECU Extract felépítésében és szintaktikájában teljes egészében megegyezik a System Modellel, annak részhalmazát képezi. A modell kommunikációhoz – és így feladatomhoz is – kapcsolódó elemeit a későbbiekben részletesen ismertetem.

A beszállítók a kapott ECU Extract alapján már elvégezhetik saját alkalmazásuk architektúrájának további finomítását (kompozíciók lebontását, komponensek részletes tervezését), valamint az Alkalmazás réteget kommunikációs szempontból kiszolgáló BSW modulok konfigurációjának előállítását. A BSW modulok konfigurációjának leírása az ECU Configuration Template [6] alapján lehetséges.

Az ECU Configuration az AUTOSAR modellezési nyelvének egy különleges területe, mivel azonos szinten biztosít lehetőséget konfigurációs modellek paramétereinek definiálására, valamint azok konkrét példányainak létrehozására és konfigurációs adatokkal való feltöltésére. Feladatom megoldása során kézenfekvő megoldás volt a nyelvnek ezen szabadon bővíthető részét felhasználnom arra, hogy az általam fejlesztett szoftvereszköz konfigurációját előállítsam és tároljam.

Az AUTOSAR modellek tárolása minden szinten szabványos `.arxml` fájlokban lehetséges. Ez egy XML alapú fájlformátum, mely kompatibilitást biztosít a fejlesztési lánc különböző résztvevői között.

Az utolsó lépésben történik az ECU konfigurációját leíró modell alapján, a már lefordítható, majd a vezérlőegységen futtatható forráskódok generálása.



1.3. ábra. *Az AUTOSAR módszertana*

1.3. Kommunikáció az AUTOSAR-ban

A kommunikáció folyamata és így az azt leíró modell is az architektúra rétegzett felépítését követi *(1.8. ábra)*. A továbbítandó adatok egyre nagyobb csomagokba kerülnek, míg végül összeáll a már fizikai hálózaton elküldhető adatkeret. A vezérlőegységek közti kapcsolatok leírását a System Template [5] Communication fejezete tárgyalja. A következőkben a metamodell feladatomhoz kapcsolódó elemeit ismertetem.

1.3.1. Alkalmazás réteg

Az AUTOSAR-ban a kommunikáció legmagasabb szintű adategysége az úgynevezett szignál. A komponensek portjaikra szignálokat írhatnak ki, és onnan szignálokat olvashatnak be. Az RTE feladata ezen jelek konzisztens és interferencia-mentes továbbítása a kommunikáló felek között. A szignálok lehetséges mérete bitszámban adható meg, mely akármilyen értéket felvehet.

A kommunikációt leíró System modell legfelső szintjén helyezkednek el a rendszerben előforduló szignálokat és szignálcsoportokat reprezentáló `SystemSignal` és `SystemSignalGroup` elemek, melyek a komponensek modellezésénél kapnak szerepet.

1.3.2. Interakciós és hálózati réteg

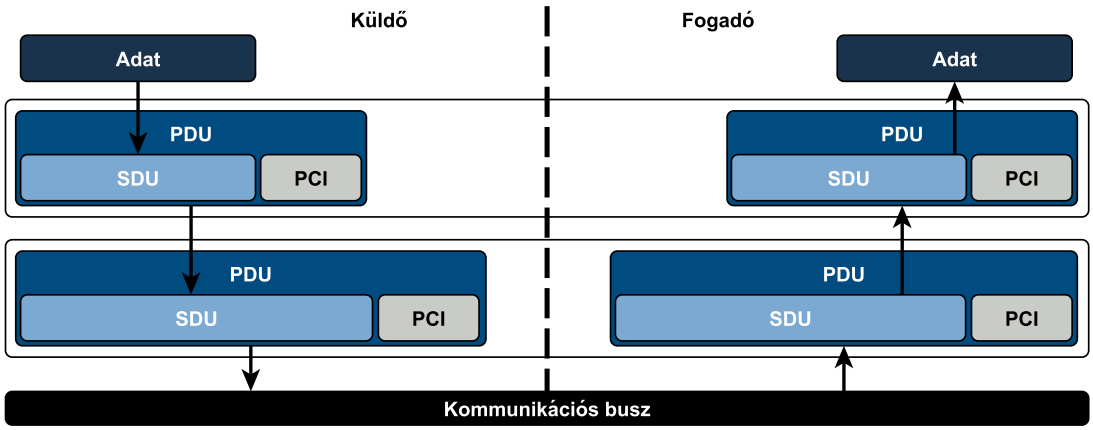
A BSW réteg kommunikációs szoftvermoduljai közti adatcsere alapegységei a PDU-k[1]. A PDU-k OSI modell szerinti felépítése látható az ábrán *(1.4. ábra)*. A csomagokban az adott réteg által továbbítandó „hasznos” információ az SDU[2] mezőben helyezkedik el, míg a PCI[3] mező az adott protokoll által értelmezhető vezérlő információkat tartalmazza. A fogadó oldalon ezen információ alapján történik meg az adatok feldolgozása és továbbítása a felső rétegek felé.

A PDU-k leírására a modell `Pdu` osztálya szolgál. Az AUTOSAR PDU-knak funkciójuktól függően több típusa létezik, melyek a modellben mind a `Pdu` osztályból származtathatók. Feladatom megoldása során ezek közül két típus bírt jelentőséggel, a szignálokat tartalmazó `ISignalIPdu`, valamint a szállítási rétegbeli protokollok által használt `NPdu`.

Szignál I-PDU

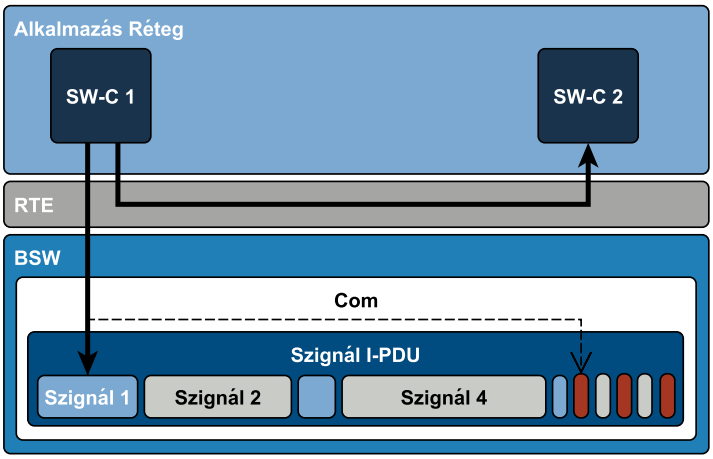
Az RTE-n keresztül az Alkalmazás réteg felől érkező (távoli komponenseknek továbbítandó) szignálok első állomása a Com BSW modul, mely a szignálokat I-PDU-kba[4] csomagolja *(1.5. ábra)*, majd továbbítja a kommunikációs folyamatban résztvevő következő modulnak. Egy I-PDU SDU mezőjében több szignál is helyet kaphat, ugyanakkor nem biztos, hogy egy adott PDU továbbításakor annak minden szignálja érvényes adatot tartalmaz. Az I-PDU-k

[1] Protocol Data Unit
[2] Service Data Unit
[3] Protocol Control Information
[4] Interaction Layer PDU



1.4. ábra. *A PDU-k OSI modell szerinti általános felépítése*

PCI mezőjében elhelyezkedő update bitek jelzik, hogy az adott szignál, vagy szignálcsoport helyén található információ éppen érvényes-e.



1.5. ábra. *A Szignál I-PDU-k felépítése*

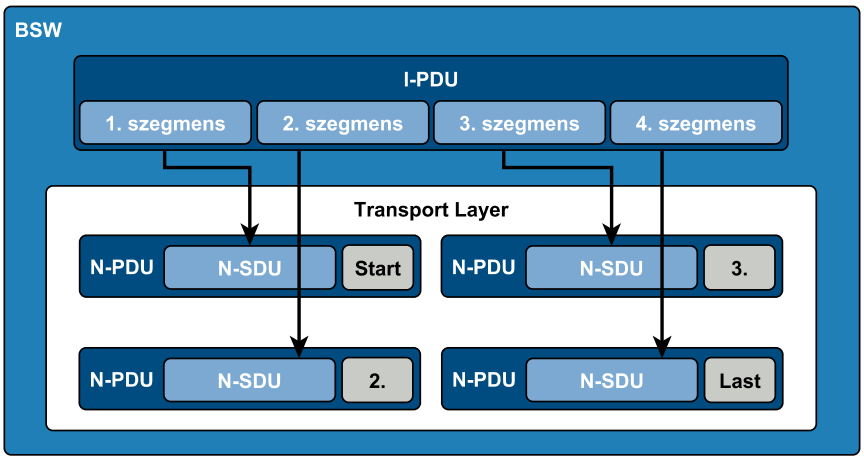
A szignálokat és szignálcsoportokat ezen a szinten a modell `ISignal` és `ISignalGroup` osztálya reprezentálja. Mindegyik rendelkezik egy-egy referenciával az általa reprezentált Alkalmazás rétegbeli `SystemSignal` vagy `SystemSignalGroup` elemre. Az Szignál I-PDU modellbeli megfelelője a `Pdu`-ból az `IPdu`-n keresztül származó `ISignalIPdu` osztály.

A PDU által tartalmazott szignálokat az `ISignalToIPduMapping` osztály írja le, mely egy referencián keresztül az `ISignalPdu`-hoz köti az adott szignált vagy szignálcsoportot, kiegészítve azokat a PDU-ban elfoglalt pozíciójukra vonatkozó információkkal. Amennyiben a szignálhoz/szignálcsoporthoz update bit is tartozik, annak PDU-ban való pozícióját is ez az elem adja meg.

Szállítási réteg I-PDU (N-PDU)

Előfordulhat, hogy az alkalmazott kommunikációs protokoll által megengedett maximális méretű keretnél hosszabb adatcsomagot kell továbbítani a hálózaton. Ezek általában diagnosztikai üzeneteket tartalmazó – a DCM modulnak szóló – PDU-k, esetleg nagyméretű szignálokat tartalmazó Szignál I-PDU-k. (A CAN esetében a maximális keretméret 8 bájt, míg a FlexRay esetében 254 bájt. Ez általános kommunikációs célokra bőven elegendő, ám a gyártás valamint szerviz során a vezérlőegységek felprogramozását is célszerűen ezeken a protokollokon végzik el.)

Ezen probléma megoldására jöttek létre a különböző szállítási rétegbeli protokollok, például FlexRay esetén a FlexRay Transport Layer (ISO 10681-2) [7], vagy CAN esetén a CAN Transport Layer (ISO 15765-2) [8]. Ezen protokollok feladata a maximális méretet meghaladó különböző típusú I-PDU-k küldő oldalon való szegmentálása kisebb méretű, szállítási rétegbeli N-PDU-kra *(1.6. ábra)*, majd a fogadó oldalon való összeállítása. Az N-PDU-k leírására az `IPdu`-ból származó `NPdu` osztály segítségével van lehetőség.



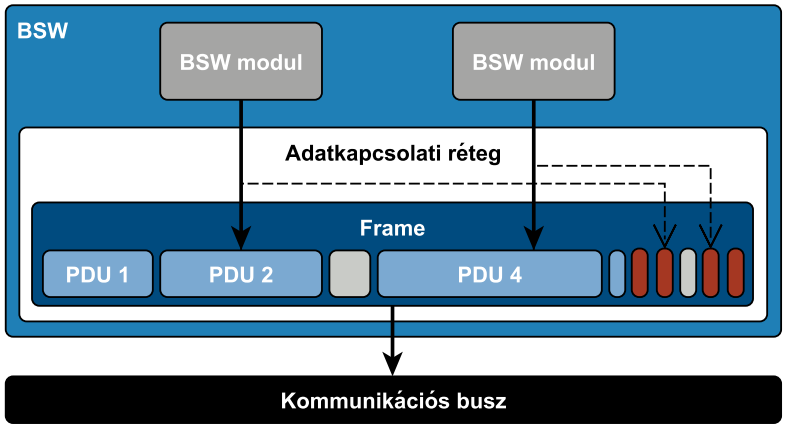
1.6. ábra. *Szállítási protokollok működése*

1.3.3. Adatkapcsolati réteg

Az adatkapcsolati rétegben történik meg a hálózati keretek (frame-ek) összeállítása, majd továbbítása a kommunikációs interfészen. A hosszabb üzeneteket is továbbítani képes protokollok – például FlexRay – esetén lehetőség van egy keretben több PDU elhelyezésére is *(1.7. ábra)*. Ezen a szinten is előfordulhat, hogy egy adott üzenetben nem minden PDU tartalmaz éppen érvényes adatot. Ennek jelzésére ebben az esetben is a PDU-khoz tartozó update bitek szolgálnak.

A keretek felépítését a `Frame` osztály írja le. A `Frame` leszármazottai a különböző protokollokhoz tartozó kereteket jelképező osztályok, például a `CanFrame` és a `FlexRayFrame` osztályok. A keret felépítését a `Frame` által tartalmazott `PduToFrameMapping`-ek segítségével adhatjuk meg. Minden egyes `PduToFrameMapping` hivatkozik egy `Pdu`-ra, valamint

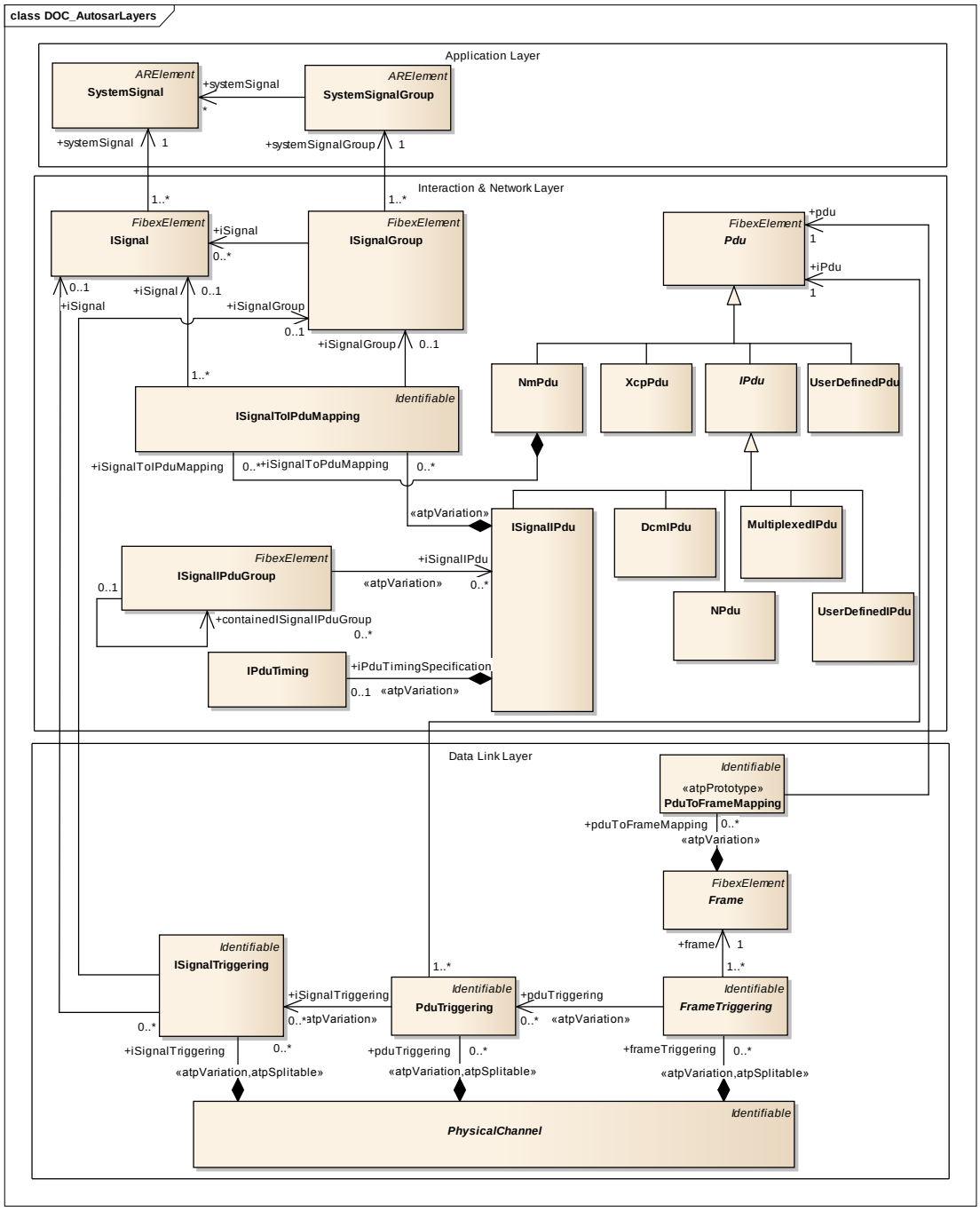
kiegészíti azt a keretben elfoglalt helyét meghatározó információkkal. Szintén itt adható meg (ha van) PDU-hoz tartozó udpate bit pozíciója is.



1.7. ábra. *A keretek összeállítása*

A `Frame` a kereteknek csupán a szerkezetét írja le, a hálózaton való továbbításhoz, illetve fogadáshoz szükséges információkat azonban nem tartalmazza. A hálózati paraméterek definiálására a `FrameTriggering` osztály szolgál, melynek protokoll specifikus leszármazottai az általa hivatkozott `Frame` mellé megadják a protokollhoz kötődő egyéb paramétereket is, például `CanFrameTriggering` esetén ilyenek az `identifier`, vagy `FlexRayFrameTriggering` esetén a `slotId` és `cycleRepetition` paraméterek.

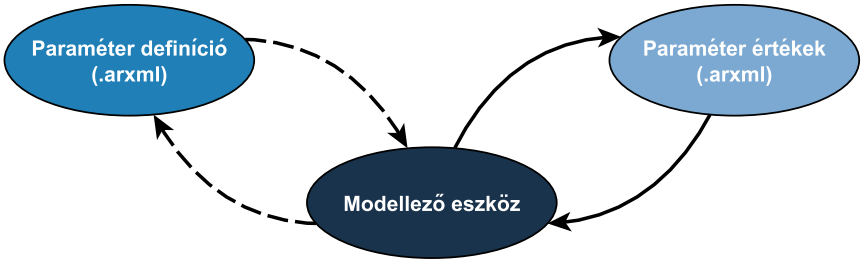
Egy vezérlőegység akár több hálózathoz is csatlakozhat. A `PhysicalChannel` osztály reprezentálja az ECU kommunikációs csatornáit. A `PhysicalChannel`-nek szintén léteznek protokollfüggő leszármazottai (például `CanPhysicalChannel` és `FlexRayPhysicalChannel`) melyeken keresztül az adott protokollhoz tartozó hálózati paramétereket érhetjük el. A csatornákon továbbítandó vagy azokon fogadandó kereteket a csatornák által tartalmazott `FrameTriggering`-ek definiálják.



1.8. ábra. *A kommunikációt leíró AUTOSAR modell*

1.4. ECU Configuration

Az AUTOSAR modellezési nyelvének egy speciális területe a vezérlőegységen futó szoftver konfigurációját leíró modell. Ez a fejlesztési láncban résztvevő legalacsonyabb szintű leírás, mely alapján már generálható a vezérlőegységen futtatható szoftver forráskódja. A modell a BSW rétegbeli szabványos valamint adott esetben gyártóspecifikus szoftvermodulok és az RTE konfigurációját tartalmazza. Mivel a konfigurálandó modulok implementációja – és így konfigurációs paraméterei is – bizonyos esetekben eltérhetnek a szabványban rögzítettől, ezért elkerülhetetlenül az azokat leíró modellnek is képesnek kell lennie ezen eltérések kezelésére. Az AUTOSAR modellek ezen szintjén lehetőség van a konfigurációs paraméterek megadásán kívül azok definíciójának bővítésére, szerkesztésére is. A szabvány erre vonatkozó előírásait az ECU Configuration [6] dokumentum tartalmazza.

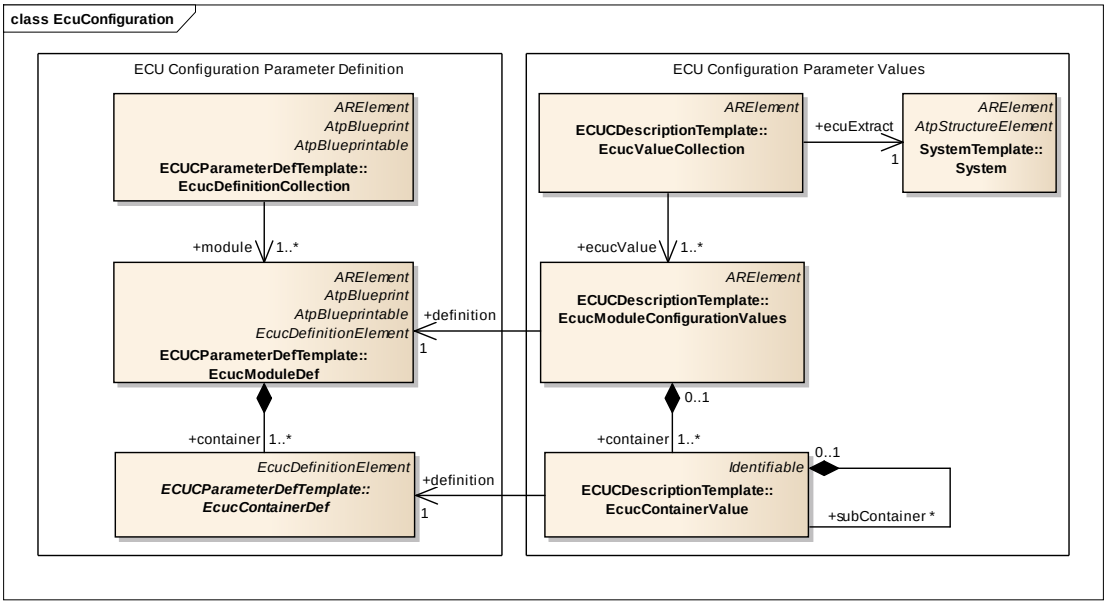


1.9. ábra. *Az ECU konfigurációs modellek kezelése*

A modellező eszköz az AUTOSAR által definiált elemkészlet felhasználásával lehetővé teszi konfigurációs definíciók létrehozását, majd az alapján példány-konfigurációk kitöltését. A konfigurációs modellek értelmezéséhez elengedhetetlenül szükséges a definíciós modellek ismerete, azok mintegy térképként szolgálnak a paramétertér bejárásához *(1.9. ábra)*.

A szabvány által biztosított elemkészlettel minden adatszerkezet-modellezési igény kielégíthető. Struktúrák – ún. containerek – definiálhatók, azokon belül argumentumok, más elemekre való referenciák, valamint további containerek hozhatók létre. Az egyes elemekhez ezek mellett – például multiplicitásra vonatkozó – kényszerek adhatók meg.

Az egy ECU-hoz tartozó szoftvermodulok konfigurációit (`EcucModuleConfigurationValues`) egy `EcuValueCollection` elem fogja össze, mely tartalmaz egy referenciát a konfigurálandó vezérlőegységhez készült ECU Extractra *(1.10. ábra)*.



1.10. ábra. *ECU Configuration*

2. fejezet

A feladat megvalósítása

A feladat megvalósítása két fő részből áll. Egyrészt a szoftver működését leíró adatmodell definiálásából, és felépítéséből, másrészt pedig a modellezett működést megvalósító Java szoftver létrehozásából. A fejezetben először ismertetem a szoftver működésének hátterében álló adatmodellt, majd bemutatom a futásidejű objektumok együttműködését.

2.1. A szoftver működését leíró adatmodell

A szoftver működésének adatmodelljét egy az ECU Parameter Definition Template alapján definiált ECU Configuration modell írja le. Ennek előnye, hogy szabadon definiálható benne bármilyen (alapvetően adatszerkezeteket leíró) modell, valamint a vállalatnál fejlesztett modellező eszköz számos ponton támogatja ezen modellek kezelését.

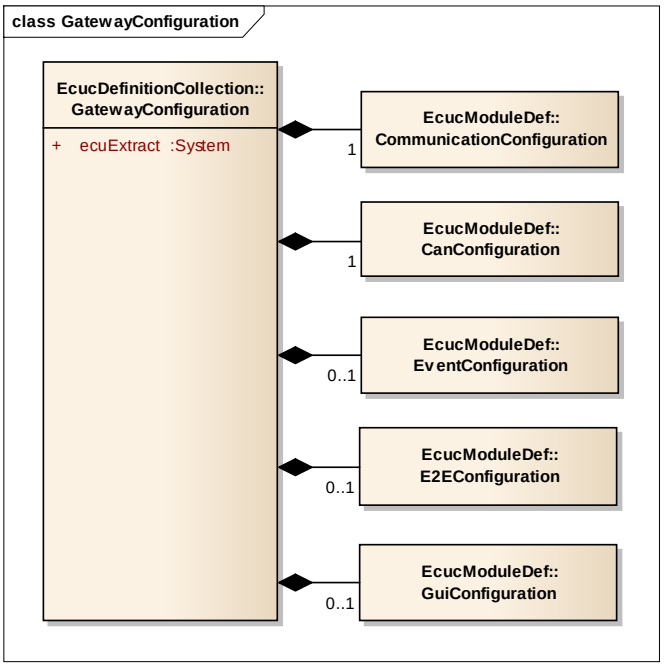
A Restbus Szimulátor és Trace konfigurációja az ábrán látható főbb modulokból áll *(2.1. ábra)*, melyek a konfigurációs adatokat funkciójuk szempontjából csoportokba foglalják.

2.1.1. CommunicationConfiguration

A `CommunicationConfiguration` modul felépítése látható az ábrán *(2.2. ábra)*. Ez az `EcucModuleDef` a kommunikációban résztvevő protokollfüggetlen elemek – szignálok, PDU-k, valamint Frame-ek – felépítését írja le. Multiplicitása az `EcucValueCollection`-ön belül 1 azaz mindenképpen léteznie kell belőle pontosan egy példánynak.

Signal: A `Signal` Container szolgál a szignálok reprezentálására. A konfigurációban nem kell kötelezően léteznie szignáloknak, hiszen előfordulhat, hogy a kommunikációt csak PDU-k, vagy Frame-ek szintjén szeretnénk vizsgálni, így ezen elem multiplicitása 0..*. A `Signal` Container tartalmazza továbbá az adott szignál hosszát bitekben, valamint a kezdeti értékét.

Pdu: A `Pdu` Container a rendszerben előforduló megfigyelendő, vagy szimulálandó PDU-k leírását szolgálja. PDU-knak szintén nem kell feltétlenül léteznie, Frame szintű megfigyelés, vagy szimuláció esetén, így a Container multiplicitása 0..*. A Container



2.1. ábra. *A Restbus Szimulátor és Trace konfigurációja*

tartalmazza a PDU hosszát, valamint egy enumerációt, mellyel a PDU típusa adható meg. A `Pdu` továbbá amennyiben típusa `I_PDU`, `PduSignal` objektumokon keresztül hivatkozhat az általa tartalmazott szignálokra.

PduSignal: A PDU-k által tartalmazott szignálok leírására szolgál a `PduSignal` Container. Tartalmaz egy referenciát az adott `Signal`-ra, valamint megadja mellé a szignál, valamint a hozzá tartozó (ha van) update bit pozícióját (bitszámban), és a szignál bájtsorrendjét.

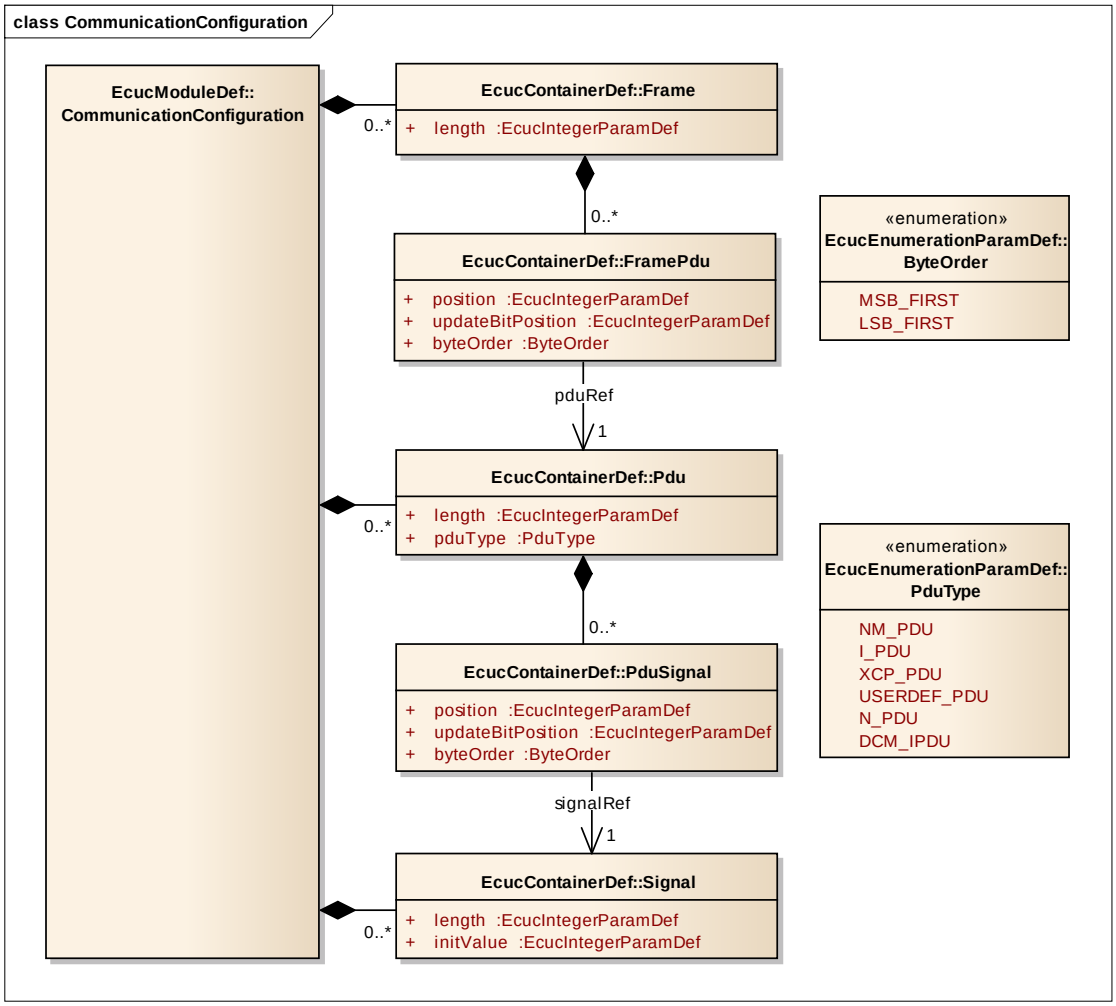
Frame: Ez a Container egy frame felépítését írja le. Megadja a keret hosszát bájtokban, valamint `FramePdu` referenciákon keresztül a keret által tartalmazott PDU-kat.

FramePdu: A `FramePdu` szolgál a `Frame` által tartalmazott PDU-k leírására. Tartalmaz egy referenciát az adott `Pdu`-ra, valamint a `PduSignal`-hoz hasonlóan megadja hozzá a frame-ben való elhelyezéséhez szükséges további információkat.

PduType: Az enumeráció a `Pdu`-k típusának megadására szolgál. Az implementáció jelenleg az `I_PDU` – mely szignálokat tartalmazó I-PDU-t jelent –, valamint `N_PDU` – mely szállítási protokollok által használt PDU-t jelent –, típusokat támogatja.

2.1.2. CanConfiguration

A `CommunicationConfiguration` a kommunikációt protokollfüggetlen megközelítésben írta le. A `CanConfiguration` modul látja el a kereteket CAN specifikus paraméterekkel, valamint tartalmazza a Gateway CAN csatornáinak konfigurációját.



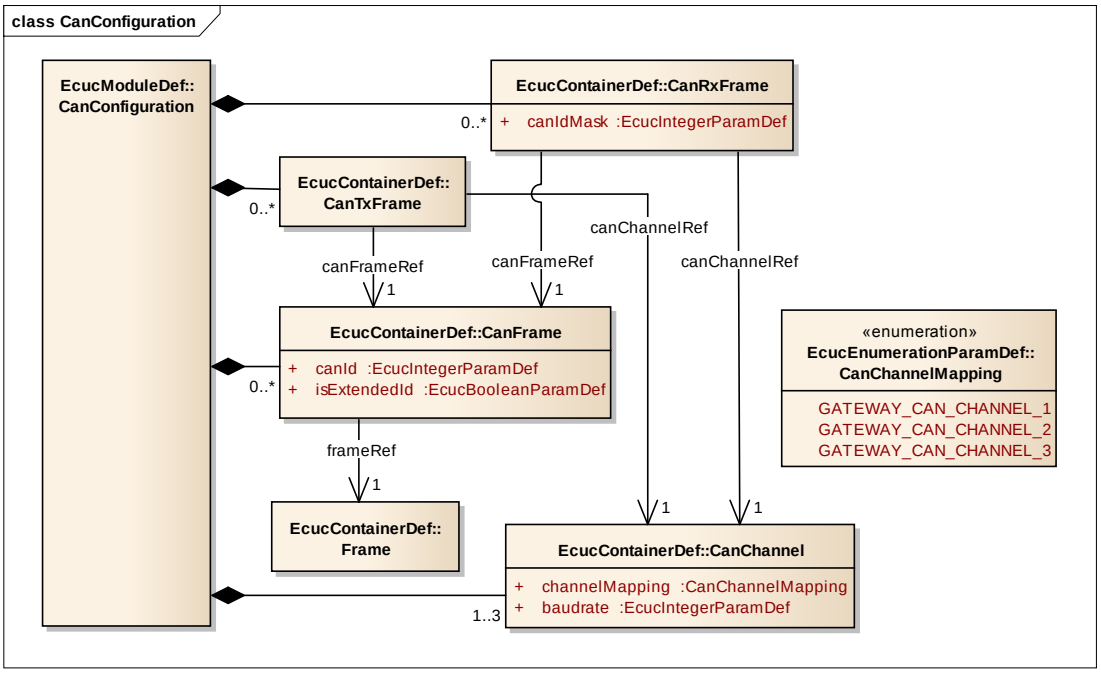
2.2. ábra. *A CommunicationConfiguration felépítése*

CanChannel: A `CanChannel` Container a Gateway egy-egy CAN csatornáját reprezentálja. Mivel a hardver három csatornával rendelkezik, és a kommunikációhoz egyre mindenképpen szükség van, ezen elem multiplicitása 1..3. Két paramétere van, az adatsebességet leíró `baudRate`, valamint a Gateway fizikai csatornáihoz való hozzárendelést megadó `channelMapping`.

CanFrame: Az osztály egy CAN üzenet leírására szolgál. Hivatkozik egy `CommunicationConfiguration` modulban definiált `Frame`-re, és ellátja azt CAN azonosítóval.

CanTxFrame: Egy CAN hálózaton a Restbus Szimulátor által elküldendő üzenetet reprezentál. Egyetlen paramétere egy referencia az adott CAN üzenetre (`CanFrame`).

CanRxFrame: Egy a Trace által megfigyelendő CAN üzenetet ír le. Hivatkozik a megfelelő `CanFrame`-re, valamint opcionálisan megadható mellé az üzenet fogadásánál használt ID maszk.



2.3. ábra. *A CanConfiguration felépítése*

2.1.3. EventConfigutration

Az `EventConfiguration` modul a Restbus Szimulátor működéséhez szükséges adatokat tartalmazza. Segítségével események és az események bekövetkeztekor végrehajtandó akciók definiálhatók, így tetszőleges üzenetküldési (restbus szimulációs) szekvencia leírható. Az események forrása lehet:

- időzítő, mely lehet periodikus is (`TimerSource`)
- egy szignál értékének megváltozása (`SignalSource`)
- egy üzenet megérkezése, fogadása (`RxIndicationSource`)
- egy üzenet elküldéséről szóló visszaigazolás (`TxConfirmationSource`)

Egy esemény bekövetkeztekor végrehajtandó akciók szintén több csoportba sorolhatók:

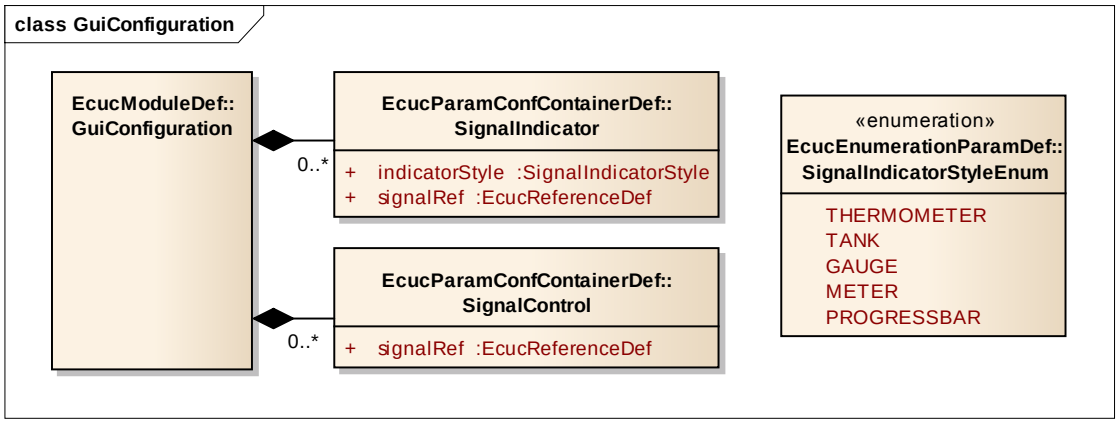
- egy üzenet elküldése (`TransmitAction`)
- más események aktiválása (`ActivateEventsAction`)
- szignál értékének megváltoztatása (`SetSignalAction`)
- akciók listájának végrehajtása (`StartActionBlockAction`)
- akciók listájának feltételes végrehajtása (`ConditionallyStartActionBlockAction`)

2.1.4. E2EConfiguration

Az E2EConfiguration a biztonságkritikus szignálok védelmére szolgáló AUTOSAR szabványos end-to-end protection támogatására szolgál. Az AUTOSAR E2E a PDU-kban elhelyezkedő szignálok csoportját egészíti ki CRC illetve üzenet számláló értékekkel, ezzel valósítva meg azok alkalmazás szintű védelmét. A szabvány ezen területe igen szerteágazó. Az end-to-end protection szabványos megvalósítása önmagában kitenne egy Diplomatervezés feladatot, a Restbus Szimulátor és Trace egyelőre nem támogatja.

2.1.5. GuiConfiguration

A modul a szoftver felhasználói felületének leírását tartalmazza. Megadhatók azok a szignálok melyeket külön indikátoron is szeretnénk megfigyelni, illetve melyeket külön kontrollal futásidőben szeretnénk változtatni.



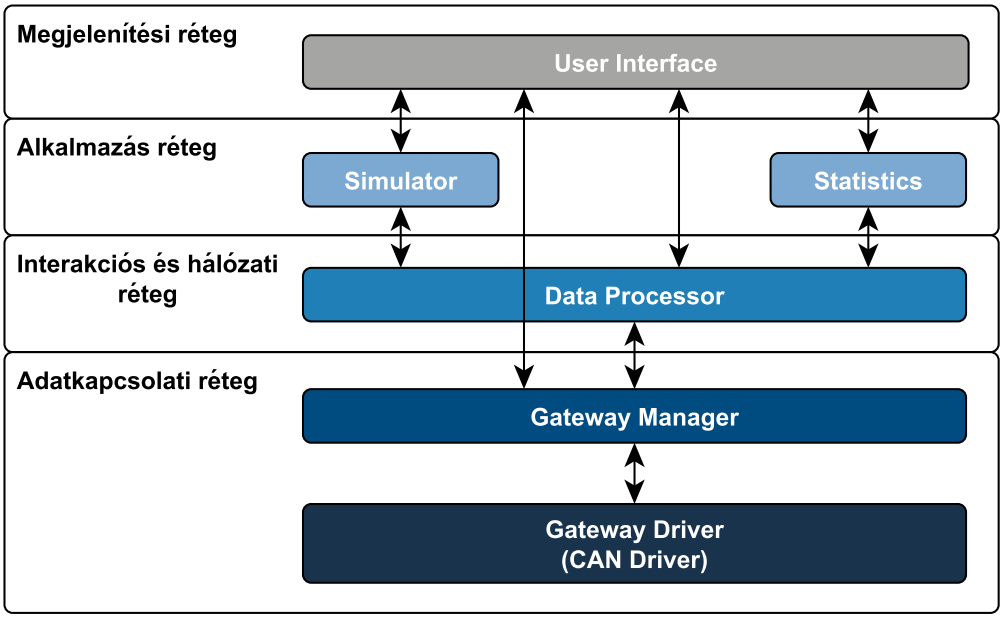
2.4. ábra. *A GuiConfiguration felépítése*

SignalIndicator Egy szignál értékét megjelenítő indikátort ad meg. Tartalmaz egy `SignalIndicatorStyle` enumerációt, mely az indikátor megjelenését határozza meg, valamint egy referenciát a megjelenítendő szignálra.

SignalControl Egy szignálhoz tartozó kontrollt hoz létre a felhasználói felületen. Egyetlen referenciát tartalmaz az általa változtatandó szignálra.

2.2. A szoftver felépítése

A szoftver felépítése – az AUTOSAR architektúrájához hasonlóan – rétegekre osztható *(2.5. ábra)*. A legalsó rétegben helyezkedik el a Fieldbus Gateway kezeléséért és az alacsony szintű funkciók irányításáért felelős Gateway Manager. A Gateway Manager felett található az adatok feldolgozását, frame-ek, PDU-k és szignálok kezelését végző Data Processor. A Data Processor felett a magas szintű funkciókat megvalósító további egységek találhatók, mint a Restbus Simulator, valamint a Statistics blokk. A legfelső réteg a User Interface, mely minden réteggel kapcsolatban áll, így képes bármely szintről származó információk megjelenítésére, valamint általa bármelyik szint működése befolyásolható.



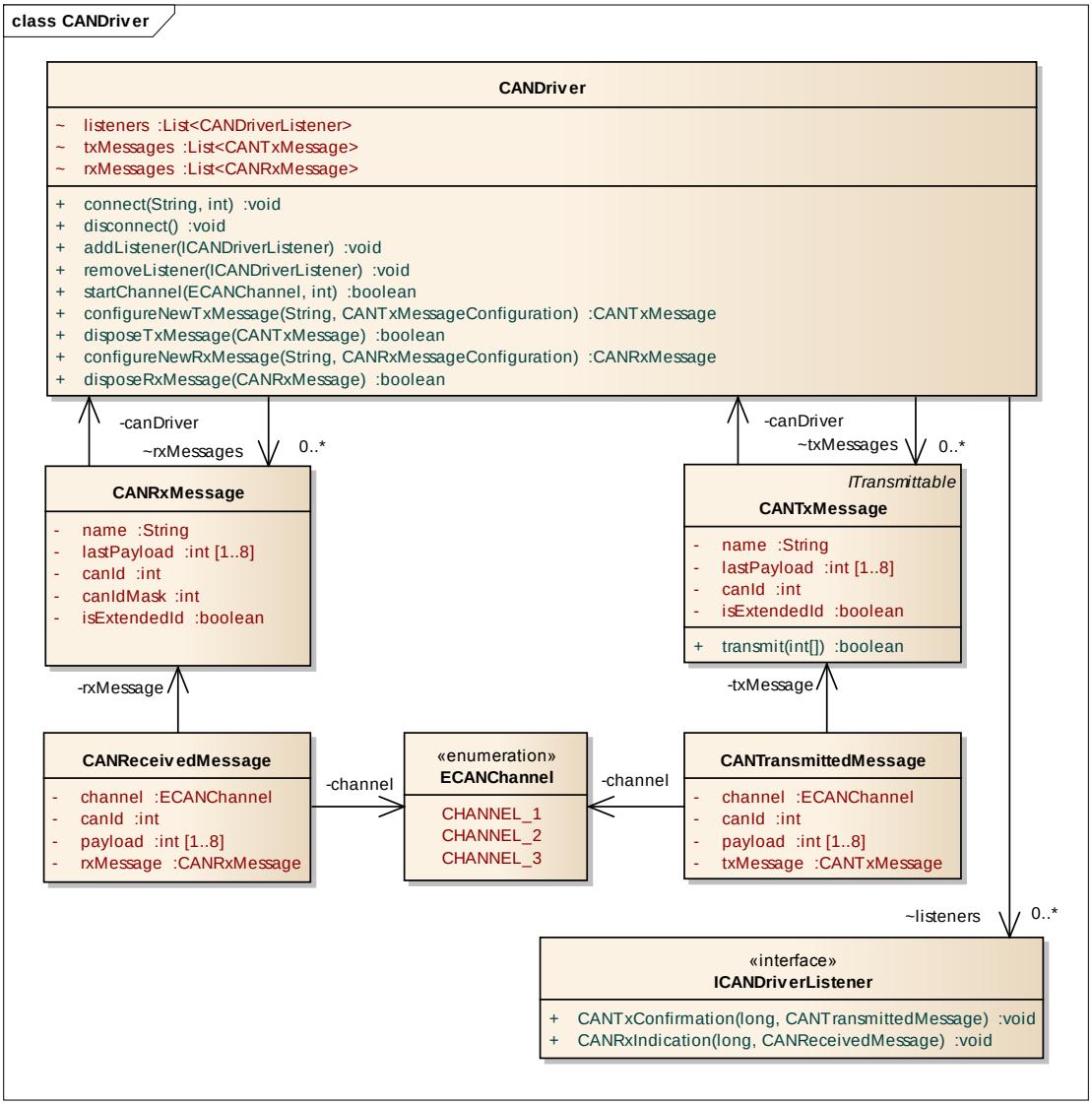
2.5. ábra. *A szoftver vázlatos felépítése*

2.2.1. Driver

A Fieldbus Gateway szolgáltatásainak elérése a Gateway Driveren keresztül lehetséges. A driver által szolgáltatott API – feladatom szempontjából releváns osztályai és metódusai – az alábbi ábrán láthatók *(2.6. ábra)*.

A Gateway-hez való kapcsolódás a driver `connect(...)` metódusán keresztül a Gateway IP címének, valamint portszámának megadásával lehetséges. A kapcsolat felépülése után a CAN csatornák elindítása a `startChannel(...)` metódus segítségével lehetséges.

A `CANDriver`-el való interakció során az Rx és Tx üzeneteket a `CANRxMessage` és `CANTxMessage` objektumok reprezentálják. Az üzenetek létrehozása a driver `configureNewRxMessage(...)` és `configureNewTxMessage(...)` metódusával lehetséges, mely a paraméterében megkapott adatok alapján a Gateway-en létrehozza a megfelelő hardver szintű üzenet objektumokat, valamint azok PC oldali megfelelőjét. Az üzenetek



2.6. ábra. *A driver által nyújtott API*

által tartalmazott adatok azok könnyebb kezelhetőségének érdekében a szoftver minden szintjén integer tömbként jelennek meg. A tömb elemeinek értéke nem lehet nagyobb 255-nél, CAN üzenetek esetén pedig a hossza nem haladhatja meg a nyolcat.

A `CNRxMessage` objektumokhoz kötődő üzenetek fogadásáról a driver az arra feliratkozott `ICANDriverListener` interfészt megvalósító objektumokat az interfész `CANRxIndication(...)` metódusán keresztül értesíti. A metódus egy mikroszekundum felbontású időbélyeget, valamint egy a fogadott üzenettel kapcsolatos részleteket tartalmazó `CANReceivedMessage` objektumot vesz át a paraméterlistáján.

Egy üzenet elküldése az adott üzenetet reprezentáló `CANTxMessage` objektum `transmit(int[])` metódusával lehetséges. Amikor a Gateway az üzenetet sikeresen továbbította a hálózaton erről egy visszaigazolást küld a PC-nek, melyről szintén értesítést kapnak az arra feliratkozott `ICANDriverListener` interfészt megvalósító objektumok. Az

üzenetek elküldéséről szóló értesítések jelzésére az interfész `CANTxConfirmation(...)` metódusa szolgál. Ez a metódus is egy mikroszekundum felbontású időbélyeget, valamint egy az elküldött üzenettel kapcsolatos részleteket tartalmazó `CANTransmittedMessage` objektumot vesz át a paraméterlistáján.

A `CANTransmittedMessage` és a `CANReceivedMessage` osztályok is egyaránt tartalmazzák az elküldött, vagy fogadott üzenet adattartalmát (`payload`), a CAN azonosítóját (`canId`), a csatorna azonosítóját melyen az esemény bekövetkezett `channel`, valamint egy referenciát a `CANTxMessage` vagy `CANRxMessage` objektumra (`txMessage` vagy `rxMessage`). Az értesítésekre való feliratkozás a driver `addListener(...)` metódusán keresztül lehetséges.

2.2.2. Keretek, PDU-k, szignálok

A kereteket, PDU-kat és szignálokat futásidőben megvalósító osztályok felépítése a konfigurációs adatmodellt követi, lényegében annak Java nyelvű leképzése. Az adatok tárolása mellett azonban, képesek azok konzisztenciájának felügyelésére is. Az osztálydiagramon a megértés szempontjából releváns attribútumokat és metódusokat ábrázoltam *(2.7. ábra)*.

FrameImpl

A `FrameImpl` osztály a `Frame` konfigurációs container futásidejű megfelelője. Egy protokollfüggetlen adatkeret reprezentáció, mely az adat tárolásán kívül egyéb szolgáltatásokat is nyújt. A keretben található PDU-kat, az adatmodellhez hasonlóan futásidőben is mapping osztályok kötik a kerethez. Egy üzenet beérkezésekor az adott `FrameImpl` `updateData(int[])` metódusának meghívásával frissíthetők a keret által tartalmazott adatok. Ilyenkor az objektum nem csak a saját adatmezőjét frissíti, hanem a mappingjeit végigjárva feldolgozza a beérkezett adatbájtokat, és a megfelelő PDU-k adattartalmát is frissíti a hozzájuk tartozó `PduImpl` objektum `updateData(int[])` metódusának meghívásával. Eközben az esetleges bájtsorrend cserét is elvégzi. A metódus visszaadja a keret frissítésének következtében frissült egyéb objektumok (PDU-k és szignálok) listáját.

A változások terjedése nem csak felülről lefelé lehetséges, hanem alulról felfelé is. A `FrameImpl` osztály `assembleFrame()` metódusának meghívásával kérhető, hogy saját adattartalmát a mappingek felhasználásával állítsa össze az általa tartalmazott PDU-k adataiból.

PduImpl

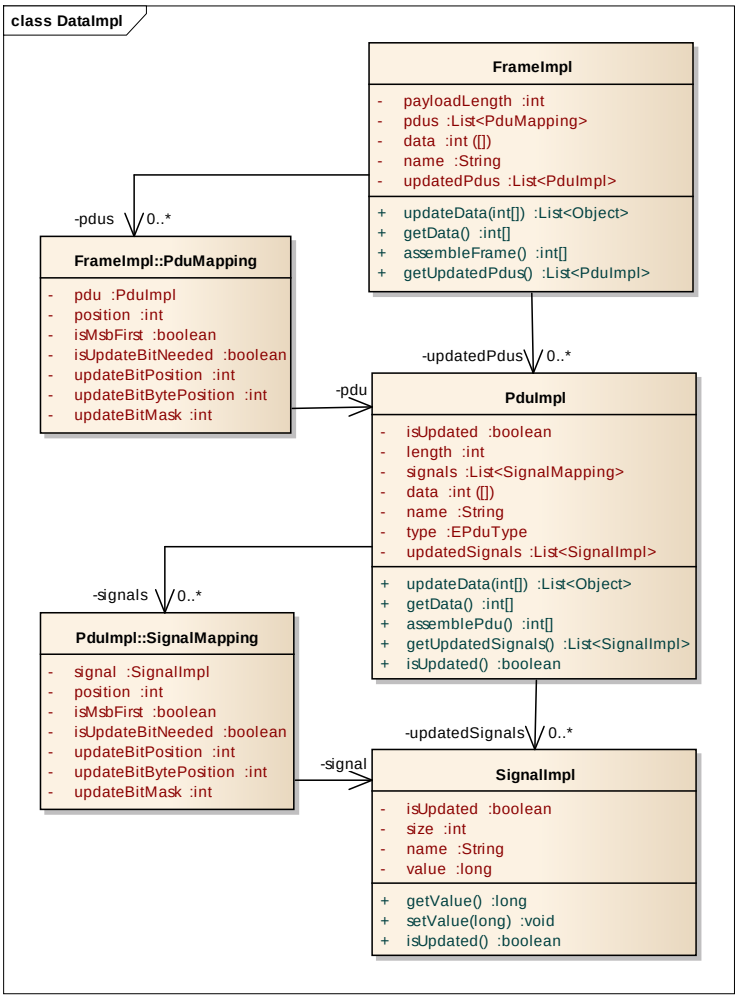
A `PduImpl` osztály az adatmodellbeli `Pdu` container futásidejű megfelelője. Szintén mappingeken keresztül hivatkozik az adott PDU által tartalmazott szignálokra. És a `FrameImpl`-hez hasonlóan felügyeli saját és az általa tartalmazott szignálok adatait. A PDU adatainak frissítésére szolgáló `updateData(int[])` metódus a `FrameImpl` osztály azonos nevű metódusához hasonlóan, szintén visszaadja a frissítés által érintett objektumok (ez esetben

szignálok) listáját.

A szignálok értékének PDU-kból való kinyerése, valamint PDU-kban való elhelyezése nem egyszerű feladat. Egy szignál a PDU adatbájtjait tartalmazó tömb bármelyik bitpozíciójától kezdődhet, és akárhány bit hosszúságú lehet. A szignál `isMsbFirst` konfigurációs paraméterétől függően pedig a PDU bájtjain csökkenő vagy növekvő sorrendben kell végigmenni a szignál értékének összeállítása során.

SignalImpl

A `Signal` container futásidejű megfelelője. A legkisebb, de legmagasabb absztrakciós szintű adategység. A szignál értékét az osztály egy `long` típusú változóban tárolja, így lehetséges akár 64 bit hosszúságú szignálok kezelése is.



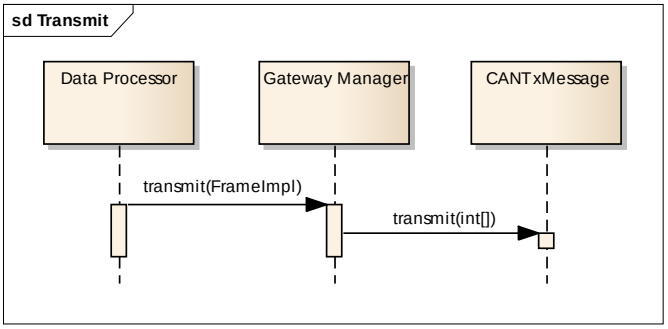
2.7. ábra. *A futásidejű adatszerkezet*

2.2.3. Gateway Manager

Az adatkapcsolati réteg Gateway Managerének feladata – az AUTOSAR architektúrájának adatkapcsolati rétegéhez hasonlóan – a drivertől és az aktuálisan használt kommunikációs protokolltól független interfészt biztosítani a felette található réteg számára. A szoftver inicializációja során kiépíti a kapcsolatot a Gateway-jel, és létrehozza a konfigurációs modell alapján a megfelelő Rx és Tx üzeneteket.

A program futása során a Gateway Manager valósítja meg az összeköttetést a driver `CANTxMessage` és `CANRxMessage` objektumai, valamint a felsőbb réteg protokollfüggetlen `FrameImpl` objektumai között. Lényegében az adatmodell `CanTxFrame`, `CanRxFrame`, `CanFrame` és `Frame` containereinek modellezett viszonyait valósítja meg.

Egy keret elküldése során a Manager megkeresi a kapott `FrameImpl`-hez tartozó `CANTxMessage` objektumot, majd a `FrameImpl` által tartalmazott adatbájtokkal meghívja annak `transmit(int[])` metódusát *(2.8. ábra)*.



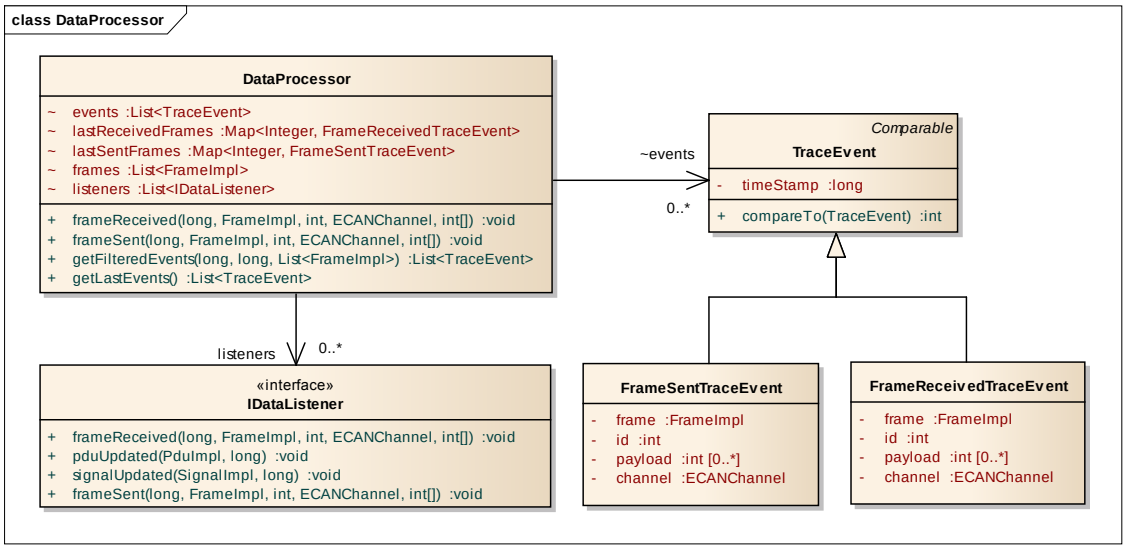
2.8. ábra. *Keretek elküldése a Gateway Manageren keresztül*

A Gateway felől érkező értesítések feldolgozásának első állomása szintén a Gateway Manager. A Manager Implementálja az `ICANDriverListener` interfészt, és az inicializáció során beregisztrálja magát a driver listenerei közé. `CANTxConfirmation(...)` valamint `CANRxIndication(...)` metódusában megkeresi az adott `CANTxFrame` vagy `CANRxFrame` objektumhoz tartozó `FrameImpl` objektumot, és értesíti a Data Processort a keret elküldésének sikerességéről, vagy a keret fogadásáról, átadva annak az eseményhez kapcsolódó további paramétereket is.

2.2.4. Az események feldolgozása

A driver értesítéseinek valódi feldolgozását a Data Processor végzi. Ennek a modulnak a feladata tárolni (logolni), valamint továbbterjeszteni az egész szoftverben a különböző kommunikációval kapcsolatos értesítéseket.

A Data Processor az eseményeket `TraceEvent` objektumok listájaként tartja számon. A `TraceEvent` egy absztrakt osztály mely a belőle származtatott különböző típusú események egységes tárolását teszi lehetővé. A `TraceEvent` egyetlen argumentuma a `long` típusú `timeStamp`, mely az eseményekhez tartozó időbélyeget tárolja. Az osztály implementálja a



2.9. ábra. *Data Processor*

Java Comparable interfészét, melyen keresztül az időbélyeg alapján össze lehet hasonlítani a `TraceEvent` objektumokat.

Két, esetünkben releváns leszármazottja az üzenetek elküldését regisztráló `FrameSentTraceEvent`, valamint az üzenetek érkezését számon tartó `FrameReceivedTraceEvent`. Mindkét osztály tárolja az elküldött, illetve fogadott üzenet adatbájtjait (`payload`), az üzenet azonosítóját (`id`), a csatornát (`channel`), valamint egy referenciát az üzenethez tartozó `FrameImpl` objektumra (`frame`). Ezen adatok alapján az események bármikor visszajátszhatók, kielemezhetők.

A Data Processortól lehetőség van az események szűrt (filterezett) listáját is elkérni. A szűrés történhet időbélyeg tartomány, valamint frame lista alapján is. Ez a modul `getFilteredEvents(...)` metódusán keresztül lehetséges, melynek argumentumlistáján meg kell adni az időbélyeg alsó és felső korlátját, valamint azon `FrameImpl` objektumok listáját, melyekhez kapcsolódóan szeretnénk megkapni az adott időtartományba eső eseményeket.

Értesítések

A Data Processor értesítéseire feliratkozni az `IDataListener` interfészt megvalósító osztályoknak van lehetőségük *(2.9. ábra)*:

frameReceived: Egy `FrameImpl` adatainak frissüléséről tájékoztat.

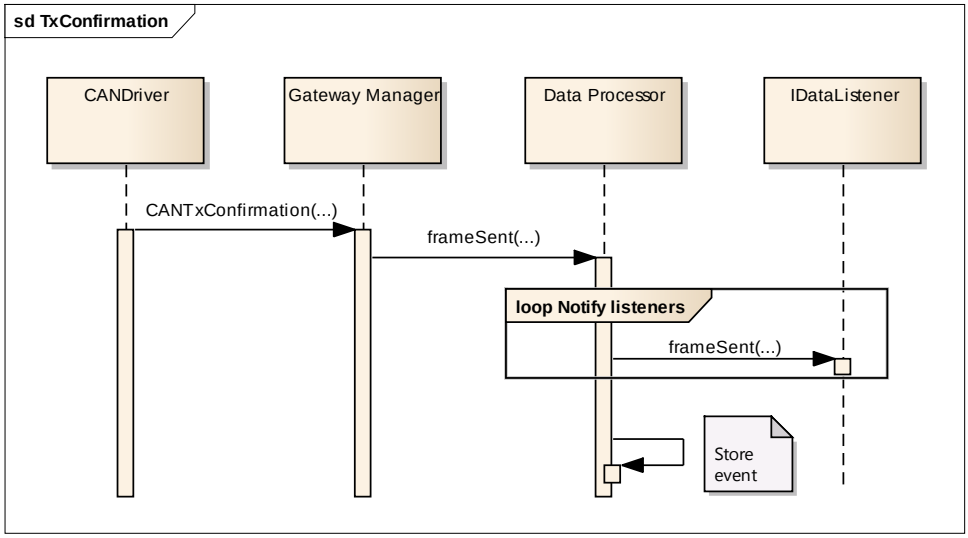
pduUpdated: Egy `PduImpl` adatainak frissüléséről tájékoztat.

frameReceived: Egy `SignalImpl` értékének frissüléséről tájékoztat.

frameSent: Egy `FrameImpl` adatainak sikeres elküldéséről tájékoztat.

TxConfirmation

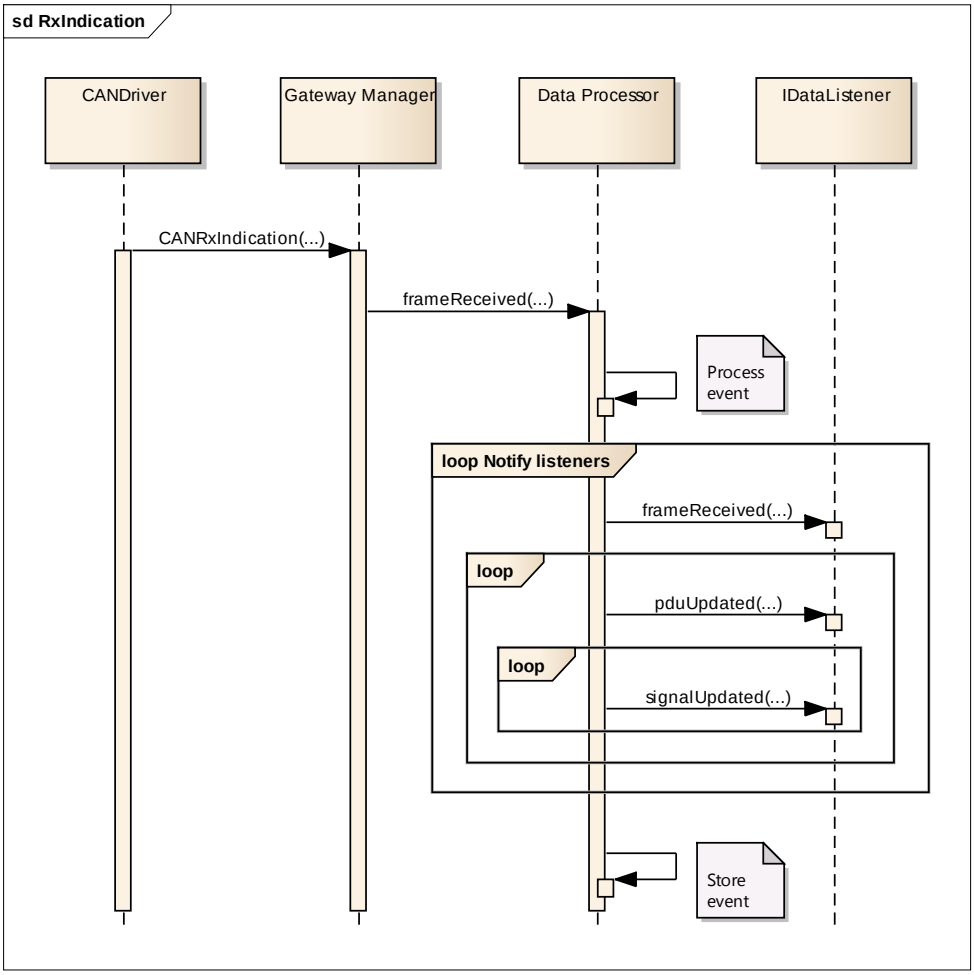
A Gateway Managertől érkező, keretek elküldéséről szóló visszaigazolásokról a Data Processor értesíti a listenereit, a `frameSent(...)` metódusukon keresztül, majd az értesítés alapján létrehoz egy új `FrameSentTraceEvent` objektumot és eltárolja az eseményeket számon tartó listáján.



2.10. ábra. *TxConfirmation*

RxIndication

A keretek fogadásáról szóló értesítések feldolgozása összetettebb feladat *(2.11. ábra)*. A Data Processor először frissíti az adott `FrameImpl` objektum adatbájtjait, melynek hatására az üzenet által tartalmazott `PduImpl` és `SignalImpl` objektumok is frissülnek, majd elkéri a `FrameImpl` által tartalmazott frissített PDU-k listáját annak `getUpdatedPdus()` metódusának segítségével. Ezek után a frissített PDU-któl az általuk tartalmazott frissített szignálok listáját gyűjti össze a `PduImpl getUpdatedSignals()` metódusán keresztül, majd sorra értesíti a listenereket a különböző eseményekről (`frameReceived(...)`, `pduUpdated(...)`, `signalUpdated(...)`).

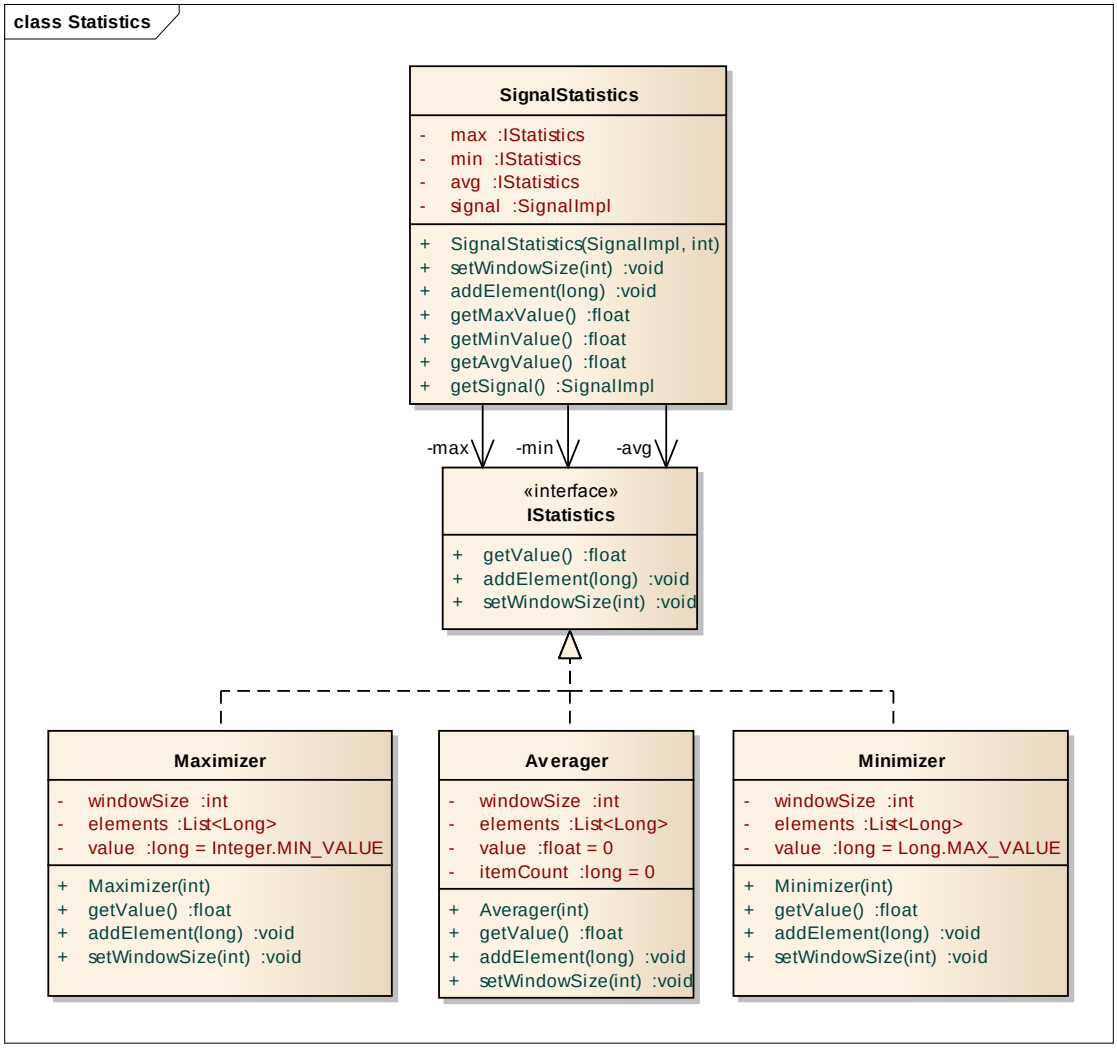


2.11. ábra. *RxIndication*

2.2.5. A statisztikai adatok előállítása

A felhasználói felületen megjelenítendő szignálokhoz kapcsolódó statisztikai adatok számítására több, egyszerű segédosztályt hoztam létre, melyek segítségével egyelőre három alapvető paraméter futásidejű számítására van lehetőség. Meghatározhatjuk a szignál értékének maximumát, minimumát, valamint átlagát.

A különböző paraméterek számítását végző osztályok mind megvalósítják az IStatistics interfészt, melyen keresztül egységesen kezelhetők, és igény szerint új statisztikai funkciókkal bővíthető a megvalósítás. Az interfész `getValue()` metódusa a számított paraméter aktuális értékét adja vissza, míg az `addElement(long)` metódussal új adatelemet szúrhatunk be. Az adatok számításánál alkalmazott mozgóablak mérete a `setWindowSize(int)` metódussal állítható be. Ha az ablakméretet 0-ra állítjuk, azzal az ablakozás kikapcsolható, ilyenkor az összes elemre történik a paraméter kiszámítása.



2.12. ábra. *Statisztikai paraméterek számítása*

Maximum és minimum meghatározása

A maximum meghatározása abban az esetben ha nincs ablakozás rendkívül egyszerű. Az osztály mindig csak az aktuális legnagyobb értéket tárolja, és amennyiben az `addElement()` metódusában kapott új minta annál nagyobb, a tárolt értéket felülírja vele. A `getValue()` metódusnak nincs más dolga, mint visszaadni az eltárolt értéket.

Ha a maximumot csak egy adott szélességű ablakra kell meghatározni, elkerülhetetlen a minták FIFO-ban tárolása és a maximum újraszámolása minden esetben. A vizsgált konfigurációkban a leggyorsabban változó szignál frissülésének periódusideje 250 mikroszekundum volt. Ez sok szignál és nagy ablakméret esetén jelentős overheadet jelent. Mivel a kiszámított paraméter értékére tipikusan ritkábban van szükség, mint amilyen gyakran új minta keletkezik, elég a maximum értékét csak a `getValue()` metódus meghívásánál kiszámítani a tárolt minták alapján. Az osztály a kiszámolt értéket megjegyzi, és mindaddig nem számolja újra, amíg új mintát nem szúrnak be az `addElement()` metódussal.

A minimum számításának módja analóg a fentiekkel.

Átlagérték

Az átlagszámítás rekurzívan történik az előző átlagérték, valamint az új minta alapján. A rekurzív számítási mód futásidő szempontjából előnyös, mivel minden új minta esetén csak egy egyszerű számítást kell elvégezni. Az ablakozás nélküli rekurzív átlagolás úgynevezett predikciós-korrekciós alakja az alábbi formulával írható le [9]:

$$\hat{x}(n+1) = \hat{x}(n) + \frac{1}{n+1}(y(n) - \hat{x}(n))$$

Ha az ablakozás ki van kapcsolva, az átlagolónak nem szükséges tárolnia a múltbeli mintákat, csupán a minták számát (n). Mozgó ablak átlagolás esetén a formula a következőképpen módosul [9]:

$$\hat{x}(n+1) = \hat{x}(n) + \frac{1}{N}(y(n) - \hat{y}(n-N))$$

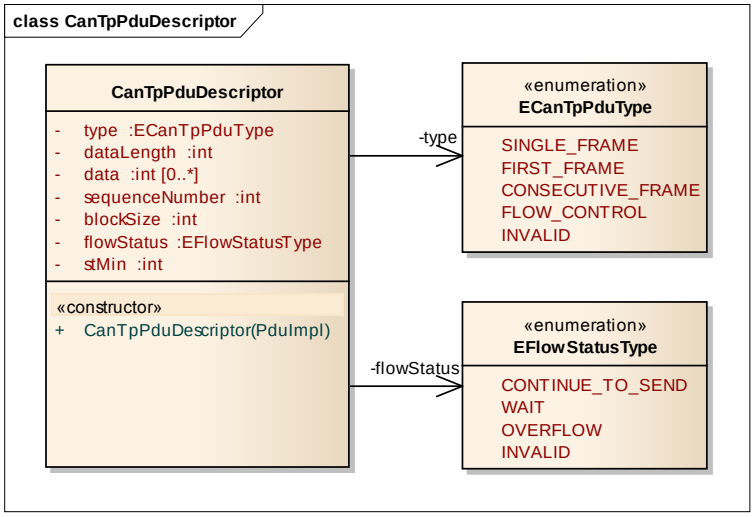
Ebben az esetben szükségessé válik az aktuális ablak mintáinak eltárolása, mivel a korrekciós tagban szerepel az ablakból kilépő minta értéke is.

A statisztikai paraméterek egységes kezelése

A szignállal kapcsolatos statisztikák kezelését a `SignalStatistics` osztály végzi. Azokhoz a szignálokhoz melyek a konfigurációban ki vannak jelölve megjelenítésre, létrejön egy-egy példánya az osztálynak, mely gondoskodik az összes paraméter számításáról. A szignál értékének frissülése esetén elegendő a hozzá tartozó `SignalStatistics` `addElement(long)` metódusát meghívni. Ha szükség van valamely paraméter értékére, az az osztályon keresztül egy dedikált metódus segítségével elérhető.

2.2.6. CAN Transport Layer PDU-k feldolgozása

A CAN Transport Layer által megvalósított protokoll által értelmezhető N-PDU-k feldolgozását, értelmezését az ábrán látható osztály segíti *(2.13. ábra)*.



2.13. ábra. *A CAN Transport Layer PDU-kat leíró osztály*

Az osztály konstruktora egy `PduImpl` objektumot vár, melynek adattartalma alapján inicializálja tagváltozóit. Ezzel gyakorlatilag dekódolta a keretet, és a szükséges paraméterek az objektumtól elkérhetők.

Az osztály `type` argumentuma a keret típusát határozza meg, az üzenet első bájtjának felső négy bitje alapján.

A `dataLength` paraméter csak *First Frame* esetén van értelmezve. Értéke a szegmentált átvitel során átvivendő teljes üzenet hosszát határozza meg.

A `data` mező csak adatokat tartalmazó, tehát *Single Frame*, *First Frame* és *Consecutive Frame* keretek esetén van értelmezve. Az üzenet által szállított hasznos adatbájtokat tartalmazza.

A `sequenceNumber` tagváltozó az adott *Consecutive Frame* sorszámát határozza meg.

A `blockSize` változó csak *Flow Control - Continue To Send* PDU esetén van értelmezve. Értéke a következő átviteli blokk méretét határozza meg.

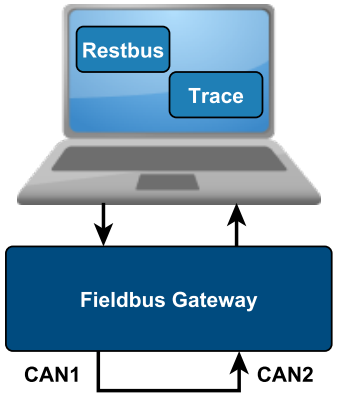
A `flowStatus` enumeráció a *Flow Control* keretek típusát adja meg.

Az `stMin` paraméter szintén csak *Flow Control - Continue To Send* keretek esetén értelmezett. Értéke a két *Consecutive Frame* átvitele között kivárandó időt adja meg.

3. fejezet

A szoftver használata

A szoftver további szolgáltatásait valamint felhasználói felületét egy példán keresztül mutatom be. A példában a Restbus Szimulátor szimulációs funkcióit használom fel a Trace funkcióinak ismertetésére. A Gateway CAN1 és CAN2 csatornáját összekötöttem, így az „saját magával kommunikálva" lehetővé teszi a Restbus Szimulátor szekvenciáinak megfigyelését a Trace-en keresztül *(3.1. ábra)*.



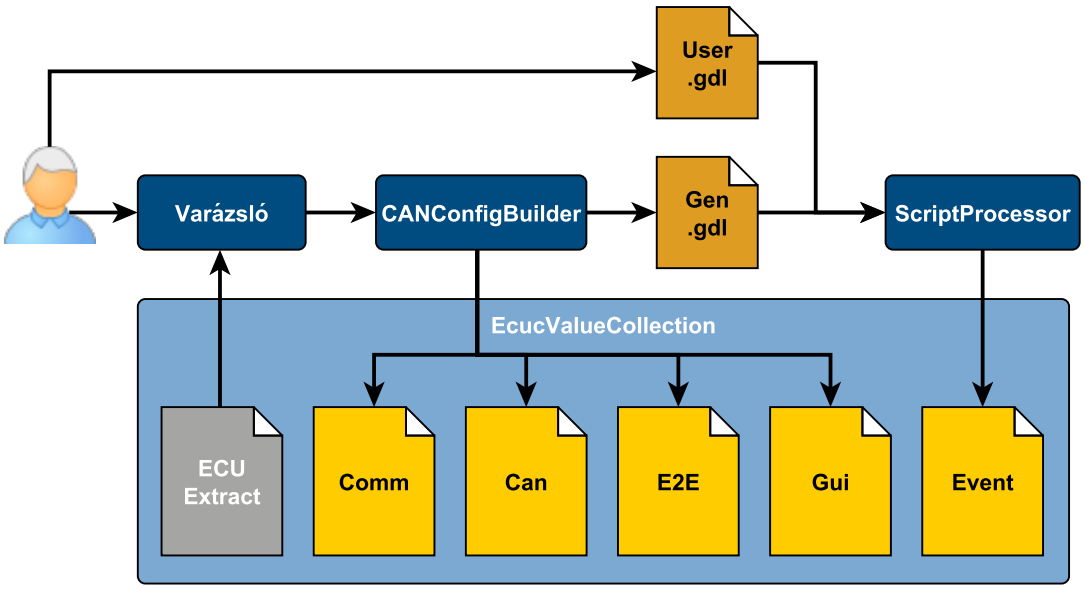
3.1. ábra. *A példában szereplő elrendezés*

A szoftver használatának főbb lépései:

1. Modeling Project majd `EcucValueCollection` létrehozása, ECU Extract megadása
2. Varázsló futtatása az `EcucValueCollection` és az ECU Extract alapján
3. A Restbus Szimulátor szimulációs szekvenciáit definiáló script megírása
4. A Restbus Szimulátor `EventConfiguration` modelljének generálása a script alapján
5. Eclipse Run Configuration létrehozása az eszköz futtatására a modellek alapján, majd az eszköz elindítása
6. Szimuláció elindítása, log elemzése, szükség esetén elmentése

3.1. A konfiguráció összeállításának folyamata

Első lépésben egy „Embedded model project” Autosar Architect projektet kell létrehozni majd beimportálni a szükséges ECU Extract modellt. Az előző fejezetben ismertetett konfigurációs modulokat a szabványnak megfelelően egy `EcucValueCollection` fogja össze. A projektben létre kell hozni egy üres `EcucValueCollection`-t és ki kell tölteni annak `ecuExtract` referenciáját. Ezek után már az `EcucValueCollection`-re jobb egérgombbal való kattintással elindítható a varázsló, mely a hivatkozott ECU Extract alapján végigvezeti a felhasználót a szoftver konfigurációjának folyamatán *(3.2. ábra)*.



3.2. ábra. *A konfiguráció folyamata*

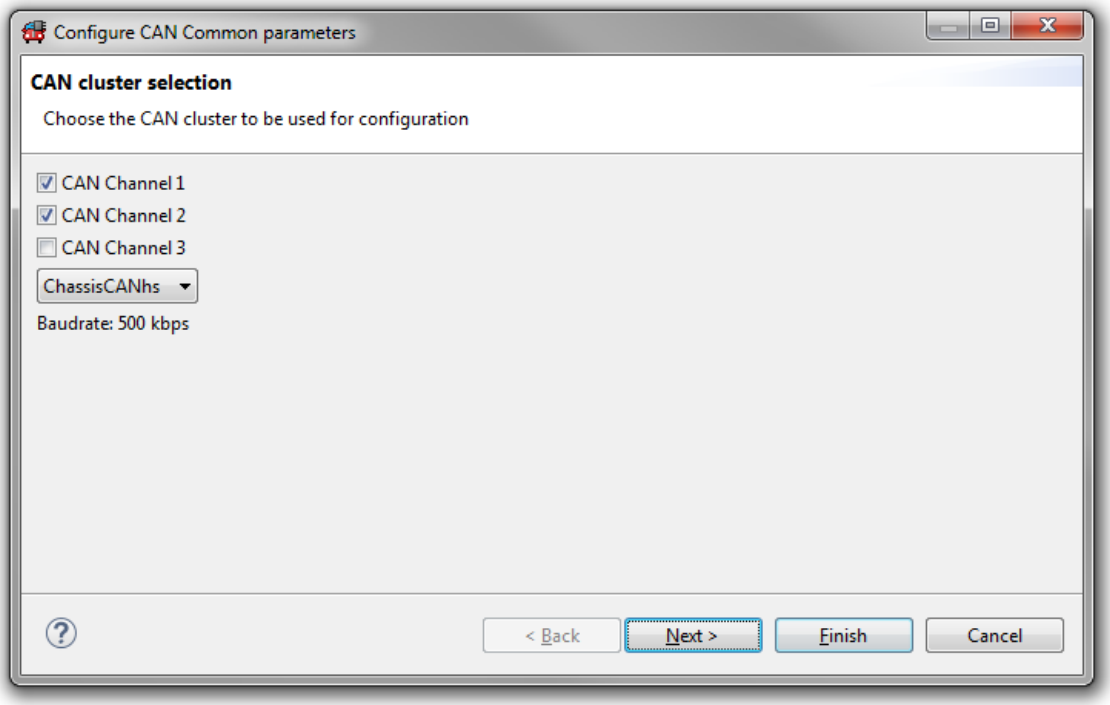
A varázsló segítségével az ECU Extract alapján kiválaszthatjuk a szimulációban valamint megfigyelésben részt vevő adatelemeket (üzeneteket, PDU-kat és szignálokat). A varázsló befejezése után, létrejönnek az alapvető konfigurációs containerek (`CommunicationConfiguration`, `CanConfiguration`, `E2EConfiguration` valamint `GuiConfiguration`).

A varázsló ezek mellett létrehoz egy `.gdl` fájlt is, mely a Restbus Szimulátor script nyelvén előre deklarálja a szimulációban felhasználható kereteket, és szignálokat. A felhasználó ezek után már megírhatja a szimulációs szkriptet, melyben hivatkozhat a generált fájlban deklarált elemekre. A Szimulátor szekvenciáit leíró `EventConfiguration` container a script feldolgozása után jön létre.

A konfiguráció összeállítása kézzel is lehetséges. Az AUTOSAR Architect biztosít egy űrlaphoz hasonló felületet az ECU Configuration modellek kezeléséhez, ennek kézzel való kitöltése azonban nagy mennyiségű adat esetén nehézkes.

3.1.1. A CAN hálózat kiválasztása

A varázsló első lapján állíthatók be a Gateway használni kívánt csatornái *(3.3. ábra)*. A varázsló kilistázza az ECU Extract által tartalmazott CAN clustereket, melyek közül a felhasználó kiválaszthatja, hogy melyikhez szeretne csatlakozni. (Mivel általában egy ECU egy CAN hálózathoz csatlakozik, az Extract is az esetek többségében csak egy cluster leírását tartalmazza. Előfordulhat azonban – például gateway feladatokat is ellátó vezérlőegységek esetén –, hogy az ECU Extract több clustert is tartalmaz.) A CAN cluster kiválasztásával a CAN hálózat fizikai paraméterei (például baudrate) ismertté válnak.



3.3. ábra. *A CAN csatornák beállítása*

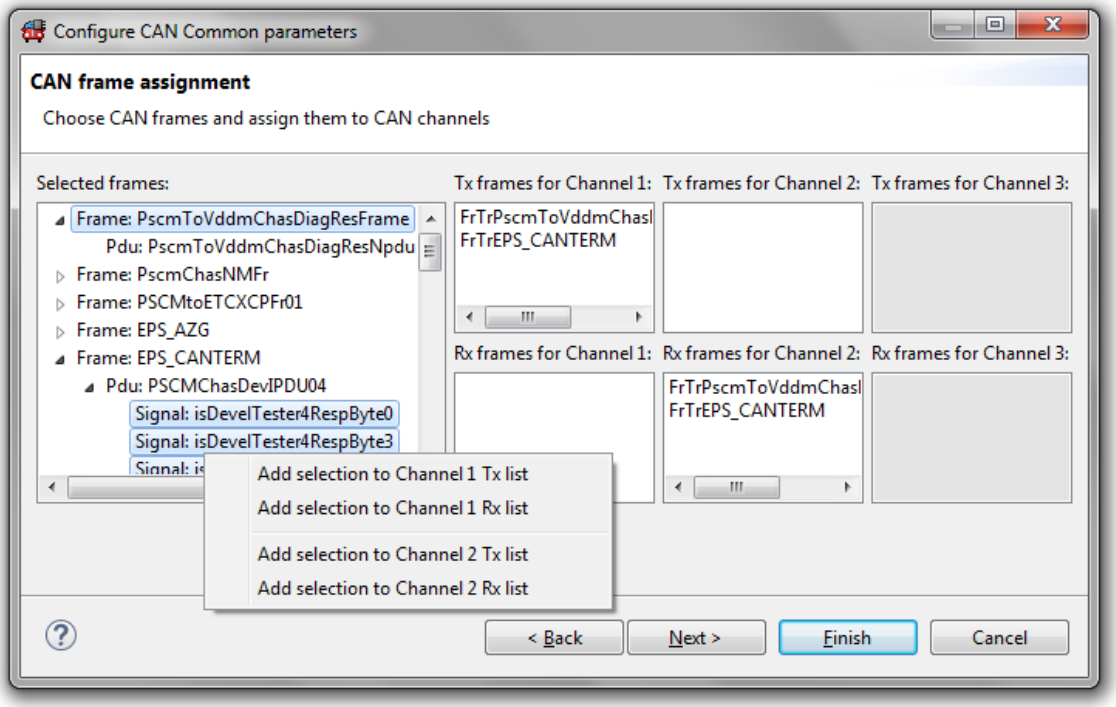
A bemutatott példában a Gateway 1-es és 2-es CAN csatornáját használom. A rendelkezésemre álló ECU Extract által tartalmazott egyetlen cluster a *ChassisCANhs* nevű cluster, mivel a kormányszervó vezérlőegysége – melyhez az Extract készült – erre a hálózatra csatlakozik. A kiválasztott hálózat adatsebessége 500 kbps.

3.1.2. Üzenetek kiválasztása

A következő lapon az ECU Extract által tartalmazott keretek, PDU-k és szignálok listájából választhatjuk ki azokat, melyeket a hálózaton küldeni (Restbus Szimulátor), illetve fogadni (Trace) szeretnénk *(3.4. ábra)*. Az Extract elemei fastruktúrába rendezve jelennek meg, így rendszerezve láthatók a keretek (Frame), az általuk tartalmazott PDU-k, valamint a PDU-k által tartalmazott szignálok.

A listában egyszerre több elem is kijelölhető, majd azokra jobb egérgombbal kattintva hozzárendelhetők az előző lapon kiválasztott csatornák valamelyikének küldési vagy fogadási listájához. Ha a felhasználó csak egy PDU néhány szignálját jelöli ki, a PDU és az ahhoz tartozó keret is hozzáadódik a konfigurációhoz.

A jobb oldalon a csatornákhoz rendelt keretek láthatók, melyek szintén jobb egérgombbal kattintva eltávolíthatók a listákból. Egy keret eltávolításával az összes általa tartalmazott PDU valamint szignál is eltávolításra kerül a konfigurációból.

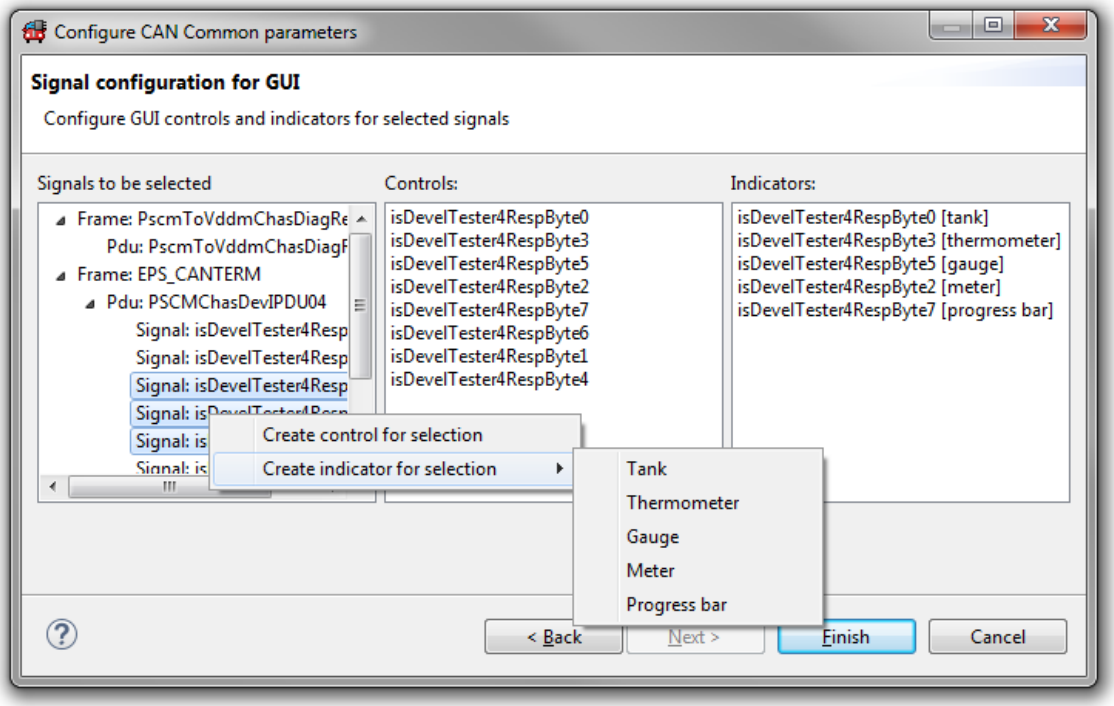


3.4. ábra. *Üzenetek kiválasztása*

Példámban két üzenet játszik szerepet. Egyik a diagnosztikai célokat szolgáló *PscmToVddmChasDiagResFrame*, másik pedig a szignálokat szállító *EPS_CANTERM* üzenet. A diagnosztikai keretben található PDU nem tartalmaz szignálokat, mivel az egy, a CAN Transport Layernek szóló N-PDU. A másik keretben található PDU azonban tartalmaz 8 darab 8 bites szignált. A két kiválasztott üzenet segítségével bemutathatom a Trace szignálokkal és a CAN Transport Protocol értelmezésével kapcsolatos funkcióit is.

3.1.3. A felhasználói felület beállításai

A varázsló utolsó lapján a külön indikátorokon megjelenítendő, valamint kézzel vezérelhető szignálokat adhatjuk meg *(3.5. ábra)*. A bal oldalon megjelennek az előző oldalon kiválasztott keretek melyek szignáljaiból válogathat a felhasználó. Jobb egérgombbal kattintva a kijelölt szignálokra azokhoz kontrollt, valamint különféle indikátorokat adhatunk. A kontrollok minden esetben csúszkaként jelennek meg később, az indikátorok kinézetét azonban öt típus közül választhatjuk ki *(3.6. ábra)*.



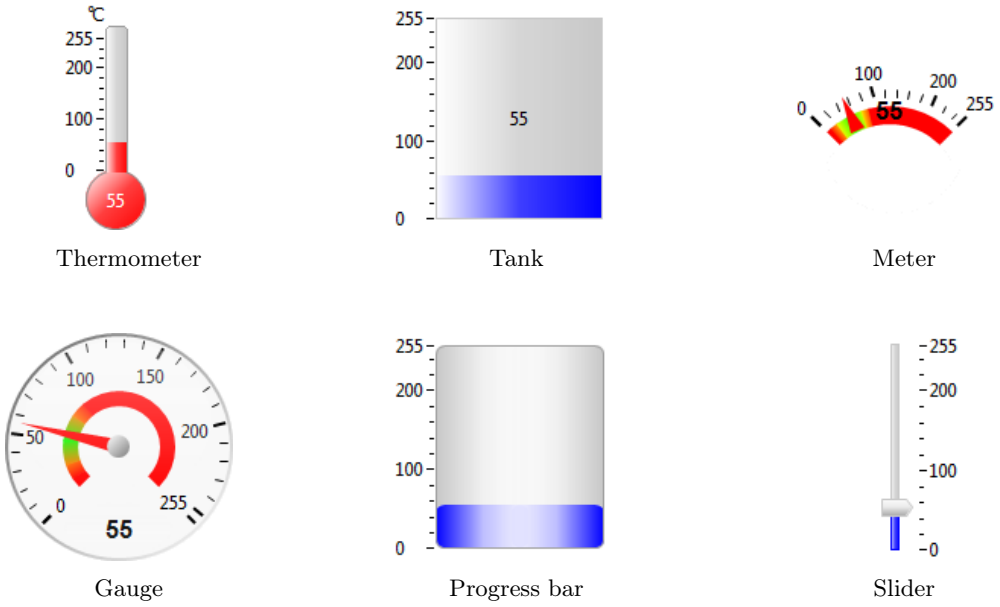
3.5. ábra. *Kontrollok és indikátorok beállítása*

3.1.4. A konfigurációs modell létrehozása a beállítások alapján

A varázsló a felhasználó által megadott adatokat a háttérben egy `CANConfigBuilderInfo` Java objektumban tárolja, mely a „Finish” gombra való kattintás után átadódik egy `CANConfiguBuilder` objektumnak, ami a kapott adatok alapján elvégzi a konfigurációs modell felépítését *(2.1. ábra)*.

A konfigurációs adatmodellek létrehozása a `CommunicationConfiguration` container felépítésével kezdődik. A generátor végigiterál a `builderInfo` által tartalmazott, ECU Extractból származó `ISignal`-okon, és létrehozza a nekik megfelelő `CommunicationCommon`-beli `Signal` elemeket. Ezután a `Pdu`-k és `PduSignal`-ok felépítése következik, melyekben már referálhatók az imént létrehozott szignálok. A `Frame` modellelemek létrehozása hasonló módon történik.

A következő lépésben a CAN specifikus konfigurációs containerek felépítése történik meg.



3.6. ábra. *GUI elemek*

Először a hivatkozott `Cluster` és engedélyezett Gateway csatornák alapján a `CanChannel` containerek jönnek létre, majd az összes kiválasztott (fogadni és küldeni kívánt) keret alapján a `CanFrame` objektumok. A `CanFrame`-ek már referálhatnak az előzőleg létrehozott `CommunicationConfiguration`-beli keretekre. Végül a kommunikáció irányát meghatározó, és a kereteket csatornákhoz rendelő `CanRxFrame` és `CanTxFrame` elemek jönnek létre, melyek hivatkoznak a létrehozott `CanChannel` és `CanFrame` containerekre.

A Trace szempontjából releváns utolsó lépés a `GuiConfiguration` létrehozása és kitöltése a `builderInfo` által tartalmazott adatokkal. Az `EventConfiguration` container generálása külön lépésben, a Restbus Szimulátor működését leíró szkript alapján történik.

3.1.5. A példában felhasznált szimulációs script

A varázsló a konfigurációs containerek létrehozása mellett generált egy script részletet, mely a szekvenciák leírásánál felhasználható szignálok és keretek deklarációit tartalmazza *(3.7. ábra)*:

```
/* Signals */
Signal isDevelTester4RespByte0
Signal isDevelTester4RespByte3
Signal isDevelTester4RespByte5
Signal isDevelTester4RespByte2
Signal isDevelTester4RespByte7
Signal isDevelTester4RespByte6
Signal isDevelTester4RespByte1
Signal isDevelTester4RespByte4

/* Tx Frames */
TxFrame FrTrPscmToVddmChasDiagResFrame_TX
TxFrame FrTrEPS_CANTERM_TX

/* Rx Frames */
RxFrame FrTrPscmToVddmChasDiagResFrame_RX
RxFrame FrTrEPS_CANTERM_RX
```

3.7. ábra. *Generált deklarációk*

Ezeket felhasználva megírtam a Trace bemutatásához szükséges egyszerű szimulációs scriptet *(3.8. ábra)*. A script belépési pontja a `run` blokk. A `run` blokk felfogható úgy, mint egy C nyelvű program `main` függvénye. Példámban a blokk két egyszerű utasítást tartalmaz, mellyel a szimuláció elején elindít két periodikusan végrehajtandó szekvenciát.

A `seq100` periodikus blokk 100 milliszekundumonként hajtódik végre. Minden periódusban az `EPS_CANTERM` üzenet első három szignálját növeli rendre 5-el, 10-el és 15-el, majd az üzenetet elküldi.

A diagnosztikai üzenet szimulálását megvalósító szekvencia feladata különböző, a CAN Transport Layer által értelmezhető üzenetek periodikus küldése. A blokk 15 másodpercenként hajtódik végre, és a protokoll által értelmezett összes kerettípusból küld egyet. Egy-egy keret elküldése között 2 másodpercet várakozik. A script a következő kerettípusokat szimulálja:

SingleFrame: Egy 7 bájt vagy annál rövidebb méretű üzenet egy keretben történő (nem szegmentált) átvitelére szolgál.

StartFrame: A 7 bájtot meghaladó méretű üzenetek szegmentált átvitelének első kerete.

ConsecutiveFrame: A 7 bájtot meghaladó méretű üzenetek további szegmensei.

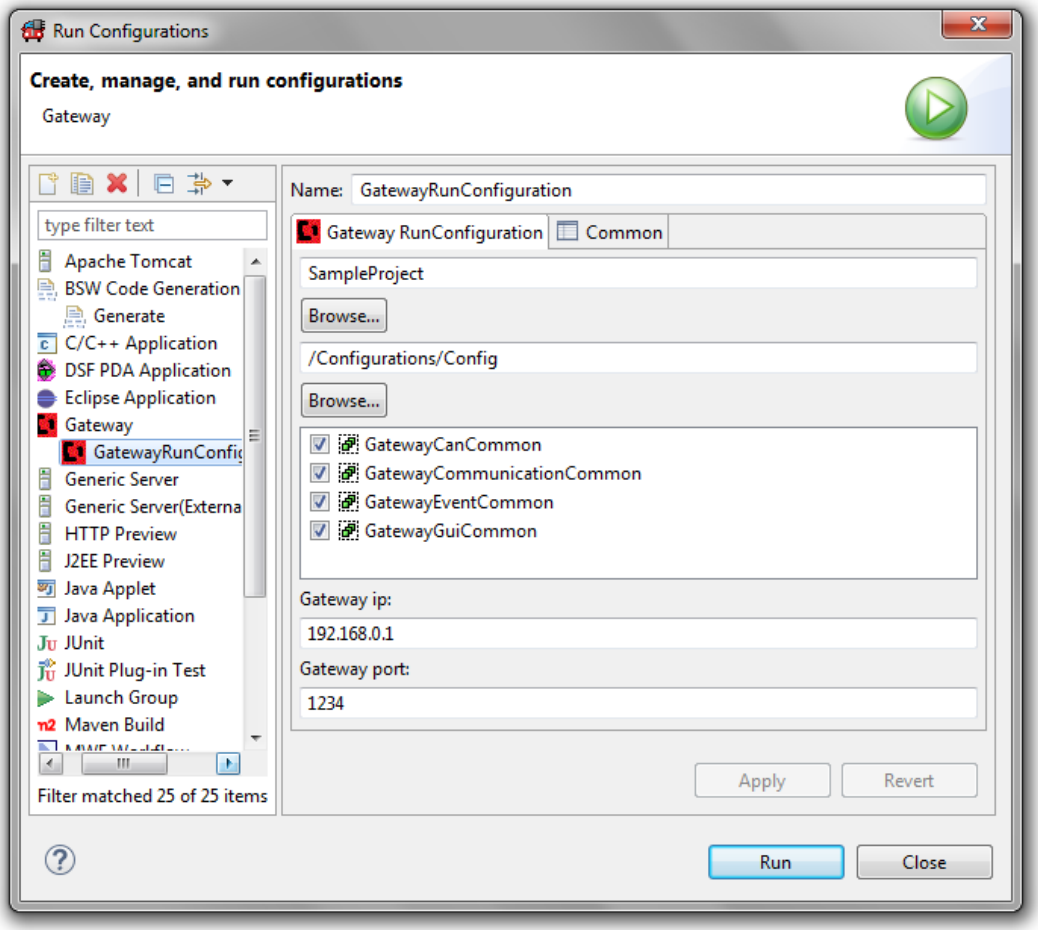
FlowControl-ContinueToSend, -Wait, -Overflow: A fogadó oldal felől érkező folyamat szabályozást megvalósító keretek. Segítségükkel az átvitel folytatása, szüneteltetése, valamint megszakítása kérhető.

```
/* Entry point of the simulation. */
run {

    /* Start needed sequences. */
    start(seq100)
    start(CanTp)
}

/* Sequence to demonstrate Signal processing functionality. */
Sequence seq100 (100) {

    /* Modify the value of the signals */
    set(isDevelTester4RespByte0 += 5)
    set(isDevelTester4RespByte1 += 10)
    set(isDevelTester4RespByte2 += 15)

    transmit(FrTrEPS_CANTERM_TX)
}
/* Sequence to demonstrate CAN Transport Layer functionality. */
Sequence CanTp(15000) {
    /*
     * Send SingleFrame
     * Length = 3 bytes
     * Data = {0x10, 0x20, 0x30}
     */
    transmit(FrTrPscmToVddmChasDiagResFrame_TX, "0x03 0x10 0x20 0x30 0x00 0x00 0x00 0x00")
    wait(2000)
    /*
     * Send FirstFrame
     * Total length = 50 bytes
     * Data = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06}
     */
    transmit(FrTrPscmToVddmChasDiagResFrame_TX, "0x10 0x32 0x01 0x02 0x03 0x04 0x05 0x06")
    wait(2000)
    /*
     * Send ConsecutiveFrame
     * Sequence Number = 4;
     * Data = {0x12, 0x55, 0x84, 0xFC, 0x3A, 0x32, 0x02}
     */
    transmit(FrTrPscmToVddmChasDiagResFrame_TX, "0x24 0x12 0x55 0x84 0xFC 0x3A 0x32 0x02")
    wait(2000)
    /*
     * Send FlowControl
     * Flow Status = ContinueToSend
     * Block Size = 16
     * STmin = 32
     */
    transmit(FrTrPscmToVddmChasDiagResFrame_TX, "0x30 0x10 0x20 0x00 0x00 0x00 0x00 0x00")
    wait(2000)
    /*
     * Send FlowControl
     * Flow Status = Wait
     */
    transmit(FrTrPscmToVddmChasDiagResFrame_TX, "0x31 0x00 0x00 0x00 0x00 0x00 0x00 0x00")
    wait(2000)
    /*
     * Send FlowControl
     * Flow Status = OverFlow
     */
    transmit(FrTrPscmToVddmChasDiagResFrame_TX, "0x32 0x00 0x00 0x00 0x00 0x00 0x00 0x00")
    wait(2000)
}
```

3.8. ábra. *A szimulációs script*

3.2. A felkonfigurált rendszer indítása

A script feldolgozása utána létrejött `EventConfiguration` containerrel már teljes a szoftvert leíró adatmodell. Az eszköz elindítására egy külön erre a célra definiált Eclipse Run Configuration felparaméterezése után van lehetőség *(3.9. ábra)*. A felhasználónak ki kell választania a konfigurációs modellt tartalmazó Eclipse projektet, betallóznia a megfelelő `EcucValueCollection`-t, majd kijelölni a futtatáshoz szükséges konfigurációt tartalmazó containereket. Ezek után a Gateway-hez való kapcsolódáshoz szükséges paramétereket kell megadni (IP cím és port), majd elindítható a Restbus Szimulátor és a Trace.



3.9. ábra. *Run Configuration*

3.3. A Trace által nyújtott funkciók

A Run Configuration futtatása után megnyílik a Restbus Szimulátor és Trace szolgáltatásait megjelenítő Eclipse View. A különböző funkciók a view-n belüli füleken érhetők el.

Common: Innen indíthatjuk el, valamint állíthatjuk le a szimulációt és a megfigyelést.

Restbus: A Restbus Szimulátor szekvenciáinak futását követhetjük nyomon.

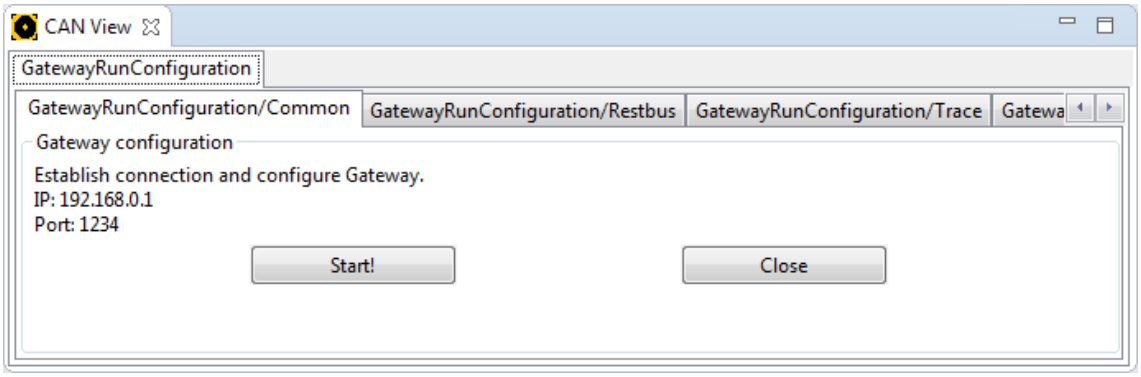
Trace: A Trace által rögzített legfrissebb eseményeket láthatjuk rajta.

TraceLog: A Trace által rögzített logot nézhetjük vissza, valamint menthetjük el.

Panel: A megjelenítendő és vezérelhető szignálok indikátorait és kontrolljait tartalmazza.

Stat: A megjelenítendő szignálok időbeli viselkedését követhetjük nyomon.

3.3.1. Common fül



3.10. ábra. *A Common fül*

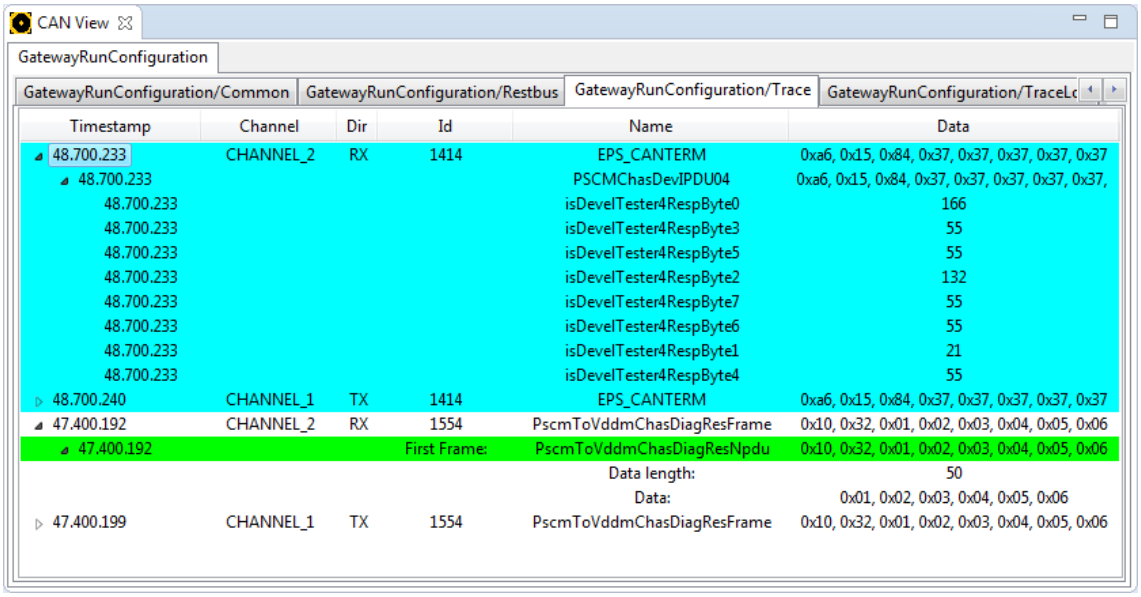
A *Start* gombra kattintva indítható a szimuláció és a megfigyelés. A gomb felirata ezután *Stop*-ra változik és megnyomásával a futás megállítható. A *Start* gomb megnyomása után a Gateway Manager kapcsolódik a Fieldbus Gateway-hez a megadott IP címen és porton, majd a Run Configuration felületen beállított adatmodellt bejárva felkonfigurálja a Gateway-t a kommunikáció megfelelő működtetéséhez.

Először létrehozza a megfelelő `CANRxMessage` és `CANTxMessage` objektumokat a driveren keresztül, majd elindítja a CAN csatornákat a beállított sebességgel. A bemutatott példa esetében két-két Rx és Tx üzenet jön létre, valamint elindul a Gateway CAN1 és CAN2 csatornája 500 kbps sebességgel.

A szoftver elindulásáról tájékoztatja a többi fület is, egyben átadva nekik a konfigurációt tartalmazó modellt. Ezek alapján a GUI konfigurációfüggő része is felépül (szignálok indikátorai és kontrolljai), és a funkciók aktívvá válnak.

A *Close* gomb megnyomásával minden lefoglalt kommunikációs erőforrás felszabadul és a teljes view bezáródik.

3.3.2. Trace fül



3.11. ábra. *A Trace fül*

Ezen a fülön kísérhetők figyelemmel az üzenetekhez kapcsolódó legutóbbi Rx illetve Tx események. A táblázatban fastruktúrába szervezve láthatók a keretek, PDU-k és szignálok. A legelső oszlopban az adott elem legutóbbi elküldéséhez vagy fogadásához tartozó időbélyeg látható. A következő oszlop a Gateway csatornájának nevét tartalmazza, melyen az adott elemet fogadtuk, vagy elküldtük. Ezek után következik a kommunikáció iránya, és az üzenet azonosítója. Az ötödik oszlop a keret, PDU vagy szignál nevét tartalmazza, míg a legutolsó oszlopban az adott elem által tartalmazott adatokat láthatjuk.

Érdemes megfigyelni, hogy az üzenetek 2-es csatornán történő fogadásához tartozó időbélyeg rendre 7 mikroszekundummal korábbi mint az 1-es csatornán való elküldésükhöz tartozó időbélyeg. Ennek a jelenségnek az oka a Gateway két csatornájának egy viszonylag rövid, körülbelül 20 centiméteres kábellel való összekötése. Ilyen távolságon a jelterjedési idő zérusnak tekinthető, így a Gateway-en futó szoftver CAN-hez tartozó Tx és Rx megszakítása gyakorlatilag egyszerre következik be. Az események időbélyeggel való ellátása még a Gateway-en megtörténik az eseményt jelző megszakítást kezelő függvényben, így elkerülhető az Ethernet és UDP kommunikáció, valamint a PC-n futó (nem real-time) operációs rendszer okozta késleltetések torzító hatásai. A két, egyszerre bekövetkező megszakítás közül az üzenetek fogadását végzőnek magasabb a prioritása, így előbb lesz kiszolgálva. Az elküldés sikerességét jelző megszakítás csak 7 mikroszekundummal később kerül sorra, ez okozza az állandó különbséget.

A fenti probléma kiküszöbölhető lenne a megszakítások prioritásának felcserélésével. Az üzenetek pontos fogadása azonban egy mérésre szolgáló eszköznél fontosabb szempont mint az elküldésükről szóló nyugta, ráadásul a gyakorlatban nem életszerű eset a Gateway csatornáinak direkt módon való összekötése, és saját kommunikációjának megfigyelése.

A felhasználót segítendő, a frissült adatokat tartalmazó sorok 1 másodperc erejéig világoskéken felvillannak, így rögtön látható mi az ami megváltozott. A példában szereplő *EPS_ CANTERM* üzenetet a script alapján 100 milliszekundumonként elküldjük a Gateway 1-es CAN csatornáján, így az folyamatosan világoskéken világít. Látható, hogy az első három szignál értéke eltér egymástól, míg a következő öt szignál értéke folyamatosan 55 (0x37). Ennek magyarázata, hogy a script csak az említett szignálok értékét változtatja, a többi szignál tartja a kezdeti, alapértelmezett értékét. (Az 55-ös alapértelmezett érték az ECU Extractból származik.)

Az ábrán a diagnosztikai üzenet éppen egy *FirstFrame*-et tartalmaz. Ezt zöld színnel jelöli a megjelenítő, valamint a keret dekódolása után kiírja annak további paramétereit. A scriptnek megfelelően a *FirstFrame* 50 bájt átvitelét ígéri, míg saját maga az üzenet első 6 bájtját tartalmazza.

A megjelenítés frissítése

A Trace fül megjelenítési funkcióinak hátterében egy időzítő (Java Timer) áll. Az időzítő tizedmásodpercenként indítja el a frissítést végző taszkot. Ebben az esetben a megjelenítés eseményektől való függetlenítése elengedhetetlen volt. Ha minden változást megjelenítünk a felületen, az nagy adatforgalom esetén jelentősen terheli a GUI-t, arról nem is beszélve, hogy nem hogy az emberi szem, de még a monitor képfrissítési sebessége sem képes követni ezen változásokat.

A frissítést végző taszk a Data Processor `getLastEvents()` metódusának meghívásával tizedmásodpercenként elkéri az adott üzenetekhez tartozó legutóbbi Rx valamint Tx eseményeket és frissíti a táblázat ezekhez tartozó sorait, valamint színüket világoskékre állítja. A taszk emellett számon tartja a sorokhoz tartozó legutóbbi frissítések időpontját, és ha szükséges (letelt az egy másodperc), a színüket visszaállítja fehérre.

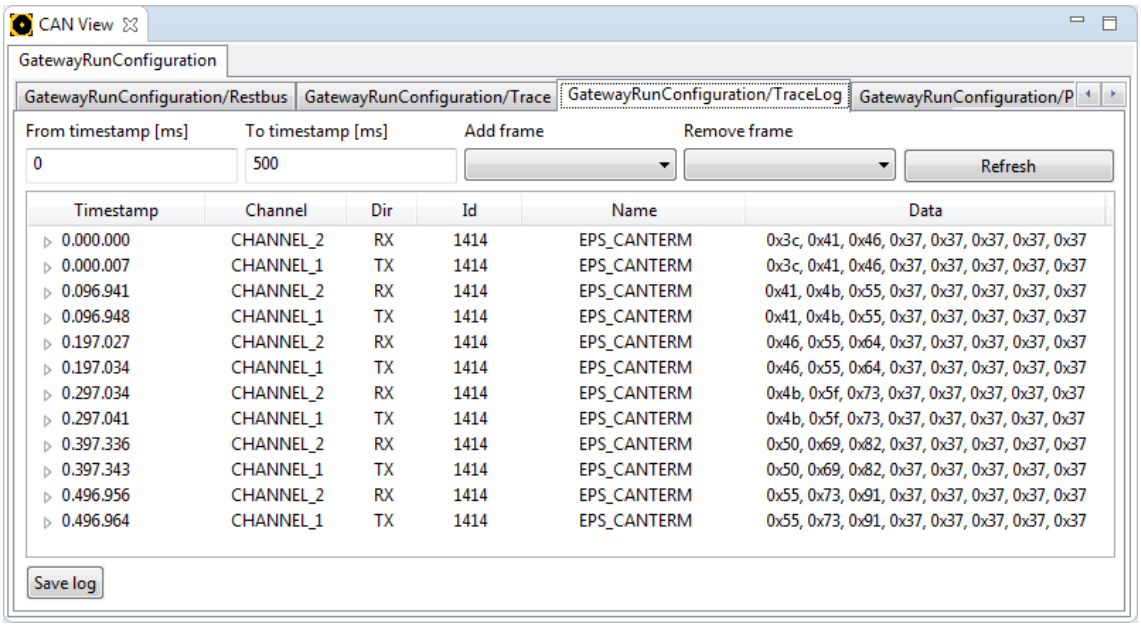
3.3.3. TraceLog fül

A *TraceLog* fül szolgál az elmúlt események elemzésére. Segítségével időtartományra és üzenetek listájára szűrve tekinthetjük meg a logot. Az események megjelenítése a *Trace* füllel azonos módon, táblázatos formában történik *(3.12. ábra)*.

A GUI felső sorában található két szövegdobozban az időbélyeg alsó és felső korlátját adhatjuk meg milliszekundumos felbontásban. Az *Add frame* legördülő menüből kiválaszthatjuk a szűrőhöz adandó kereteket, melyek ezek után az *Add frame* menüből eltűnnek, és megjelennek a *Remove frame* menüben. A *Remove frame* menüből kiválasztva a szűrőfeltételből törölhetők a keretek. A szűrés a *Refresh* gombra kattintva végezhető el.

A gomb megnyomásával a táblázat eddigi tartalma törlődik, és a GUI a Data Processor `getFilteredEvents(...)` metódusának segítségével lekérdezi a feltételeknek megfelelő események listáját, majd azok alapján feltölti a táblázatot.

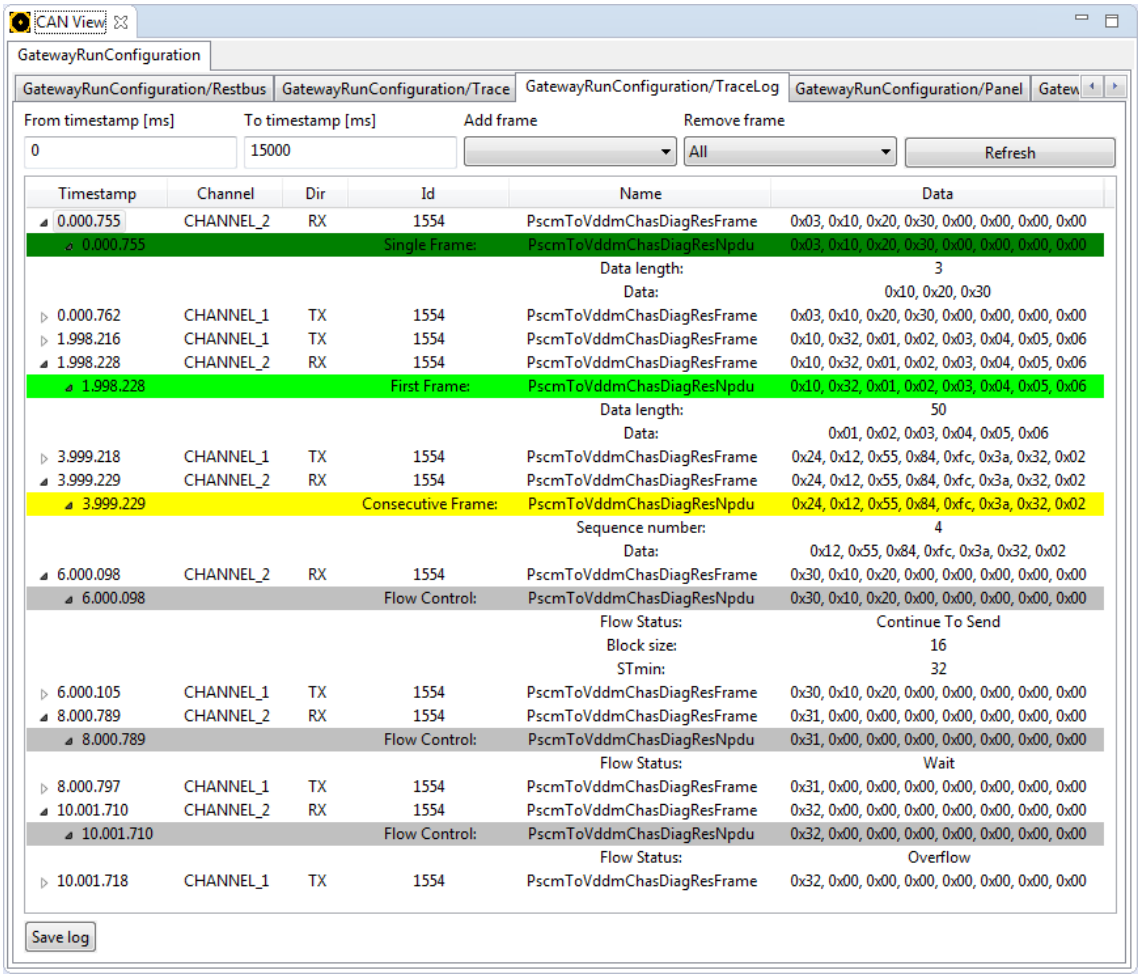
Szignálok változása



3.12. ábra. *A szignálok változása a TraceLog fülön*

Az ábrán nyomon követhető a szignálok scriptben megírt változása. A szűrést az első 500 milliszekundumra és az *EPS_CANTERM* keretre végeztem el. Az üzenet első bájtjában található szignál minden 100 milliszekundumban 5-el a második 10-el, míg a harmadik 15-el nő. (Továbbra is megfigyelhető a Tx és Rx események előző fejezetben leírt felcserélődése. A köztük lévő különbség továbbra is pontosan 7 mikroszekundum.)

A CAN Transport Layer üzenetei



3.13. ábra. *A CAN Transport Layer üzenetei a TraceLog fülön*

A CAN Transport Layer által megvalósított protokoll üzenetei a könnyebb átláthatóság érdekében ebben a nézetben is színezve jelennek meg. A színek segítségével könnyedén követhető és elemezhető egy-egy átviteli szekvencia. A példában minden kerettípusból megjelenik egy-egy. Az üzenetek a scriptnek megfelelően két másodpercenként követik egymást *(3.13. ábra)*.

A keretek első bájtjának felső négy bitje határozza meg a keret típusát. A keret további bájtjai az adott kerettípustól függően eltérő jelentéssel bírnak.

Single Frame: A keret megjelenítési színe sötétzöld. Az üzenet első bájtjának alsó négy bitje a keretben elküldött bájtok számát adja meg. Ebben az esetben a 0x10, 0x20 és 0x30 bájtokat küldtük, tehát a megjelenítő helyesen dekódolta az üzenet tartalmát.

First Frame: A keret megjelenítési színe világoszöld. A teljes átvivendő üzenet hosszát a keret első bájtjának alsó négy bitje, valamint második bájtja együttesen tartalmazza. Ebben az esetben ez 0x32, vagyis decimális 50. A keret fennmaradó bájtjai az üzenet első szakaszát tartalmazzák.

Consecutive Frame: A keret megjelenítési színe sárga. A keret sorszámát (*Sequence Number*) első bájtjának alsó négy bitje adja meg. Ebben az esetben ez az érték 4. A fennmaradó bájtok az üzenet egy darabját tartalmazzák.

Flow Control - Continue To Send: A *Flow Control* keretek megjelenítési színe szürke. A keret első bájtjának alsó négy bitje határozza meg a *Flow Status* paramétert, melynek 0 értéke az átvitel folytatását kéri a másik féltől. Az üzenet második bájtja a következő blokk méretét határozza meg. Példánkban ez 0x10, vagyis valóban decimális 16. Az üzenet utolsó értékes bájtja a harmadik, mely a keretek küldése között kivárandó időt határozza meg. Esetünkben ez 32. A bájt értéke és az időzítés valódi mértéke közötti átváltást a szabványban található táblázat definiálja [8].

Flow Control - Wait: Ebben az esetben az első bájt alsó négy bitje 1 értékű. Az üzenet az átvitel ideiglenes felfüggesztésére kéri az adó felet. Más paramétert az üzenet nem tartalmaz.

Flow Control - Overflow: Ebben az esetben az első bájt alsó négy bitje 2 értékű. Az üzenet az átvitel megszakítására kéri az adó felet buffer túlcsordulás miatt. Más paramétert az üzenet nem tartalmaz.

A log fájlba mentése

A megjelenített log a *Save log* gombra kattintva a – későbbi feldolgozhatóság érdekében – `.csv` fájlba menthető. A gomb megnyomása után egy szabványos párbeszédablakon keresztül megadható a fájl elérési útja, majd a program a táblázat sorain végigmenve, legenerálja a kimeneti fájlt.

A logban szerepelnek a pontos időbélyegek, az üzenetek azonosítója és neve, valamint adattartalma, így az elmentett adatok alapján a kommunikációs folyamatok maradéktalanul reprodukálhatók.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0.000.755	CHANNEL_2	RX	1554	PscmToVddmChasDiagResFrame	0x03	0x10	0x20	0x30	0x00	0x00	0x00	0x00
2	0.000.762	CHANNEL_1	TX	1554	PscmToVddmChasDiagResFrame	0x03	0x10	0x20	0x30	0x00	0x00	0x00	0x00
3	1.998.216	CHANNEL_1	TX	1554	PscmToVddmChasDiagResFrame	0x10	0x32	0x01	0x02	0x03	0x04	0x05	0x06
4	1.998.228	CHANNEL_2	RX	1554	PscmToVddmChasDiagResFrame	0x10	0x32	0x01	0x02	0x03	0x04	0x05	0x06
5	3.999.218	CHANNEL_1	TX	1554	PscmToVddmChasDiagResFrame	0x24	0x12	0x55	0x84	0xfc	0x3a	0x32	0x02
6	3.999.229	CHANNEL_2	RX	1554	PscmToVddmChasDiagResFrame	0x24	0x12	0x55	0x84	0xfc	0x3a	0x32	0x02
7	6.000.098	CHANNEL_2	RX	1554	PscmToVddmChasDiagResFrame	0x30	0x10	0x20	0x00	0x00	0x00	0x00	0x00
8	6.000.105	CHANNEL_1	TX	1554	PscmToVddmChasDiagResFrame	0x30	0x10	0x20	0x00	0x00	0x00	0x00	0x00
9	8.000.789	CHANNEL_2	RX	1554	PscmToVddmChasDiagResFrame	0x31	0x00	0x00	0x00	0x00	0x00	0x00	0x00
10	8.000.797	CHANNEL_1	TX	1554	PscmToVddmChasDiagResFrame	0x31	0x00	0x00	0x00	0x00	0x00	0x00	0x00
11	10.001.710	CHANNEL_2	RX	1554	PscmToVddmChasDiagResFrame	0x32	0x00	0x00	0x00	0x00	0x00	0x00	0x00
12	10.001.718	CHANNEL_1	TX	1554	PscmToVddmChasDiagResFrame	0x32	0x00	0x00	0x00	0x00	0x00	0x00	0x00

3.14. ábra. *Az elmentett log*

3.3.4. Panel fül

3.15. ábra. *A Panel fül*

Ez a fül szolgál a konfiguráció során beállított szignálok futásidőben történő nyomon követésére, valamint kézi vezérlésére. A GUI ezen elemének elrendezése a *Common* fül *Start* gombjának megnyomása után a modell feldolgozása során jön létre. A program először az indikátorokat, majd a kontrollokat jeleníti meg típusaik szerint csoportosítva.

A GUI ezen elemei a Nebula Project által fejlesztett speciális SWT widgetek *(3.6. ábra)*. A példában a *EPS_CANTERM* üzenet első négy szignálját állítottam be megjelenítésre. A szignálokhoz a legszemléletesebb *Gauge* stílusú indikátort választottam.

A csúszkák a szimulációban résztvevő szignálok futásidőben történő vezérlésére szolgálnak. Az indikátorokkal egy panelon megjelenítve lehetőség van a szignálok változtatásával egy időben, a változások hatásainak megfigyelésére is. A konfigurációban az üzenet minden szignáljához felvettem kontrollt, így az ábrán 8 csúszka látható.

Az GUI elemek és a szignálok kapcsolata

A nagy adatforgalom esetén jelentkező gyors változások a *Trace* fülhöz hasonlóan itt is gondot jelentenek. A megjelenítés frissítése ebben az esetben sem egyesével, az adott szignálok változásával egy időben történik, hanem a *Trace* fülhöz hasonlóan egy tizedmásodpercen-

ként futtatott taszk végzi el. A taszk az összes megjelenítendő szignálhoz tartozó indikátort és kontrollt frissíti a jel aktuális értékének megfelelően.

A kontrollok frissítésére azért van szükség, mert nem kizárt, hogy egy, a Trace felhasználói felületére kontrollként felvett szignál értékét a háttérben a Restbus Szimulátor scriptje megváltoztatja. Ebben az esetben viszont előfordulhat az is, hogy miközben a felhasználó a csúszkát állítja a script futása közben, az a tizedmásodpercenkénti frissítés következtében folyamatosan el-el ugrik az adott pozícióról. Ennek kiküszöbölésére azon GUI elemek állapotát melyeken éppen a fókusz van, a taszk nem frissíti.

A kontrollok és indikátorok szignálokkal való összeköttetését Java Mapek valósítják meg, melyek a GUI felépülésével együtt jönnek létre a szoftver indulásakor a modellek feldolgozása során.

Az elemek elrendezése

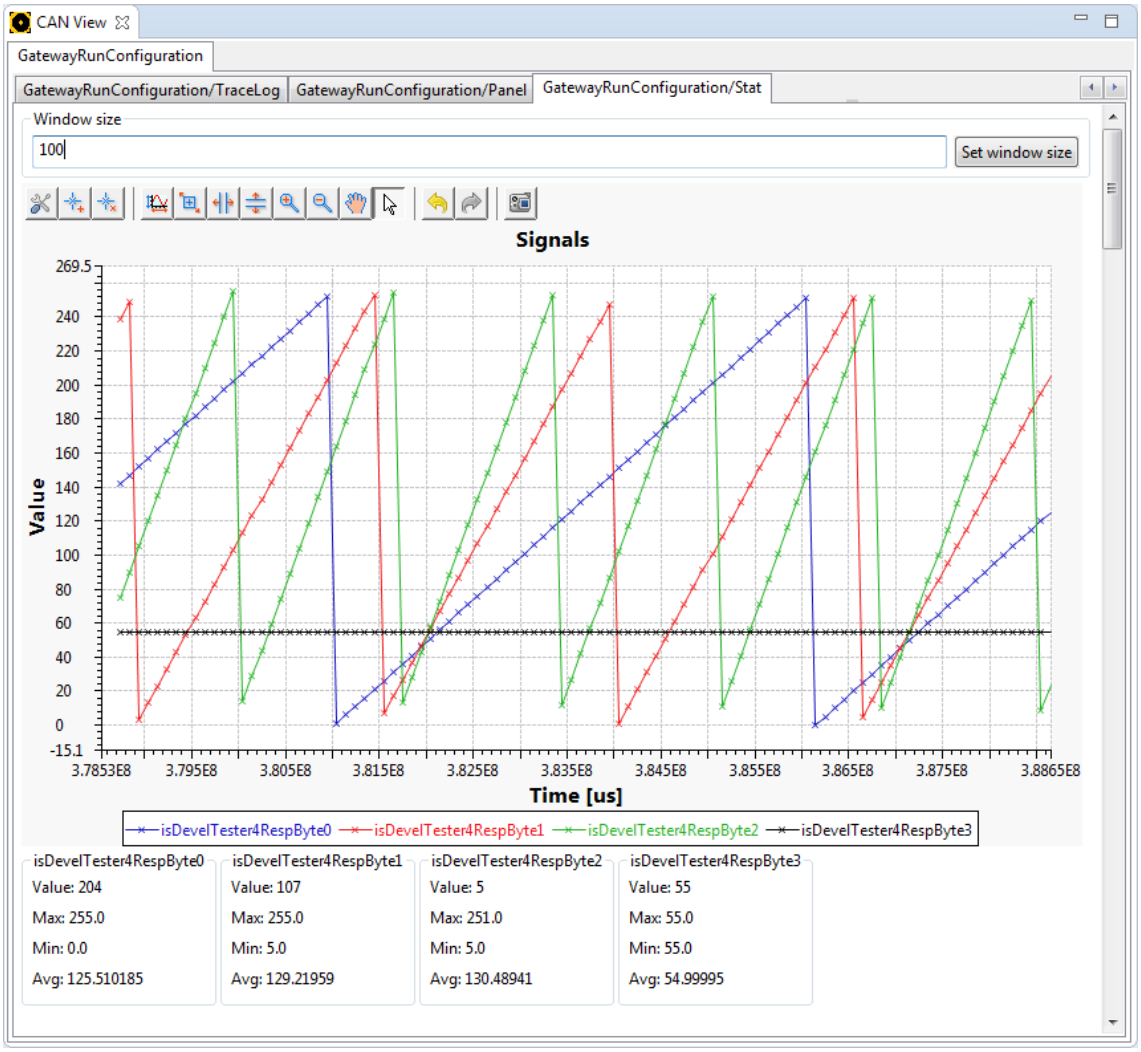
Jelenleg a felületen található elemeket a szoftver a modell szerinti sorrendben, típusonként csoportosítva helyezi el. Az ablak átméretezésével a layout is változik, azonban ergonómiai szempontból előnyösebb lenne, ha a felhasználó maga tudná meghatározni az elemek pozícióját és méretét.

Erre két megoldást találtam ki, melyek implementálása a jövőben megfontolandó lehet. Az egyik elképzelés szerint a megjelenítendő elemek pozícióját és méretét konfigurációs időben, a varázsló segítségével lehetne meghatározni. Ezt egy sakktáblához hasonló felület támogatná, melynek különböző tartományaihoz lehetne hozzá rendelni az egyes indikátorokat és kontrollokat. Ennek a megoldásnak az előnye, hogy viszonylag egyszerűen implementálható, azonban az elemek mérete és pozíciója kvantálttá válik.

A másik, rugalmasabb elképzelés szerint az elemek méretét és pozícióját a felhasználó futás közben – például jobb egérgombbal kattintva, majd az elemeket a megfelelő helyre húzva – határozhatná meg. Ennek a megoldásnak a kivitelezése bonyolultabb, ellenben nagyobb szabadságot enged a felhasználónak.

Mindkét esetben szükséges a `GuiConfiguration` container kiegészítése az elemek koordinátáinak leírásához szükséges paraméterekkel.

3.3.5. Stat fül



3.16. ábra. *A Stat fül*

A szignálok időbeli viselkedésének szemléletes megjelenítése a *Stat* fül funkcióinak segítségével lehetséges. Ebben a nézetben grafikonon követhetjük nyomon a megjelenítésre kiválasztott szignálok értékének változását, valamint a hozzájuk tartozó statisztikai paramétereket *(3.16. ábra)*.

A szignálok vizualizációja

A megjelenítésre használt grafikon widget szintén a Nebula Projectnek köszönhető. Számos hasznos funkcióval bír, melyek az ábrán látható eszköztár segítségével érhetők el. A megjelenített ábra nagyítható, kicsinyíthető, valamint annak kiválasztott pontjai címkékkel láthatók el. A grafikon címe, tengelyeinek elnevezése, valamint a megjelenített szignálok stílusa (vonal típusa, színe, pontok stílusa) szintén testre szabhatók. A grafikon igény esetén `.png` fájlba menthető.

A GUI ezen nézetét vezérlő osztály megvalósítja az `IDataListener` interfészt, és a program indulásakor feliratkozik a Data Processor értesítéseire. Innentől kezdve minden esetben amikor egy szignál értéke a rendszerben megváltozik a `signalUpdated(...)` metóduson keresztül a GUI értesül a változásról. A metódusban elkéri a szignál aktuális értékét, és az időbélyeggel együtt felvesz egy új pontot a grafikonon.

Az egyszerre megjelenítendő minták száma a grafikon felett található *Window size* szövegdobozba írt számmal határozható meg.

Az ábrán a példa script által megvalósított működés jól nyomon követhető. A keret legelső szignáljának (kék) értéke 100 milliszekundumonként 5-el nő. A második szignál (piros) értéke 10-esével változik, ez látszik a kék szignálhoz viszonyított meredekségéből is. Leggyorsabban a harmadik (zöld) szignál értéke változik 15-ös léptékben. A zöld jel ennek megfelelően a piros egy periódusa alatt három periódust ír le.

Statisztikai paraméterek megjelenítése

A statisztikai paraméterek a grafikon alatt elhelyezkedő blokkokban láthatók. Egy blokk egy szignál különböző paramétereit tartalmazza:

- Aktuális érték
- A megfigyelési ablakban felvett maximum
- A megfigyelési ablakban felvett minimum
- A megfigyelési ablakon számított átlag

A megfigyelési ablak méretét szintén a *Window size* szövegdobozban megadott érték határozza meg, így a paraméterek gyakorlatilag a grafikonon is éppen megjelenő minták alapján számolódnak.

A GUI paraméterek kijelzésére szolgáló elemei a szoftver indulásakor, a modell feldolgozása során jelennek meg. Ezek után a konfigurációban megadott szignálokhoz kötődően létrejönnek a `SignalStatistics` osztály megfelelő példányai *(2.12. ábra)* is.

Egy szignál értékének frissülése esetén (`signalUpdated(...)` az új érték be lesz léptetve az adott szignálhoz tartozó statisztikákat kezelő `SignalStatistics` objektumba, annak `addValue(long)` metódusán keresztül. Ennek hatására a paraméterek számolása folyamatosan, futásidőben történik. A GUI-n kijelzett értékek azonban a már előzőekben ismertetett okokból, ezen a felületen is csak tizedmásodpercenként frissülnek.

Összefoglalás

A Diplomatervezés feladatom során megvalósított szoftver egy összesen két éves munkafolyamat befejező állomása. Az MSc képzés Önálló laboratórium tárgyainak keretein belül elkészítettem egy autóipari kommunikációs átjáró beágyazott, C nyelven írt programkódját, valamint a hozzá kapcsolódó számítógépen futó, az eszköz alacsony szintű szolgáltatásainak elérését lehetővé tevő Java nyelvű drivert és API-t. A dolgozatomban bemutatott szoftver ezen eszköz szolgáltatásaira építve, azt kiegészítve egy, az autóipari buszokon zajló kommunikáció megfigyelését lehetővé tevő teljes értékű eszközt kínál a felhasználó számára.

Feladatom megvalósítását a szoftver leendő felhasználóival történő egyeztetéssel kezdtem. Lefektettük és rangsoroltuk a követelményeket aszerint, hogy a jelenleg általuk használt szoftvereknek milyen funkcióit mennyire tartják hasznosnak mindennapi munkájuk során. Ezek közül az elmúlt két félévben a feladatkiírásban szereplő pontokat mind sikeresen teljesítettem.

A megvalósítás a vállalatnál fejlesztett Eclipse alapú modellező eszközbe épül. Működésének hátterében a konfigurációját leíró adatmodellek állnak, melyeknek segítségével a felhasználó megfogalmazhatja a szoftvertől aktuálisan elvárt működést. A folyamatban, egy varázsló is segítségére van, mely képes szabványos AUTOSAR modellek feldolgozására, és azok alapján a Trace szoftver konfigurációjának előállítására.

A szoftver megjelenítéssel kapcsolatos funkciói támogatják a kommunikáció üzenet, PDU és szignál szintű elemzését is. A kiválasztott szignálokhoz a felhasználói felületen külön elhelyezkedő indikátorok rendelhetők, így a megfigyelés szempontjából fontosnak tartott jelek az ember által gyorsabban értelmezhető formában jeleníthetők meg. A megfigyelési időszakban lezajlott események naplója későbbi – feldolgozás céljából – CSV fájlba menthető. Ezek mellett a szoftver képes szabványos autóipari szállítási rétegbeli protokollok üzeneteinek értelmezésére, és azoknak áttekinthető formában való megjelenítésére is.

A jelek értékének időbeni változása grafikonon is nyomon követhető, valamint azokkal kapcsolatosan különböző statisztikai paraméterek számíthatók. A paraméterek számításánál, valamint a grafikonon való megjelenítésnél alkalmazott ablakméret futásidőben változtatható.

A Trace szoftver konfigurációjában és megvalósításában kölcsönösen együttműködik a szintén az elmúlt év során kifejlesztett Restbus szimulációs programmal.

Feladatom megvalósítása során számos Eclipse alapú, valamint modellezéssel kapcsolatos technológia használatába nyertem betekintést. Megismertem az Eclipse keretrendszer

felépítését, és elvégeztem több a keretrendszerre épülő plugin fejlesztését. A modellek feldolgozásának során az AUTOSAR Architect, valamint az Eclipse Modeling Framework szolgáltatásaira támaszkodtam.

Az elkészített szoftvert a vállalatnál máris többen használják, és folyamatosan merülnek fel új funkciókkal kapcsolatos igények. A jelenleg megfigyelési és szimulációs célokra felhasználható program a jövőben a fent leírt funkciók mellett számos szolgáltatással bővülhet. Tervben van az eszköz kiegészítése egy autóipari kalibrációs protokollokat (CCP, XCP) támogató felülettel, melynek segítségével az eszköz kalibrációs és szoftveres mérési célokra is használható.

Előnyös lehet továbbá egy diagnosztikai funkciókat megvalósító plugin kifejlesztése is, mely szabványos ODX leírók feldolgozásával képes lenne a vezérlőegységek diagnosztikai szolgáltatásainak elérésére. Az eszköz segítségével így az ECU-n tárolt hibakódok kiolvasása, vagy akár annak felprogramozása is lehetővé válna.

A modellalapú megközelítés, valamint a szoftver implementációjának moduláris felépítése lehetővé teszi a jövőbeni bővítések zökkenőmentes integrációját a meglévő implementációba.

Ábrák jegyzéke

Rövidítések és kifejezések jegyzéke

API	Application Programming Interface - egy program azon metódusai és szolgáltatásai, melyet más programok felhasználhatnak
BSW	Basic Software Layer – alapszoftver réteg
ECU	Electronic Control Unit – elektronikus vezérlőegység
EMF	Eclipse Modeling Framework - Eclipse modellező keretrendszer
GUI	Graphical User Interface - grafikus felhasználói felület
I-PDU	Interaction Layer Protocol Data Unit - az interakciós réteg kommunikációs adategységei
PCI	Protocol Control Information - protokoll vezérlő információ
PDU	Protocol Data Unit - protokoll adategység
RTE	Runtime Environment – futtató környezet
SW-C	Software Component – AUTOSAR szoftverkomponens
SWT	Standard Widget Toolkit - az Eclipse alkalmazások grafikus felhasználói felületének létrehozására szolgáló komponensek gyűjteménye
VFB	Virtual Function Bus – virtuális függvénybusz

Irodalomjegyzék

[1] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[2] Erich Gamman and Kent Beck. *Contributing to Eclipse: Principles, Patterns, and Plugins*. Addison-Wesley, 2003.

[3] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. Emf: Eclipse modeling framework (2nd edition), 2008.

[4] AUTOSAR Consortium. *Software Component Template R4.0 Rev 3*, 2012. `http://www.autosar.org/fileadmin/files/releases/4-0/methodology-templates/templates/standard/AUTOSAR_TPS_SoftwareComponentTemplate.pdf`.

[5] AUTOSAR Consortium. *System Template R4.0 Rev 3*, 2012. `http://www.autosar.org/fileadmin/files/releases/4-0/methodology-templates/templates/standard/AUTOSAR_TPS_SystemTemplate.pdf`.

[6] AUTOSAR Consortium. *Specification of ECU Configuration R4.0 Rev 3*, 2012. `http://www.autosar.org/fileadmin/files/releases/4-0/methodology-templates/templates/standard/AUTOSAR_TPS_ECUConfiguration.pdf`.

[7] International Organization for Standardization. *ISO 10681-2 Road vehicles – Communication on FlexRay – Part 2: Communication layer services*, first edition, június 2010.

[8] International Organization for Standardization. *ISO 15765-2 Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Communication layer services*, second edition, november 2011.

[9] BME MIT Tanszéki Munkaközösség. Digitális jelfeldolgozás, szeptember 2008.