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

Matlab-Simulink rendszerek modell alapú validációja

Szakdolgozat

Készítette

Búr Márton

Konzulensek

dr. Horváth Ákos
tudományos munkatárs

Hegedüs Ábel
tudományos segédmunkatárs

2013. december 13.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Búr Márton*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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, 2013. december 13.

Búr Márton
hallgató

Kivonat

A beágyazott, biztonságkritikus rendszerek fejlesztése során kiemelt fontosságú a tervezett rendszer működésének tervezés idejű ellenőrzése. Ezek a rendszerek általában a környezetükkel interakcióba lépve végzik feladataikat, amely beavatkozók irányítását jelenti az érzékelők által végzett megfigyelések alapján. A tervezési idejű ellenőrzés során a környezet szimulációjával adatokat szolgáltatunk az érzékelőknek a beavatkozók kimenetei és előre meghatározott környezeti adatok alapján, eközben figyeljük a rendszer belső állapotát és működését.

A Matlab-Simulink a beágyazott, biztonságkritikus rendszerek fejlesztésében széles körben elterjedt modellezési és szimulációs eszköz. A Simulink segítségével lehetőség van komplex rendszerek hierarchikus megvalósítására és a rendszer komponenseinek szimulációjára. Azonban a Simulink általános modellezési megközelítése miatt nehézkes ellenőrizni, hogy az elkészített modellben található komponensek megfelelnek-e a tervezett rendszer felépítésére előírt strukturális kényszereknek. Ezen kényszerek gyakran gráf jellegű megkötéseket írnak elő a tervezendő rendszer architektúrájára, amiket nehézkes átláthatóan, imperatív módon specifikálni.

A szakdolgozat keretében egy olyan módszert valósítottunk meg, amelyben a Simulink modellekre vonatkozó rendszervalidációs kényszerek deklaratív módon definiálhatók, és ezen kényszerek teljesülése tetszőleges Simulink modellen ellenőrizhető. A megoldásunk két lehetőséget ajánl ennek megvalósítására.

Egyik módszer, hogy a Simulink modelleket átalakítjuk az Eclipse Modeling Framework által használt reprezentációra, majd az EMF-IncQuery modell lekérdező eszköz segítségével végezzük el a validációt. A modell reprezentáció hatékonyságának és a lekérdező eszköz inkrementális működésének köszönhetően a módszer képes akár százezres méretű Simulink modellekre is skálázódni. A kialakított eszköz képes a modell lekérdező által szolgáltatott eredményeket a Simulink felhasználói felületén megjeleníteni.

A másik választás a deklaratív szabálydefiníciókból ellenőrző imperatív Matlab függvények generálása. Erre a célra az úgynevezett local-search alapú keresési technikák módszertanát használtuk fel, ellentétben a dolgozatban bemutatott másik módszer inkrementális megközelítésével. Ekkor, mindössze a szabályok megadását leszámítva, a teljes ellenőrzési folyamat a generált kód felhasználásával már közvetlenül a Simulink modelleken Matlab segítségével történik.

Abstract

The verification of the system behavior is essential during the early phases of the development of embedded, safety-critical systems. Such systems usually interact with their environment during their operation by controlling actuators based on observations and measurements of sensors. The design-time verification often includes the simulation of the environment by providing data to sensors based on the output of actuators and predefined environmental information, while also monitoring the internal state and behavior of the system.

MATLAB-SIMULINK is a modeling and simulation tool that is popular and wide-spread in the development of embedded, safety-critical systems. SIMULINK supports the hierarchical implementation of complex systems and the simulation of the system components. Unfortunately, due to the generic modeling approach of SIMULINK, it is difficult to validate that the components in the prepared model conform to the structural constraints defined for the designed system. These constraints often describe graph-based restrictions on the architecture of the developed system which are hard to specify with a clear, imperative definition.

We have developed an approach for the validation of system constraints on SIMULINK models, where constraints are defined declaratively and arbitrary SIMULINK models can be checked for violations. Our solution offers two separate ways to carry out the validation of a model.

One of the methods first transforms SIMULINK models to the model representation of the Eclipse Modeling Framework then the validation is performed using the EMF-INCQUERY model query engine. Based on the incremental evaluation techniques and the EMF model representation provided by the EMF-INCQUERY, the approach can scale up to SIMULINK models with hundreds of thousands of elements. Our technique can automatically annotate the results provided by the model query engine back to the SIMULINK models, thus the corresponding, constraint-violating parts of the model can be properly identified.

The other offered method generates imperative MATLAB validation functions using the declarative description of model constraints. For code generation, the steps of local-search based techniques are applied, instead of an incremental approach used by the other introduced validation method. In this case only the constraint definition is done outside the MATLAB environment, all other steps of the process are carried out directly using models in SIMULINK.

1. fejezet

Bevezetés

A beágyazott, biztonságkritikus rendszerek fejlesztése során kiemelt fontosságú a tervezett rendszer működésének tervezés idejű ellenőrzése. Ezek a rendszerek általában a környezetükkel interakcióba lépve végzik feladataikat, amely beavatkozók irányítását jelenti az érzékelők által végzett megfigyelések alapján. A tervezési idejű ellenőrzés során a környezet szimulációjával adatokat szolgáltatunk az érzékelőknek a beavatkozók kimenetei és előre meghatározott környezeti adatok alapján, eközben figyeljük a rendszer belső állapotát és működését.

Motiváció. A Matlab-Simulink a beágyazott, biztonságkritikus rendszerek fejlesztésében széles körben elterjedt modellezési és szimulációs eszköz. A Simulink segítségével lehetőség van komplex rendszerek hierarchikus megvalósítására és a rendszer komponenseinek szimulációjára. Emellett képes különböző beágyazott, biztonságkritikus rendszerekre automatikusan kódot generálni az adott szakterületre (például repülőgépipar, autóipar, irányítástechnika) vonatkozó tanúsítványoknak megfelelően.

Probléma felvetés. Azonban a Simulink általános modellezési megközelítése miatt nehézkes ellenőrizni, hogy az elkészített modellben található komponensek megfelelnek-e a tervezett rendszer felépítésére vonatkozó szigorú tanúsítványozási követelményeknek (pl. DO-178C [16]). Ezen követelmények gyakran gráf jellegű megkötéseket írnak elő a tervezendő rendszer architektúrájára és topológiájára. Az ilyen megkötések leírására és kiértékelésére a Matlab legfeljebb a beépített, imperatív szkript nyelvével ad lehetőséget, amelyben a kiértékelés megvalósítása nehézkes és komplex követelmények esetén nem minden esetben hatékony. Ezen túlmenően nehézségeket okoz az is, hogy a szakterület-specifikus követelmények a Simulink általános célú modellezési nyelvében csak bonyolult módon definiálhatóak. A problémához kapcsolódóan a szakdolgozat az alábbiakkal foglalkozik:

- A Matlab-Simulink környezet és annak modellezési paradigmájának bemutatása. Ahhoz, hogy a modellekre felírt strukturális kényszerek betartását lehessen vizsgálni szükség van általánosan egy modell struktúrájának ismeretére.

- A modellekre vonatkozó deklaratív szabályok definiálására kidolgozott módszer ismertetése. A szabálydefiníciót egy, már létező technológiával, az EMF-IncQuery segítségével végezzük. Azonban ahhoz, hogy a szabályok megfogalmazhatók legyenek, vannak bizonyos előfeltételek, többek között egy olyan alkalmas metamodell megalkotása, melyre igaz, hogy a Simulink-beli modellek tekinthetők példánymodelljeinek.

- Egy futtató keretrendszer elkészítése a definiált szabályok kiértékeléséhez. A szabálydefiníciók alapján a rendszer képes az automatikus ellenőrzésre, továbbá a Matlab felhasználói felületén is jelezze a validáció eredményét.

- A megvalósított rendszer hatékonyságának mérése. A bemutatott módszer hatékonyságának értékeléséhez méréseket is kell végezni, melyek alapján összehasonlítható a tényleges teljesítménye az elvárttal. A mérések segítségével a dolgozatban bemutatott lehetőségek egymással is összehasonlíthatók.

Megközelítés. A probléma megoldására egy olyan módszert valósítottunk meg, amelyben a Simulink modellekre vonatkozó rendszervalidációs kényszerek deklaratív módon definiálhatók, és ezen kényszerek teljesülése tetszőleges Simulink modellen hatékonyan ellenőrizhető.

A validáció elvégzésére a megvalósított keretrendszerben két különböző módon van lehetőség. Az egyik, hogy a *validációs szabályokat deklaratív módon, gráfminták formájában* definiáljuk a Matlab-Simulink rendszerek általános fogalmainak segítségével. A *validációs kényszerek* nagy hatékonyságú *kiértékelését* az EMF-IncQuery modell lekérdező eszköz segítségével valósítottuk meg.

Az EMF-IncQuery inkrementális működésének köszönhetően a módszer képes akár százezres méretű Simulink modellekre is felskálázódni. Végül, a validációs szabályokat sértő elemeket a kialakított eszköz képes automatikusan *visszavetíteni az eredeti* Simulink *modellre*.

Ebben az esetben a modell helyességét Matlab-ban ellenőrizzük le úgy, hogy egy külön, erre a célra készített saját kódgenerátor segítségével előállítjuk azt az *imperatív* Matlab *script kódot*, mely képes az adott mintát egy modellre illeszteni. Az illesztés eredményét itt Matlab rendszerben kapjuk, így könnyen jelezhető a felhasználónak a modellben talált hiba.

Kapcsolódó munkák A Simulink rendszerek modell-alapú validációját más megoldások is részben támogatják, amelyek közül a MATE eszköz [7, 11] egy, a Fujaba modelltranszformációs eszközre építő megoldás, amely segítségével tervezési elvek példa alapon ellenőrizhetők. A [14] egy hibrid megközelítés, amely mind EMF, mind adatbázis technológiákat használ Simulink modellek tárolására, azonban nem alkalmas deklaratív validációs szabályok leírására. Végül a [6]-ban bemutatott megközelítés a VMTS transzformációs keretrendszerre épülve képes komplex modelltranszformációkat és lekérdezéseket végrehajtani Simulink modellek felett.

Az említett megoldások mind modell-alapon működnek, azonban az általam megvalósított módszer inkrementális megközelítést használ a validációs szabályok hatékony kiértékelésére.

A kódgeneráláson alapuló megközelítésünkhöz hasonlót, azaz deklaratívan megfogalmazott szabályok alapján a modell strukturális helyességét vizsgáló kódot generálni képes eszközt az általunk ismert szakirodalomban nem találtunk. Jelen dolgozat foglalkozik az említett módszerek hatékonyságának összehasonlításával, illetve az eredmények értékelésével is.

A dolgozat felépítése

- A 2. fejezetben egy repülőgépipari mintapéldán keresztül röviden bemutatom a MATLAB-SIMULINK modellezési megközelítését, az Eclipse Modeling Framework keretrendszert, és az EMF-INCQUERY inkrementális modell lekérdező rendszert.

- A 3. fejezetben egy magasszintű áttekintést adok az általunk megvalósított modell-alapú validációs megközelítésről.

- A 4. fejezetben részletezem az EMF alapú megvalósítás legfőbb lépéseit: (i) a SIMULINK modellek EMF alapú reprezentációját (4.1.3. fejezet), (ii) a validációs szabályok definiálásának módját (4.2. fejezet) és végül (iii) az eredmények visszajelzését az eredeti modellbe (4.3. fejezet).

- Az 5. fejezet bemutatja az újszerű megközelítésünket a szabályok MATLAB környezetben történő kiértékeléséhez. Bemutatjuk a módszer megvalósíthatóságának hátterét és alapjait (5.1. fejezet). Ezt követően foglalkozunk az elkészült megoldás bemutatásával, és a sajátosságok, újszerű megoldások hangsúlyozásával (5.2. fejezet).

- A megvalósított módszert értékelem a 6. fejezetben, míg a dolgozat zárásaként a 7. fejezetben összefoglalom a leírtakat.

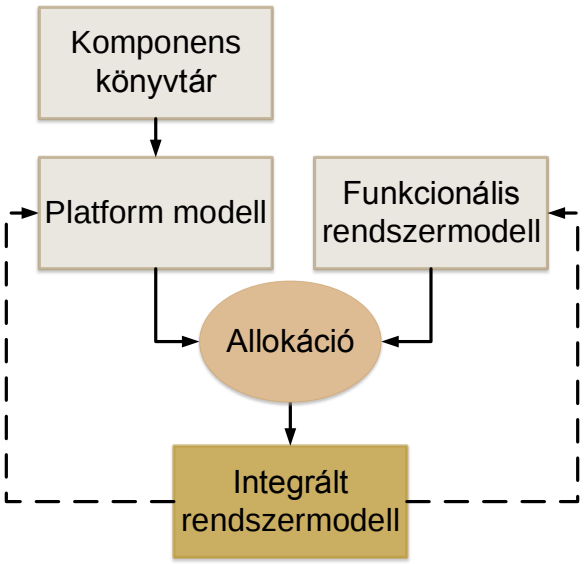
2. fejezet

Modellezés és Matlab-Simulink

Beágyazott és biztonság-kritikus rendszerek fejlesztése során a rendszermodelleket gyakran a MathWorks cég által készített MATLAB program és modellező rendszerében, a SIMULINK-ben készítik. Ebben a fejezetben egy repülőgépipari alkalmazáson és egy hozzá kapcsolódó példán keresztül bemutatom a MATLAB-SIMULINK modellezési megközelítést, továbbá a modellvezérelt rendszerfejlesztésben széleskörben használt Eclipse Modeling Framework keretrendszert és az arra épülő EMF-INCQUERY inkrementális modell lekérdező eszközt.

2.1. Trans-IMA, mint alkalmazási terület

A projekt célja, hogy egy modellvezérelt megközelítés segítségével állítsa elő egy tervezett repülőgép komplex, a hardvert és a funkcionalitásokat is magába foglaló szimulálható SIMULINK modelljét. Ehhez kiindulásként rendelkezésre áll a funkciókat magába foglaló *funkcionális rendszermodell* (Functional Architecture Model, röviden: FAM), illetve a hardverplatform leírására szolgáló komponens könyvtár. Ezek felhasználásával a cél, hogy létrehozzuk az integrált rendszermodellt úgy, ahogy azt a 2.1. ábra is szemlélteti.



2.1. ábra. *Trans-IMA koncepció*

Az ábra egyes komponensei a következők:

- *Funkcionális rendszermodell*: a repülőgép egyes szoftverfunkcióinak részletes modellje. Fontos jellemzője ennek a modellnek, hogy nem tartalmaz semmi információt arra vonatkozóan, hogy az adott funkciót a konkrét implementációban milyen eszköz valósítja meg, ezért gyakran *platformfüggetlen modellként* is hivatkoznak rá.
- *Komponens könyvtár*: ez tartalmazza azokat a hardvereket, eszközöket, melyeket a tervezőmérnökök a repülőgép építéséhez használhatnak.
- *Platform modell*: a komponens könyvtár építőelemeiből összeállított modell, mely a gépen elérhető hardverkomponensek és azok kapcsolatait tartalmazza.
- *Allokáció*: az a folyamat, amikor a rendszertervező mérnök specifikálja, hogy az egyes funkciók melyik konkrét fizikai egységen hajtódjon végre. Az allokáció emellett magába foglalja azt a feladatot is, mely automatikusan azt vizsgálja, hogy az egyes FAM funkciók az allokáció után milyen kommunikációs útvonalakon képesek egymással az információcserére.
- *Integrált rendszermodell*: a MATLAB-SIMULINK környezetben szimulálható, a rendszerről az allokáció alapján automatikusan előállított, átfogó modell.

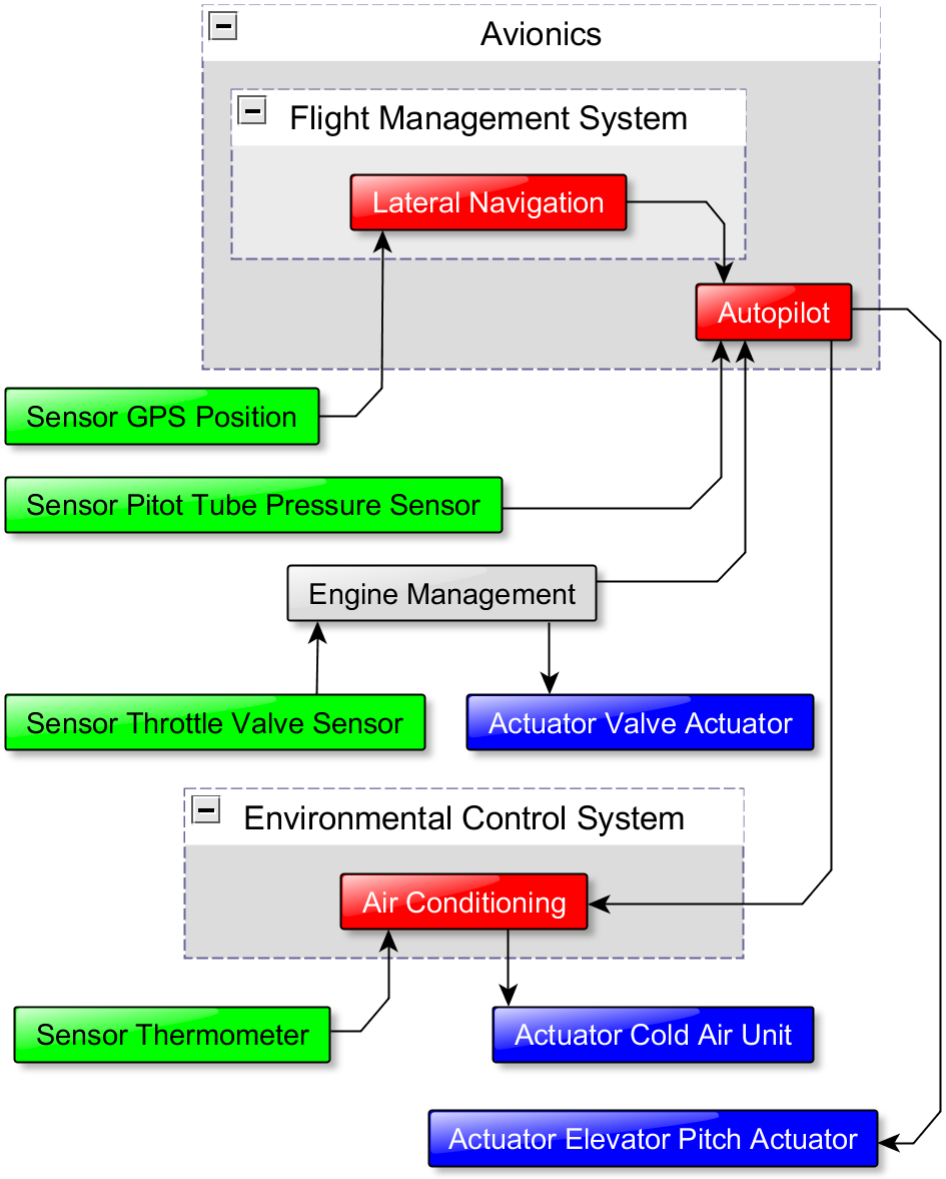
A kezdetben rendelkezésre álló modellek (komponens könyvtár és a funkcionális rendszermodell), illetve a komponens könyvtár elemeiből a tervezőmérnök által összeállított platform modell a Trans-IMA esetében mind SIMULINK modellek. Emiatt a modellekben a szimulációs programrendszer adta lehetőségek felhasználásával kell a *szakterület specifikus* tulajdonságokat leírni.

2.2. Mintapélda: repülőgép funkcionális modell részlete

A biztonságkritikus rendszerek alkalmazásának egyik fő példája a repülőgépek. Ezek nagyon sok komponensből álló bonyolult rendszerek, továbbá tervezésükre szigorú szabályozások érvényesek.

Példa: Egy egyszerűsített példa FAM szerepel a 2.2. ábrán, mely a civil repülőgépiparban használt fogalmakból építkezik. Az egyes dobozok jelölhetnek érzékelőket (ezek a *Sensor*ok), beavatkozókat (az összes *Actuator*) és funkciókat. A feltüntetett nyilak az adatáramlás irányát szemléltetik az egyes modellelemek között. A modellen belüli hierarchizáláshoz megkülönböztetünk bizonyos FAM funkciókat: a legfelső szinten *Root*, a köztes szinteken *Intermediate*, a legalsó szinteken pedig *Leaf* funkciók szerepelhetnek. A modellezett fő feladatok az alábbiak:

Avionics azon feladatok összessége, amelyek a repüléshez kapcsolódnak. Jelen esetben több alfeladatot is tartalmaz. Egyik ilyen az *Autopilot*, azaz robotpilóta, mely a gép kormányzását képes a rendelkezésre álló információk alapján végezni. Emellett egy másik



2.2. ábra. *Egy példa funkcionális architektúra modellre*

funkciót az úgynevezett *Flight Management System* lát el, mely lehetővé tette, hogy a mai modern utasszállítókon sok olyan folyamat is automatizálódott, amihez korábban külön személyzetre volt szükség a fedélzeten, mint például a repülési terv megalkotása. A modell pontosításának érdekében ezen feladaton belül fel van tüntetve a *Lateral Navigation*, azaz a vízszintes navigációt teljesítő funkció. A feladat helyes működéséhez a modell alapján szükség van a GPS segítségével meghatározott pozícióra, illetve a Pitot-csővel mért sebességadatokra.

Engine Management a hajtóművek motorjainak monitorozását és megfelelő irányítását hajtja végre. A példában ehhez szüksége van a szelepeknél elhelyezett érzéke-

lők adataira, amin keresztül visszajelzést kap a motorokról. Emellett feladata a szelepek visszacsatolás alapján történő vezérlése is.

Environmental Control System az utasszállító gép törzsében gondoskodik arról, hogy repülés közben az utastérben a nyomás, hőmérséklet és egyéb, az emberek életben maradásához szükséges alapvető körülmények megfelelők legyenek. A modellen kiemelt *Air Conditioning* alfunkció működését, melynek szerepe a friss és fertőtlenített levegő biztosítása, befolyásolja a robotpilóta aktuális ismerete a magasságról, továbbá a példánkban az utastéri hőmérő. Beavatkozója az úgynevezett *Cold Air Unit*, mely a levegő keringetését végzi a gép törzsében.

A dolgozatban szereplő további példák a könnyebb megértés és átláthatóság végett mindig az imént bemutatott rendszerrel kapcsolatosak.

2.3. Matlab-Simulink

A Simulink hivatalosan 1990 óta [13] a Matlab részét képezi. A világon azóta sok helyen elterjedt, számtalan területen alkalmazzák, így nagyon sok mérnök is. Szokták mondani, hogy a világon bizonyos szinten az angol hivatalos nyelvnek tekinthető napjainkban. Ehhez hasonlóan, ha két fél valamilyen rendszer tervét, számítást vagy rendszer szimulációt szeretne megosztani egymással, akkor ők nagy valószínűséggel választják a Matlab-ot vagy a Simulink-et közös nyelvnek. A grafikus felületén könnyűszerrel megépíthetők rendszerek, melyek szimulálhatók, tesztelhetők és akár a modellezett eszközöket működtető programkód is generálható belőlük. A szoftver elterjedtsége és népszerűsége miatt a kompatibilitás az előző verziókkal igen lényeges szempont. Emiatt szinte bizonyos, hogy az ebben a rendszerben elkészített modelleket leíró, mára kiforrottnak mondható belső struktúra lényegében nem fog változni, ami néhány nem túl gyakori, de mégis nagyon fontos alkalmazás esetén nehézségeket és gondot okozhat.

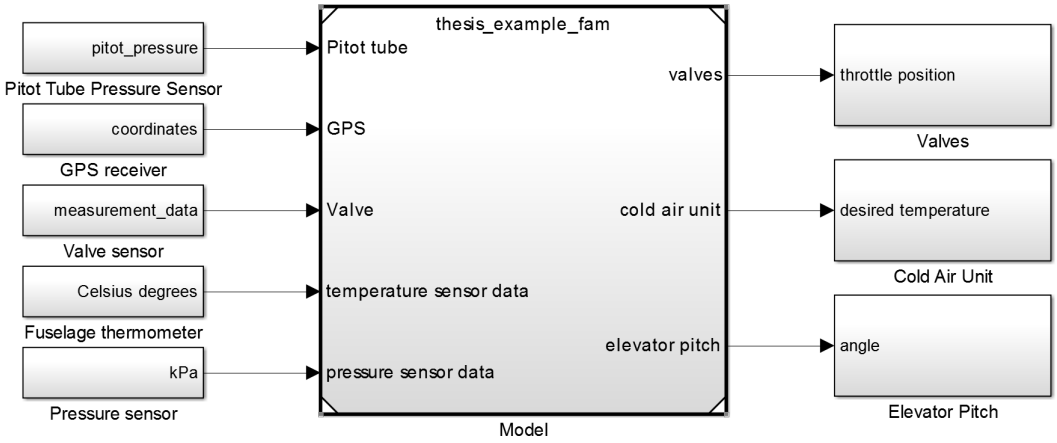
A dolgozat szempontjából a modellek szerkezeti felépítése a hangsúlyos. Az alábbi felsorolásban szerepelnek a leggyakoribb komponensek:

- Az elemi építőelem a *blokk (block)*. Ezek többségében valamilyen funkciót valósítanak meg, melyek a rendszer viselkedését befolyásolják és meghatározzák.
- A blokkok rendelkezhetnek *portokkal (port)*, ezeken keresztül csatlakozhatnak a többi blokkhoz.
- A blokkok között *összeköttetések (connection)* futhatnak, melyek egy blokk portjából indulnak, és egy vagy több blokk portjához csatlakoznak.

A felhasználható blokkokat *könyvtárak (library)* tartalmazzák, ezek másolásával, és ha szükséges módosításával hozhatók létre *rendszerek (system)*, másnéven *modellek (model)*.

Példa: A FAM működés közbeni szimulációjára alkalmas Simulink blokkdiagramon (2.3. ábra) középen szerepel a funkcionális architektúra modell, két oldalán pedig a hozzá kap-

csolódó elemek, amik jelen esetben érzékelők és beavatkozók. Általánosságban is igaz, hogy a rendszer működésének helyessége szimuláció segítségével sokszor olyan esetre is leellenőrizhető, amely a megvalósítás után már nehéz vagy nagyon költséges lenne.



2.3. ábra. *A példa FAM és szimulációs környezete*

A környezet állapotáról információt szolgáltató jelek a 2.3. ábra bal oldalán elhelyezkedő blokkokból származnak. A blokkok közötti nyilak az összeköttetések, melyek outportokból indulnak és inportokba mutatnak, ezzel egyben a jelterjedés irányát is meghatározzák. Egy jel több helyre is rákapcsolható. Egy port jelen modell esetén csak egyirányú kommunikáció lebonyolítását támogatja.

A blokkokat reprezentáló téglalapok belsejében van feltüntetve az egyes portok neve, illetve a blokk alatt szerepel a diagramonként egyedi blokknév.

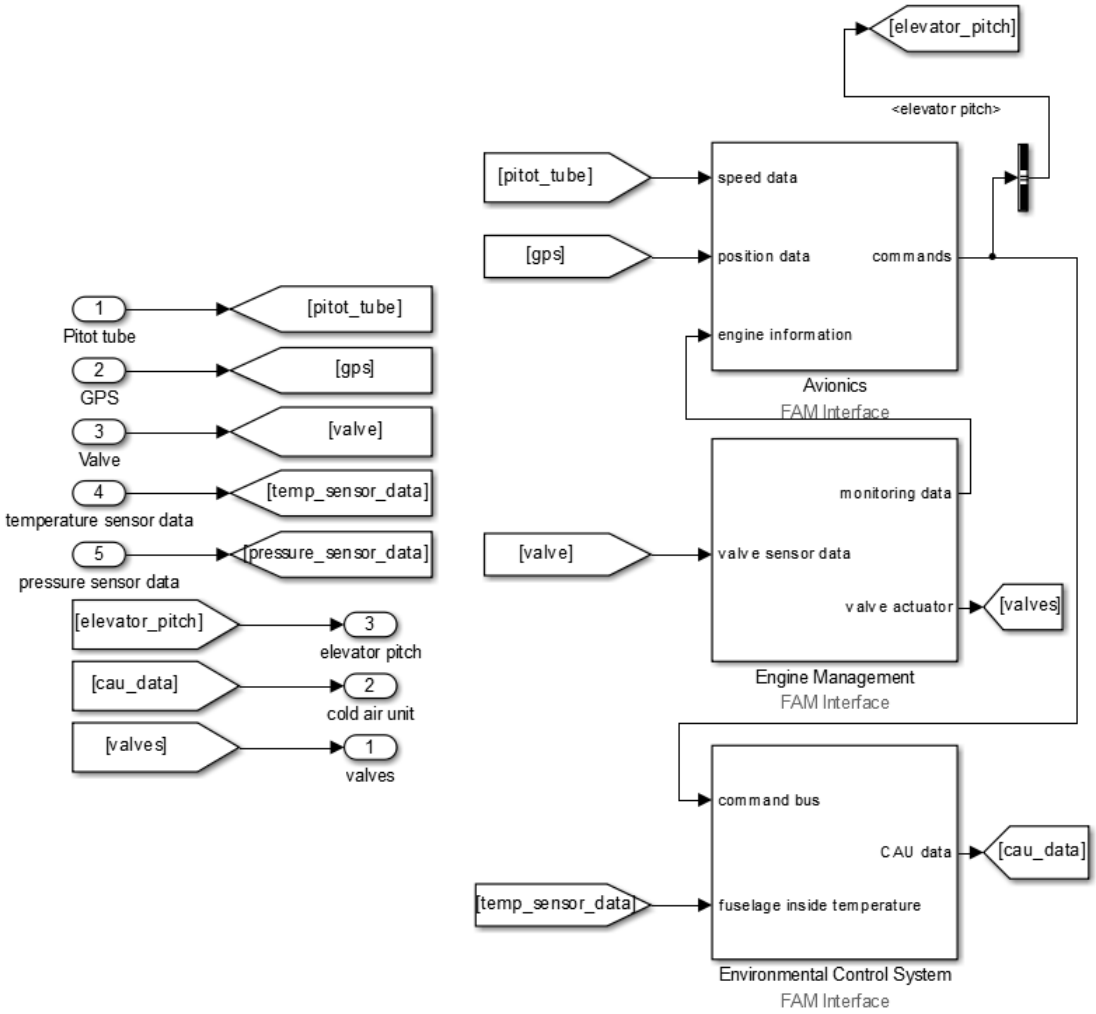
2.3.1. Rendszer, Modell (System, Model)

A Simulink-ben a fenti két fogalom többségében ugyanazt jelöli, így a dolgozatban mindkettő használt. A modell felépítését tekintve hierarchikus; blokkokat tartalmazhat. Már meglévő blokkok *másolásával* lehet egy modellen újakat létrehozni. Egy blokk lehet úgynevezett *alrendszer (subsystem)* is, ekkor további elemeket tartalmazhat a belső felépítésében. Továbbá ez igaz visszafele is, azaz ha egy blokk tartalmaz további blokkokat, akkor az csak alrendszer lehet.

Alrendszerek létrehozhatók blokkok csoportosításával közvetlen a modellen is (ekkor hierarchiát tekintve egy szinttel mélyebbre kerülnek a létrehozott alrendszerben összefogott elemek a tartalmazási fában), de vannak előre definiált alrendszerek is. Összeköttetések a blokkok között akkor hozhatók létre, ha azok azonos szinten szerepelnek. Alrendszerek esetén a bejövő jeleket speciális, úgynevezett *port blokkok (port block)* reprezentálják a belsejében.

Példa: A 2.4. ábra mutatja be a példa FAM Simulink reprezentációját, ahol minden funkcionális elemet blokkok reprezentálnak. A Simulink grafikus modellszerkesztőjében

egy időben egyszerre csak egyetlen hierarchiaszintet lehet megjeleníteni, így csak a FAM legfelső szintjén elhelyezkedő funkciók láthatók.



2.4. ábra. *A példa FAM legfelső szintje* Simulink*-ben*

A blokkdiagramon megjelenő jellemző Simulink modellelemek az alábbiak:

- Egy *blokk* a portot reprezentáló *Pitot tube* (ami ennek köszönhetően egyben egy *port blokk* is).
- *Alrendszerek* az *Avionics*, *Engine Management* és az *Environmental Control System*, amelyek belső implementációi a nevüknek megfelelő FAM funkciót valósítják meg.
- Egy blokkhoz tartozó (egyik) attribútum a *Tag*, mely értéke a fent említett alrendszerek nevei alatt olvasható. Jelen alkalmazásban a célja, hogy az ebben tárolt érték segítségével lehessen azonosítani egy modellben az egyes FAM funkciók típusait, ami lehet *Interface, Root, Intermediate* vagy *Leaf*.
- A *goto-from* blokk párosok egymás között továbbítják a jelet a hozzájuk rendelt *gotoTag*nek nevezett érték alapján anélkül, hogy a modell átláthatóságát rontanák. Ezeket az ábrán az ötszög alakú elemek jelenítik meg.

2.4. Eclipse Modeling Framework

Az Eclispe Modeling Framework egy Java alapú keretrendszer és kódgeneráló eszköz. Olyan alkalmazások létrehozására alkalmas, amelyek alapja egy strukturált modell. Az EMF biztosít egy metamodellt (ez az Ecore modell), mely példányosításával strukturált modellek hozhatók létre. A keretrendszer az ilyen módon létrehozott modelleket felhasználva biztosít eszközöket és futásidejű támogatást, hogy a JVM-ben a modellek reprezentációjára alkalmas Java-osztályokat létrehozza, emellett a modellek megjelenítését és az alapszintű szerkesztését lehetővé tegye.

A *metamodell* egy modell a modellről. Azt írja le, hogy egy modellen milyen modell elemek szerepelhetnek, illetve milyen kapcsolatban állhatnak.

Egy metamodell *példánymodellje* jelenti azt a modellt, aminek a felépítését a metamodell határozza meg. A szövegben a *modell* szó általában a példánymodellt jelöli, de a szövegkörnyezettől függően jelölhet metamodellt is.

Példa: Egy Simulink metamodell részlete szerepel a 2.5. ábrán. A felvett modellben megtalálhatók a Simulink modellen a leggyakrabban előforduló elemek, köztük a példáé is. Az Ecore bemutatása során ez egyes elemek jelentésére a magyarázat ezen példához kötődik.

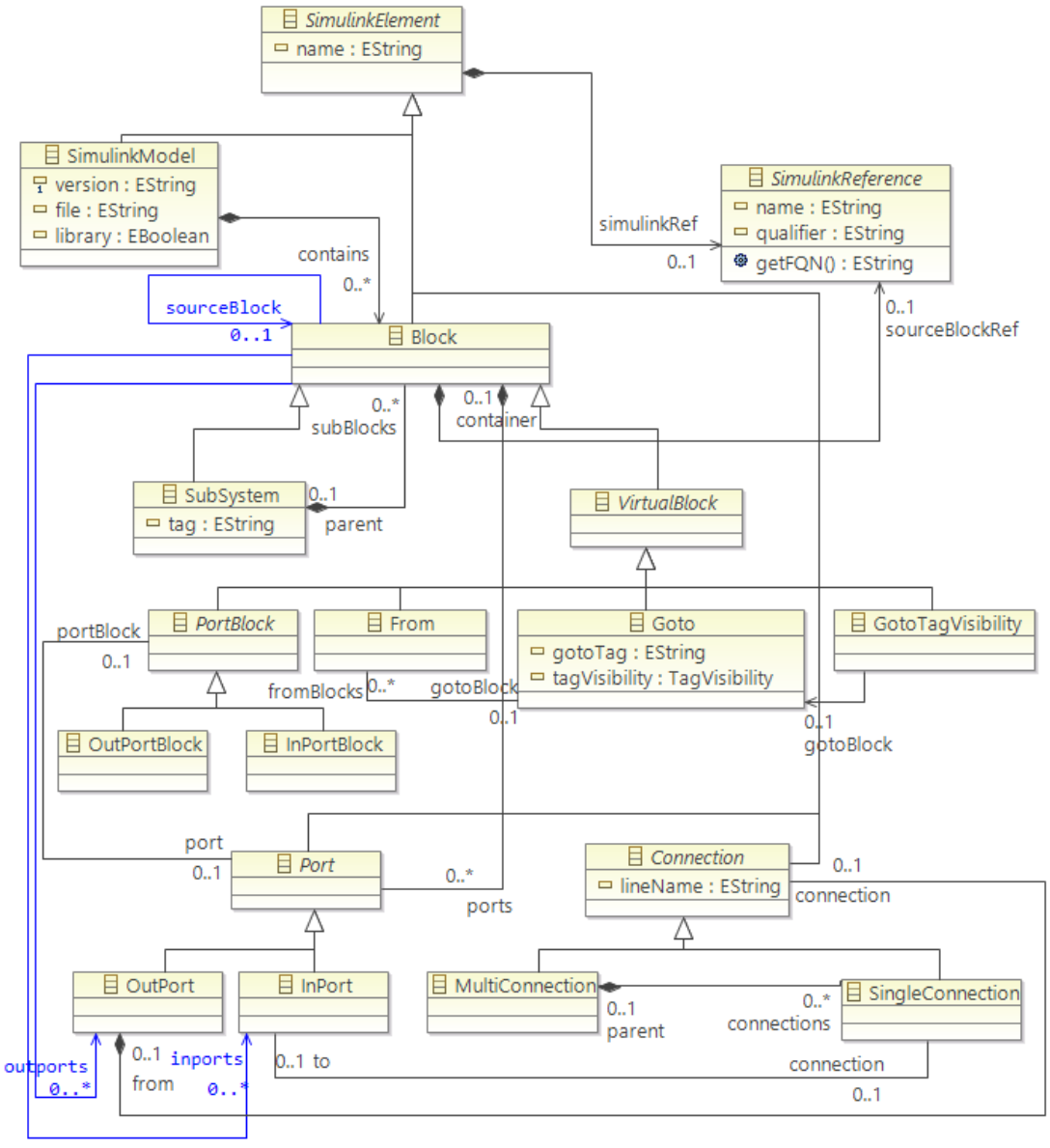
Az Ecore lényegében az UML osztálydiagram aldiagramja. Az Object Management Group (OMG) nevéhez fűződő Meta Object Facility (MOF) sepcifikáción alapul. Az Ecore metamodell elemei:

- *EClass* magukat az osztályokat modellezi. Az osztályokat a nevük azonosítja és tartalmazhatnak attribútumokat és referenciákat. Az öröklés támogatott osztályok között, egy osztálynak több őse is lehet. A példa Ecore modellben EClass-ként szerepel az összes Simulink diagramon előforduló elem, például a blokk is.

- *EAttribute* modellezi az attribútumokat, az objektum adatainak komponenseit. A nevük és típusuk azonosítja őket. A példában ezek a *Goto* EClass esetén: *gotoTag* és *tagVisibility*

- *EDataType* modellezi az attribútumok típusait, reprezentálva az objektum típusokat, amelyek Java-ban definiáltak, de EMF-ben nem. Az adat típusok szintén a nevükkel azonosíthatók. Az alábbi diagramon a *Goto* EClass *gotoTag* EAttribute-ja például *EString* típusú

- *EReference* az osztályok közötti asszociációk modellezésére használható, egy ilyen asszociáció egyik végét modellezi. Hasonlóan az attribútumokhoz, a referenciák is a nevükkel és típusukkal azonosíthatók. Jelen esetben a modellen névvel és kardinalitással ellátott kapcsolatok reprezentációjára alkalmazható, mint például egy *Goto* esetén a *fromBlocks* kapcsolat.



2.5. ábra. *A* Simulink *rendszerek metamodelljének részlete az EMF diagram-szerkesztőjében*

Az Ecore modell reprezentációja. Az alapvető formája egy EMF modellnek egy Ecore modell XML Metadata Interchange (XMI) szerializációja. Ennek ellenére az egyik legfőbb jellemzője mégis az, hogy sokféleképp definiálható az Ecore modell:

- egy (Ecore) *XMI dokumentum* létrehozható egy szöveg vagy XML szerkesztővel, vagy az EMF egyszerű tree-based sample Ecore editorának segítségével
- egy Ecore modell létrehozható egy *UML modellező eszköz* segítségével, mint például a Rational Rose
- sima *Java interfészek, melyek speciális Java annotációkkal vannak ellátva* (`@model`), szintén használhatóak Ecore model leírására
- egy *XML séma*, mely definiálja a modell adatszerkezeteit konvertálható EMF modellé
- az EMF egy könnyen kiterjeszthető keretrendszer/eszközkészlet, tehát *más formái is támogatottak* a modelldefiníciónak

Az első megközelítés a legközvetlenebb, de általában csak az XML-ben jártasabbaknak vonzó. A második lehetőség abban az esetben lehet kedvező, ha a felhasználó már ismerős az UML-lel és a modellvezérelt szoftverfejlesztéssel, amíg a Java-s megközelítés azoknak előnyös, akik tisztán Java környezetben fejlesztenek, például Java Development Tool (JDT) [4] használatával. Az XML séma alapú megadás akkor célszerű, ha az alkalmazás feladata olyan XML-ben tárolt adatok manipulálása, amelyek illeszkednek a sémára (mint a webes szolgáltatások esetén). Függetlenül attól, hogy melyik formáját választjuk a definíciónak, az előnyök ugyan azok, és majdnem minden típus direkt generálható az Ecore modellből.

Az Ecore kiterjesztése. Egy Ecore modellből az EMF generátora képes létrehozni az őket megvalósító Java osztályokat. Minden ilyen generált EMF osztály a keretrendszer alaposztályából, az `EObject`-ből származik le, ami lehetővé teszi az objektumok könnyű integrálhatóságát és megjelenését az EMF futásidejű környezetében. Az `EObject` egy hatékony, *reflektív API*-t biztosít az objektum tulajdonságainak generikus hozzáféréséhez. Mindemellett a *change notification (változás jelzés)* egy alapvető tulajdonsága minden `EObject`-nek, és az objektumok kiterjesztésének támogatásához egy csatoló keretrendszer használható fel. Az EMF egyik fő előnye, hogy *dinamikus modellekhez* is ad támogatást, ami azt jelenti, hogy az Ecore modellek amik létrejöttek (a memóriában), kódgenerálás nélkül példányosíthatók. Minden modellezett objektum implementálja az `EObject` interfészét, hogy biztosíthassa az alábbiakat:

- Az EMF *reflektív API*-ja [15] lehetővé teszi, hogy minden attribútum és referencia, ami az `EObject`-hez köthető, lekérdezhető legyen az `eGet` és `eSet` függvényekkel. Ez koncepcióját tekintve megegyezik a `java.lang.reflect.Method.invoke()` Java metódussal, habár annál sokkal hatékonyabb teljesítmény szempontjából.

- *Notification observers/listeners* (jelzés figyelők) elnevezése az EMF-ben adapter (csatoló) [3], mivel a megfigyelő státuszuk mellett gyakran a megfigyelt objektum viselkedését egészítik ki (így támogathat egy objektum több interfészt anélkül, hogy leszármazna egy újabb osztályból). Egy `Adapter`, mint sima megfigyelő, bármilyen `EObject`-hez hozzáredelhető, mindössze az adott objektum `eAdapters` listájához kell hozzáadni. Az adapter implementál egy `notifyChanged` nevű függvényt, ami az `Adapter` objektumot tartalmazó `EObject` manipulációjakor meghívódik. A változással kapcsolatos minden információt egy jelzés objektum (notification object) tartalmaz, ami a `notifyChanged` függvény input paramétere.
- EMF támogatja mind meta- és példányszinten a *dinamikus modelleket* [5]. A keretrendszer dinamikájának és a reflektív API-nak a kombinálásával lehetőség nyílik a metamodell futásidejű megváltoztatására.

2.5. EMF-IncQuery

Az EMF-IncQuery [2] egy keretrendszer, mely lehetővé teszi deklaratív lekérdezések (query-k) definiálását EMF modellek esetén. Ezen lekérdezések aztán hatékonyan végrehajthatóak egyéb kézzel írt kód nélkül is.

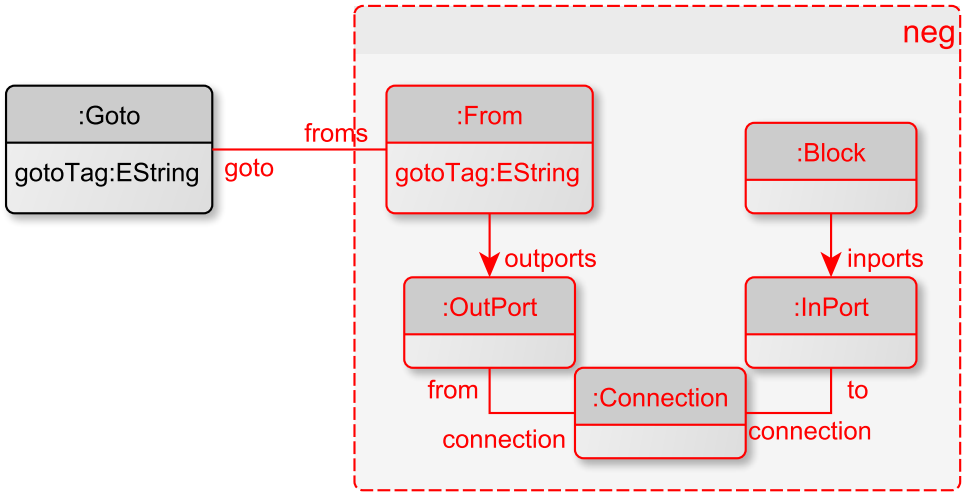
A lekérdezőnyelvben a gráfminták koncepciója kerül újra felhasználásra, mivel ez egy egyszerű és tömör módja bonyolult strukturális modell lekérdezések megfogalmazásának. A nagy futásidejű teljesítményt inkrementális gráfminta illesztő technikákkal éri el.

Példa: Legyen az egyik kikötés a Simulink modellel szemben az, hogy minden goto blokkhoz létezzen legalább egy hozzárendelt from blokk, amely egy blokk inportjához csatlakozik, hiszen ellenkező esetben nem továbbítódna a goto-ba vezetett jel sehova a modellben, így az csak egy felesleges elemként szerepelne. Ha van olyan goto, amely nem teljesíti a feltételt, a modell hibás. Ennek megfogalmazása szerepel minta formájában a 2.6. ábrán.

A feltüntetett kapcsolatok az egyes metamodellbeli EReference-ek nevei, az elemek pedig a metamodellben szereplő EClassok példányai. Általánosságban és az ábrán is, a *neg* felirattal ellátott rész jelentése, hogy ha a neg-ben szereplő objektumok és a közöttük létező kapcsolatok nem pont ebben az elrendezésben szerepelnek, a minta illeszkedik, és a keretrendszer egy hibajelzést generál a felhasználó számára.

Az EMF-IncQuery egy saját, *szöveges nyelvet* használ a minták megadására. Ennek segítségével írhatók le az egyes mintákban szereplő változókra alkalmazandó különböző megkötések. A mintadefinícióhoz az alábbi fő kulcsszavak és fogalmak tartoznak:

- `pattern`: ez a kulcsszó mutatja egy mintadefiníció kezdetét, utána szerepel a minta neve és bemenő paraméterei. Az ezt követő { és } zárójelek közé foglalt blokk (bizonyos esetekben több blokk is) jelenti a *minta testét* (angolul *pattern body*).
- Megkötések (angolul constraints): a változókra *típusmegkötések* (angolul *classifier*



2.6. ábra. *Validációs szabály hibásan szereplő goto blokkra*

constraint) és útvonalkifejezések (angolul *path expression*) adhatók meg. A típusmegkötések a behelyettesített változó típusára vonatkoznak, az útvonalkifejezések pedig a változók, mint modellelemek egymáshoz viszonyított kapcsolatára fogalmaznak meg feltételeket.

- `find`: ez használható korábban megírt minta *hívására* egy másik mintából. Célja, hogy a változókra olyan megkötéseket is fel lehessen írni, amit egy másik mintára való illeszkedéssel fogalmazunk meg.
- `neg`: a `find` kulcsszó elé írva fogalmazható meg vele egy minta nem illeszkedésére feltétel. Azaz a `neg find` segítségével hívott minta akkor fog visszaadni illeszkedést, ha a `find` után szereplő hívás nem illeszkedik.

Példa: A 2.6. ábrán bemutatott gráfminta EMF-IncQuery szöveges nyelvén megadott definícióját a 2.7. ábra mutatja. A `notWellformedGoto` minta definíciójához előbb definiáltuk a helyes goto blokkokra illeszkedő mintát, ez a `wellformedGoto` minta. Ennek bemeneti paramétere a `G` változó, mely olyan értékeket vehet fel, ahol `G` típusa `Goto`, emellett `fromBlocks` kapcsolat köti össze egy `F` elemmel, ahol `F From` típusú. Továbbá elvárás, hogy `F` össze legyen kötve egy tetszőleges blokkal (neve `_B`), ezt adja meg a `find directedLineExistsBetween(F,_B)` feltétel.

Ezt követően a hibás goto blokkokra illeszkedő minta megfogalmazása ennek segítségével egyszerű: az a `Goto` típusú modellelem hibás, akire a helyességet leíró minta nem illeszkedik.

A minta feletti `@Constraint` annotáció a hibajelzés megjelenítésére szolgál a grafikus felhasználói felületen.

A keretrendszer által támogatott deklaratívan megfogalmazott gráfminták definiálásá-

```
pattern wellformedGoto(G){
    Goto(G);
    Goto.fromBlocks(G,F);
    From(F);
    find directedLineExistsBetween(F,_B);
}

@Constraint(
    location = G,
    severity = "error",
    message = "Goto block $G.name$ has no properly connected corresponding from"
)
pattern notWellformedGoto(G){
    Goto(G);
    neg find wellformedGoto(G);
}
```

2.7. ábra. *Hibás goto blokra illeszkedő minta definíciója* EMF-IncQuery *használatával*

val a modell bejárásának implementálása nem feladatunk, automatikusan megtörténik. Továbbá ilyen módon kényelmesen fogalmazhatóak meg EMF objektumok közötti bonyolult relációk is, melynek számos előnye van:

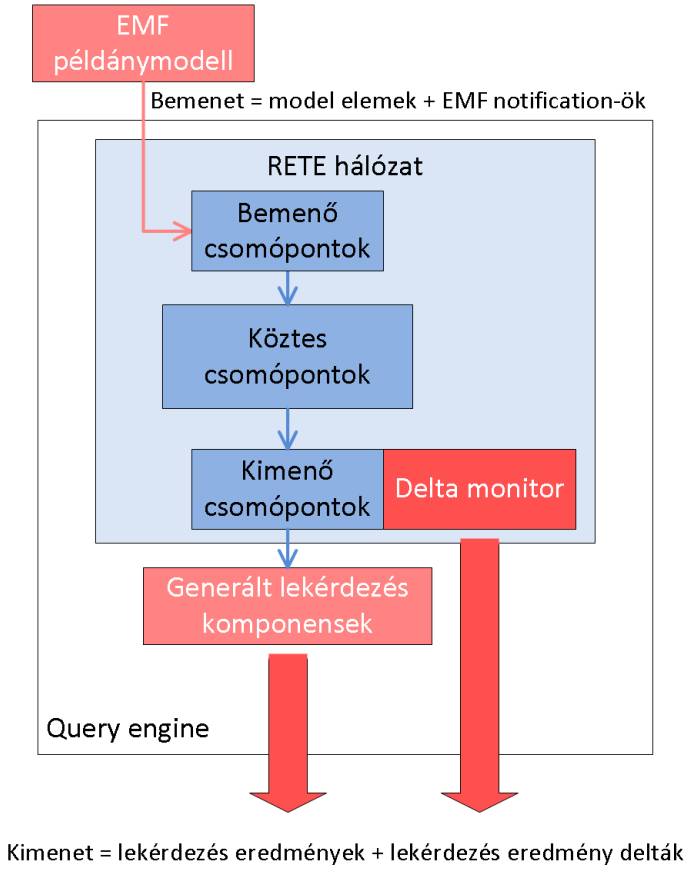
- a nyelv igen kifejező és több nemtriviális nyelvi elemet is tartalmaz, mint például a tagadás (negálás) vagy a számlálás
- a minták összeilleszthetők és újra felhasználhatók
- lekérdezések paraméterei futási időben változtathatók
- az EMF ismert hiányosságait figyelembe veszi:
  - egyszerű és hatékony felsorolása egy osztály összes példányának
  - egyszerű követése az EReference kapcsolatoknak mindkét irányban akkor is, ha nincs megadva a kapcsolathoz EOpposite
  - objektum keresése attribútumérték alapján
- az inkrementális lekérdezés jelentősen gyorsabb bonyolult strukturális minták lekérdezése esetén, ha közben a modell változása nem jelentős (például folytonos validáció esetén)
- a deklaratív módon megfogalmazott lekérdezésekből minta illesztő kód generálódik, ami egy kevés függőséggel rendelkező Eclipse plug-inban foglal helyet

Az inkrementális gráfminta illesztés folyamat bemenete az EMF példánymodell, amire a mintát kell illeszteni és a hozzátartozó notification API. A notification API lehetővé teszi callback függvények regisztrálását a példánymodell elemeihez, amik jelzés objektumokkal (angol elnevezés a notification object, ilyen lehet például ADD, REMOVE, SET, stb.) paraméterezve hívódnak meg, ha valami alapvető művelet hajtódott végre. Ehhez kapcsolódóan már az EMF adapterekről volt szó a 2.4. részben.

A modell lekérdezés leírásának függvényében az EMF-IncQuery létrehoz egy Rete szabály kiértékelő hálózatot, ami feldolgozza a példánymodellben tartalmazott elemeket, hogy megalkossa az eredményt, mint kimeneti csomópontot. A lekérdezéseket az automatikusan generált lekérdezés komponensek ez után újra feldolgozzák, hogy így biztosítsanak típushelyes hozzáférési réteget a könnyű alkalmazásbeli integrációhoz. Ez a Rete hálózat addig marad fenntartva, amíg a lekérdezésre szükség van: továbbra is megkapja az elemi változásokról az értesítéseket, és tovább terjeszti őket. Ennek következtében lekérdezés eredmény deltákat (query result delta) hoz létre a delta monitor lehetőség segítségével, amiket az eredmény inkrementális frissítésére használ fel.

Ezen megközelítés miatt a lekérdezés eredmények (például a gráfminta illeszkedések találatai) folyamatosan karban vannak tartva egy memóriában, és direkt módon hozzáférhető. Annak ellenére, hogy ez egy nem túl nagy teljesítménybeli terhet jelent, és a memóriában elfoglalt helye arányos a találatok számával (nagyjából a találatok halmazával egyenlő nagyságú).

Az EMF-IncQuery képes hatékonyan kiértékelni bonyolult lekérdezéseket nagyméretű modelleken is. Ez a speciális teljesítmény karakterisztika, amíg elegendő a rendelkezésre álló memória, jól skálázhatóságot eredményez. Ez az architektúra a 2.8 ábrán látható.



2.8. ábra. *Az* EMF-IncQuery *architektúrája [9]*

3. fejezet

Matlab-Simulink modell alapú validációjának áttekintése

A modellvalidáció célja az esetleges *hibák megkeresése és jelzése* a felhasználónak. Annak érdekében, hogy ezt megtehessük, először szükség van a szabályok precíz definíciójára, illetve egy olyan környezetre, mely képes az így megadott strukturális kényszerek érvényesítésére. Ennek elérése érdekében két különböző megközelítést dolgoztunk ki, amit a 3.1. ábra szemléltet.

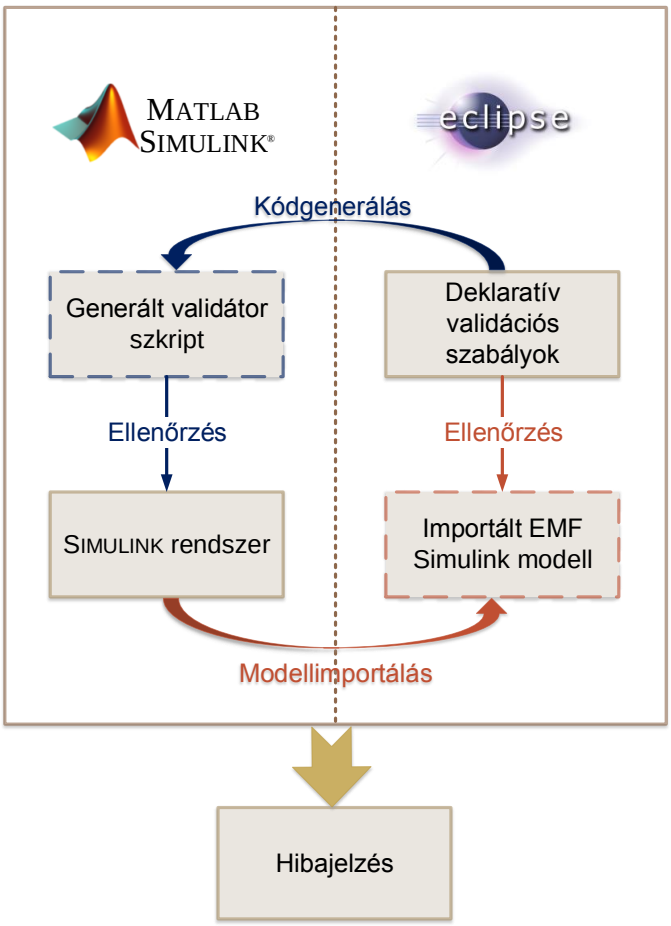
Mindkét megoldás esetére a szabályok definiálásához az EMF-IncQuery gráfminta leíró deklaratív nyelvét használjuk. Ez a választás a modellek gráf jellegére való tekintettel kedvező, hiszen ezáltal könnyű megfogalmazni olyan, a modellek felépítésre vonatkozó elvárásokat is *gráfminták formájában*, melyeket egy imperatív nyelv használatával nehézkes.

Modell alapú validáció EMF felhasználásával Egyik lehetőségként az EMF-IncQuery keretrendszer felhasználható a validációs szabályok Simulink modellek feletti ellenőrzésére, ehhez kapcsolódnak a 3.1. ábrán a piros komponensek. Ez az eszköz képes több, mint 100 000 elemet tartalmazó modellek esetén is az ellenőrzést hatékonyan elvégezni *inkrementális* algoritmusok használatával (lásd 4.2. fejezet). Ez a rendszer közvetlenül értelmezi a deklaratív módon definiált gráfmintákat, majd a modelleket automatikusan ellenőrzi, és megtalálja az esetleges hibákat.

Ennek használatához azonban szükséges a Matlab-Simulink-ben található modellek leképezése EMF reprezentációra. Ehhez elsőként az importálni kívánt Simulink *modellek reprezentációjára alkalmas metamodellt* kell megtervezni (lásd 4.1. fejezet).

Mindemellett gondoskodni kell a *rendszerek közötti integrációról* is, hiszen el kell tudni érni az eredeti modelleket a validációt végző rendszerből, de ugyanakkor a modellellenőrzés eredményéről visszajelzést kell küldeni a felhasználónak (lásd 4.1.2. fejezet). Az integráció megvalósításával és metamodell elkészítésével lehetővé válik modellek importálása Simulink-ből EMF alá (lásd 4.1.3. fejezet).

A hibás elemekről kapott lista alapján azonosítani kell tudni a nekik megfelelő Simulink-beli elemeket, annak érdekében, hogy a felhasználót a Matlab-Simulink *felületén* értesítsük (lásd 4.3. fejezet).



3.1. ábra. *A kétféle validációs megoldás vázlata*

Modell alapú validáció kódgenerálás segítségével Ezt a megoldást az összefoglaló ábrán kékkel jelölt elemek reprezentálják. Alapja, hogy közvetlen SIMULINK modellek fölött, MATLAB környezetben ellenőrizzük a megadott szabályok betartását. Ahhoz hogy ezt elvégezhessük egy kódgenerátor segítségével minden szabályt jelentő gráfmintához egy, a szakdolgozat keretében elkészített kódgenerátor eszközzel elő kell állítani a konkrét mintát illeszteni képes validátor függvényt (lásd 5. fejezet).

A kódgeneráláshoz egy *local-search módszertanon* alapuló keresési technikát használtuk fel (lásd 5.1.1. fejezet), ezáltal a korábbitól eltérő megközelítést alkalmazva a minták illesztésére. Ekkor magát a validációt a MATLAB rendszer végzi, így csupán a szabály megadása és a megfelelő szkript előállítása kötődik más technológiákhoz és környezethez.

A visszakapott lista alapján, mely valamennyi szabályt megsértő elemet tartalmaz, a hibák már könnyen megkereshetők, és ezek alapján javíthatók.

4. fejezet

Modell alapú validáció EMF felhasználásával

Ebben a fejezetben részletesen bemutatjuk a kialakított módszer azon lépéseit, mely egy EMF modell felett végzi el a validációt. Először a Simulink rendszerek EMF alapú modell reprezentációját és e modellek Matlab-ból való importálási módját ismertetjük (4.1. fejezet). Ezután bemutatjuk a validációs szabályok deklaratív definícióját ezen előállított modellek felett (4.2. fejezet) és végül a validáció eredményeinek visszajelzését az eredeti modellen (4.3. fejezet).

4.1. Simulink rendszerek modellezése

Minden Simulink modell a struktúráját tekintve azonos: blokkokból épül fel, a blokkok portokkal rendelkezhetnek, ezeken keresztül kapcsolódhatnak egymáshoz az egyes blokkok. Emellett a blokkok további blokkokat tartalmazhatnak, ezáltal egy hierarchikus tartalmazási struktúrát létrehozva. A modellelemek típusai tehát semmilyen információt nem hordoznak azzal kapcsolatban, hogy pontosan mit reprezentálnak. Erről mindössze az adott elem szöveges paraméterei adnak információt.

Ennek megfelelően kellett elkészíteni azt az EMF metamodellt, aminek a példányai alkalmasak tetszőleges Simulink modellek reprezentációjára a modellezett rendszer szakterületétől függetlenül. Ennek a metamodellnek egy részlete a korábban a mintapélda kapcsán bemutatott a 2.5. ábrán látható, amely tartalmazza a leglényegesebb, és a példa megértéséhez szükséges elemeket.

4.1.1. Az egyszerűsített metamodell elemei

Szinte minden, a Simulink-ben megtalálható elem EMF reprezentációja egy olyan EClass, ami az absztrakt `SimulinkElement` osztály leszármazottja, mivel ez az osztály tartalmazza az azonosításhoz szükséges `simulinkRef` referenciát. Minden esetben, ahol a nyomonkövethetőség fontos, az adott objektum tartalmaz egy `SimulinkReference` típusú objektumot. Ez hordozza az egyértelmű azonosításhoz szükséges *teljes nevet* (ez a `qualifier` és a `name` attribútumok konkatenációjából áll, *fully qualified name*, rövidítve FQN).

A Simulink-beli modellnek a `SimulinkModel`, míg a blokknak a `Block` EClass feleltethető meg. Egy `SimulinkModel` típusú objektum `Block` példányokat tartalmazhat, ezt fejezi ki a `contains` reláció.

Mivel egy blokk is tartalmazhat alblokkokat, ezért ennek kifejezésére a `Block` objektumok között lévő gyermek-szülő viszonyt jelentő `parent` és `subBlock` referenciák állíthatók be.

A modell lehetővé teszi tetszőleges blokk paraméterek tárolását. Erre azért van szükség, mert Simulink elemek is csak a paramétereiknek köszönhetően lesznek szakterület specifikusak, és ezzel kapcsolatos információk sok esetben nem elhanyagolhatók. A `properties` referencia olyan objektumokra mutat, amikben tetszőleges Simulink blokk paraméter név szerint, szöveges formában megőrizhető. Továbbá ha van információ az adat típusáról, akkor az is megadható.

A `Block` egy specializációja a `VirtualBlock`, aminek a szimuláció során nincs szerepe, csak a modell képét egyszerűsítik. Ebből származik le egy `PortBlock` absztrakt EClass. A port blokkok szükségesek a többszintű modellen belül, a szintek között futó jelek helyes továbbításához. Minden olyan blokk, mely nem atomi és van legalább egy portja, az kell hogy tartalmazzon egy port blokkot, mint alblokk. A port blokk összeköttetésben áll a porttal, a blokk belsejében a portba befutó vagy onnan kimenő jeleket továbbítja. A metamodellen szereplő `Port` és `PortBlock` közötti kapcsolat ezt írja le. A kimeneti portokhoz az `OutPortBlock`, a bemeneti portokhoz az `InPortBlock` Ecore osztályok egy-egy példányt rendeljük.

`Port` objektumokra tartalmazhat referenciát egy `Block` példány, erre szolgálnak az `outPorts` és `inPorts` kapcsolatok, de összeköttetésre közvetlen nem. Mivel az összeköttetések portok között futnak, és egy kimenő jel több bemeneti porthoz is csatlakozhat, így ennek figyelembe vételével két Ecore osztályt, a `SingleConnection`-t és a `MultiConnection`-t kell leszármaztatni az absztrakt `Connection` EClass-ból. A `SingleConnection` objektumok reprezentálják a portok közötti 1-1 kapcsolatokat, míg a `MultiConnection` példányok több, ugyan ahhoz az `OutPort`-hoz kötődő `SingleConnection` példányt fognak egybe, ezáltal 1-több kapcsolatot megvalósítva.

4.1.2. Programozott kommunikáció a Matlabbal

A Matlab parancssori felületén keresztül minden olyan funkció is elérhető, melyhez a program grafikus felületet biztosít. Ez alól a Simulink-ben elérhető funkciók sem kivételek, így modelleket akár tisztán *parancssori utasításokkal (command)* létre lehet hozni, amik később a beépített modellszerkesztőben is hozzáférhetők. Parancssori utasításokkal minden paraméter le is kérdezhető (és ha engedélyezett a módosítás, akkor beállítható), akár olyan is, ami a grafikus felületen nem érhető el.

A parancsok ezen (a Matlab keretei között) szinte korlátlan lehetőségét kihasználva a Matlab valamely alkalmas interfészén [12] keresztül parancsok kiadásával lehetővé válik a programozott kommunikáció. Ehhez kapcsolódóan jelentkezik az első megoldandó probléma, hogy milyen módon kell kezelni azokat a parancsokat, amik valamilyen visszatérési

értéket szolgáltatnak. Az ilyen parancsok „eredményének” általában valamilyen szöveges reprezentációját kapja vissza a parancsot kiadó fél, amiből sem a jelentésére, sem a hozzá kapcsolódó objektumra nem lehet következtetni. Annak érdekében, hogy létrejöhessen hatékony kommunikáció, ismerni kell az egyes parancsokra válaszként adható adatok belső felépítését a hibamentes értelmezéshez. Ez a parancsot kiadó oldal felelőssége, hogy a visszatérési értékből előállítsa a hasznos információt.

Mivel jelen esetben a cél egy objektum-orientált modell megalkotása egy másik keretrendszerben a hatékony validáció végett, így tipikusan a modellek létrehozásával és szerkesztésével kapcsolatos parancsok és az azokra adott válaszok ismerete szükséges.

A Simulink modellezésre használható minden eleme egyértelműen azonosítható az *teljes nevével (FQN)*, illetve a memóriába betöltött elemek az *egyedi leírójukkal (handle)* is. A teljes név egy perzisztens karaktersorozat, míg az egyedi leíró valamilyen valós érték, ami minden betöltéskor rendelődik az adott elemhez.

Ezen ismeretek birtokában Simulink-beli könyvtárak és modellek lekérdezésére az alábbi algoritmus alkalmazható:

1. Blokkok bejárása:
   I. a bejárni kívánt modell név szerinti azonosítása és betöltése a Matlab környezetbe
   II. minden olyan blokk nevének lekérése és azonosítása, melyet az adott modell közvetlen tartalmaz
   III. minden blokkra (hasonlóan a fenti lépéshez) az összes közvetlen tartalmazott alblokk nevének lekérése
   IV. a fenti lépést rekurzív ismétlése az alblokkokra, amíg az éppen vizsgált blokk nem tartalmaz további alblokkot
2. Portok felvétele a blokkokhoz: ez egyszerűen megtehető, mivel a blokkok az 1. lépés után már ismertek. Egy tetszőleges sorrendben végigiterálva a blokkokon a portjaikat lekérdezve a portok bejárása és azonosítása.
3. Kapcsolatok létrehozása az összeköttetésben lévő elemek között: mivel a portok a 2. lépés után ismertek, így minden kimeneti portot egy tetszőleges sorrendben végignézve, az adott kimeneti portból induló összes kapcsolat felvétele.

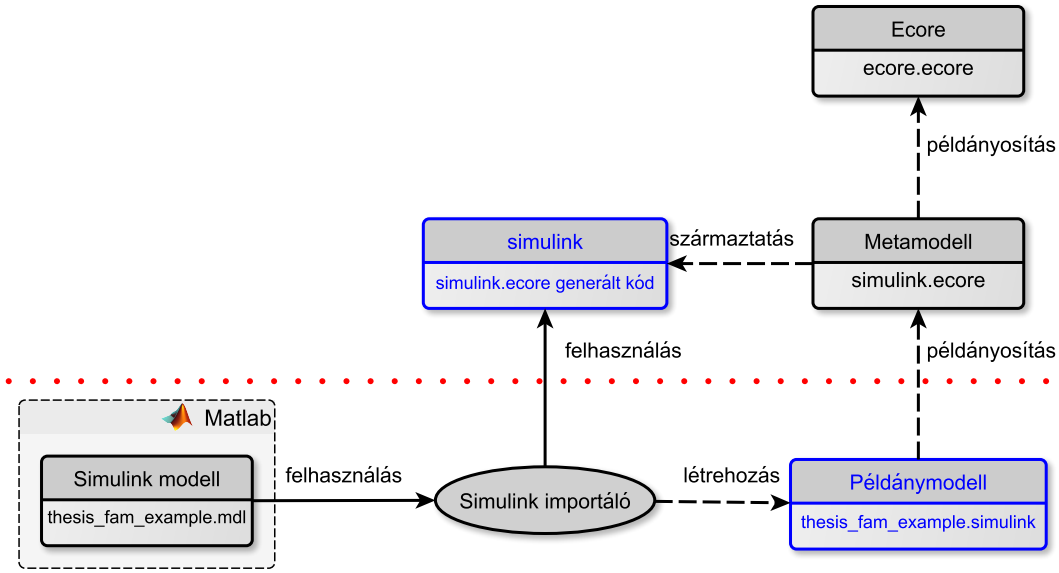
Az algoritmus végrehajtása során az elemek azonosításával egyidőben létre kell hozni objektumokat és beállítani a megfelelő kapcsolatokat.

4.1.3. Simulink modell importálása

Az EMF reprezentáció létrehozásához szükséges elemek közé tartozik a Simulink metamodellből generált kód és egy bejárást végző modul. A modellhez tartozó kód automatikusan generálható az EMF beépített eszközeinek segítségével. A bejáró implementálásakor a 4.1.2. fejezetben felsorolt lépések megvalósítása szükséges, melyek ez esetben magukba

foglalják a generált kód hasznosítását. A bejárás eredményeképp egy Simulink példánymodell jön létre.

Példa: A 2.2. fejezet mintapéldájához tartalmazó *thesis_fam_example.mdl* fájlban tárolt modell importálásához felrajzolt blokkvázlat a 4.1. ábrán látható. Az ábra bal oldalán található elemek az előfeltételei az importálásnak. A *Simulink metamodellből generált kódot* használja a *Simulink importáló*, ami a MATLAB-bal kommunikálva *bejárja a* SIMULINK *modellt*. A bejárás eredményeképp létrehozza az *thesis_fam_example.simulink* modellt, a *Simulink metamodell* egy példányát. A folyamatról készített ábrán a pontozott vonal feletti rész fejlesztési időben, míg a pontok alatti részek lépései futás időben kerülnek megvalósításra. A jobb oldalon a modellek a metaszinteknek megfelelő sorrendben helyezkednek el.



4.1. ábra. SIMULINK*-beli modell EMF-be történő importálásának vázlata*

4.2. Validáció

Az importált *Simulink példánymodellek* feletti közvetlen szabálykiértékelésre az EMF-INCQUERY ad lehetőséget. Minták megfogalmazásával kell azokat az eseteket vizsgálni, amik egy helyesen felépített modell esetén nem fordulhatnak elő. Az ilyen hibás eseteket leíró minták illeszkedésekor elvárás, hogy a rendszer hibát jelezzen.

Példa: A korábban már bemutatott 2.6. ábrán szereplő minta egy példa a szabályok szerint helytelenül használt *goto* blokkra.

EMF Simulink példánymodell Az EMF-be importált, Simulink modellek alapján létrehozott Simulink modellekre illeszthetők minták validáció céljából. Minden Simulink modell megpróbálja minél pontosabban, a Simulink-beli jellemzőkhöz ragaszkodva leírni az eredeti modelleket. Ebből következik, hogy az EMF reprezentációban minden olyan információ jelen van a modell struktúrájára vonatkozóan, ami alapján eldönthető a modell helyessége.

4.3. Visszajelzés a validáció eredményéről

Az EMF modellen végzett validáció eredményét vissza kell jelezni a felhasználónak. Lehetőség szerint azon a felületen kell ezt megtenni, ahol a modellek szerkesztésére módunk van, jelen esetben a Simulink rendszerében az eredeti modellen kell jelezni a hibát. Fontos a visszajelzés esetén az is, hogy minél pontosabban lehessen a hibát az eredeti helyén megmutatni.

Minden `Block` típusú elem tárolja a Simulink-beli azonosíthatósághoz elégséges információt (teljes nevet). Ez alapján a Matlab-nak kiadhatók parancsok, amik végrehajtják az értesítést és kijelölik az érintett blokkot vagy blokkokat.

Példa: A 2.6. ábrán szereplő példa minta felhasználásakor a mintának megfelelő minden `Goto` típusú EMF modellobjektumhoz meg kell keresni, melyik a neki megfeleltethető *goto* az eredeti Simulink modellben.

4.4. Megvalósítási részletek

A következőkben bemutatásra kerülnek a modellvalidáció fontosabb implementációs lépései. Ennek érdekében a 4.4.1. fejezet összefoglalja mik a validáció ezúton történő elvégzésének technikai feltételei, míg a 4.4.2. fejezet leírja, hogyan kell azt kivitelezni és az eredményt visszajelezni a felhasználónak.

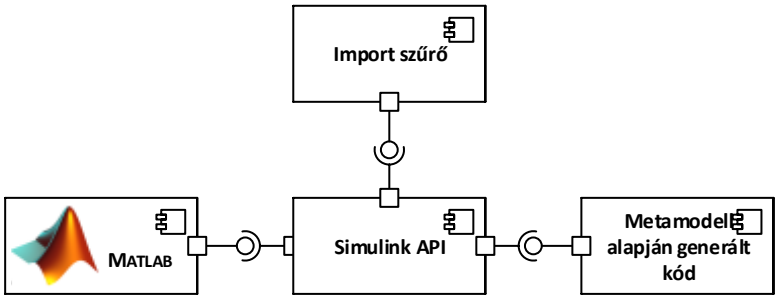
4.4.1. A validáció előkészületei

A Simulink modellek validációjának lehetővé tételéhez a Simulink metamodel gondos megtervezésén kívül az alábbi két lépés volt szükséges:

- Kommunikáció a Matlab-bal.
- Simulink modell importálása, EMF-beli megjelenítése.

Az elvégzett munka során elkészült, illetve felhasznált szoftverkomponensekről és kapcsolatukról ad áttekintést a 4.2. ábra.

Elsőként a Matlab-bal történő kommunikációt kellett megoldani. Erre a Matlab Java interfészét használtuk, mivel később is Java alapú technológiák kerültek felhasználásra.



4.2. ábra. *A modellek importálásához használt komponensek és kapcsolataik*

Ehhez elérhető egy MatlabControl [10] nevű, Java nyelvet használó, a kommunikációt csomagoló programozói interfész, amit az egyik egyetem oktatói kezdtek el fejleszteni abból a célból, hogy MATLAB-ot alkalmazó géptermi gyakorlatokat könnyen elő lehessen készíteni. Azóta a csomag ennél jóval többet tud, és a fejlesztő bárkinek a rendelkezésére bocsájtja.

A MatlabControl hiányosságai és konfigurációs nehézségei zavarták egy projekt kapcsán együttműködő repülőgépipari partnerünket. Fő hibája az volt, hogy nem tudtak futó MATLAB példányhoz csatlakozni az eszközzel. Ennek következtében a cég elkészített egy Java RMI technológiát használó saját, célzottan a MATLAB kommunikációra alkalmas komponenst. Ezt követően a SIMULINK modellimportáló szoftverhez a MATLAB-hoz való kapcsolódást lehetővé tevő különböző megoldások támogatására definiáltunk egy *parancs végrehajtó interfészt* és elkészítettünk egy *kiterjesztési pontot*. Ez alkalmas arra, hogy a későbbiekben a kommunikációt megvalósító programmodul az igényeknek és használati módoknak megfelelően cserélhető legyen.

Az elkészített API lehetővé tette alapvető MATLAB parancsok kiértékelését. A használata bizonyos szempontból nehézkes, mivel egyszerű szövegként kell megadni a MATLAB saját nyelvén a kívánt utasítást. Cserébe segítségével tetszőleges parancs kiadható, ezért ezt felhasználva, e szakdolgozathoz kapcsolódó munka keretében elkészült egy erre épülő, újabb csomagoló osztály, ami a jelen kontextusban használt MATLAB parancsokat megfelelően paraméterezve kiadja. A készített API már sok esetben típushelyesen kezeli a visszaadott értékeket, de a kivételes esetekre felkészülve továbbra is támogatja a MATLAB nyelvén szöveges formában leírt parancsok kiadását, és ezekre esetlegesen válaszul kapott tisztán szöveg vagy valós szám formátumú adatok fogadását is.

Az így elkészült, *Simulink API*-nak nevezett komponenst, és az EMF eszközeinek segítségével generált kódot felhasználva Java nyelven implementáltuk a 4.1.2. fejezetben leírt algoritmust. Erre a célra egy külön osztály került megvalósításra, amely bejáró függvénye létrehozza egy adott modellhez a neki megfelelő Simulink példánymodelljét.

Az importáláshoz opcionálisan megadható, hogy a vizsgált blokkok közül mik kerüljenek be a Simulink modellbe. Erre azért lehet szükség, mert egy FAM esetén előfordulhat, hogy egy Leaf funkció alrendszerként van reprezentálva, de a belső felépítése a funkciók szempontjából lényegtelen. Ekkor az importált modellben kívánatos lehet, hogy csak az adott funkciót reprezentáló alrendszer jelenjen meg, és a tartalmazott blokkok már ne kerülje-

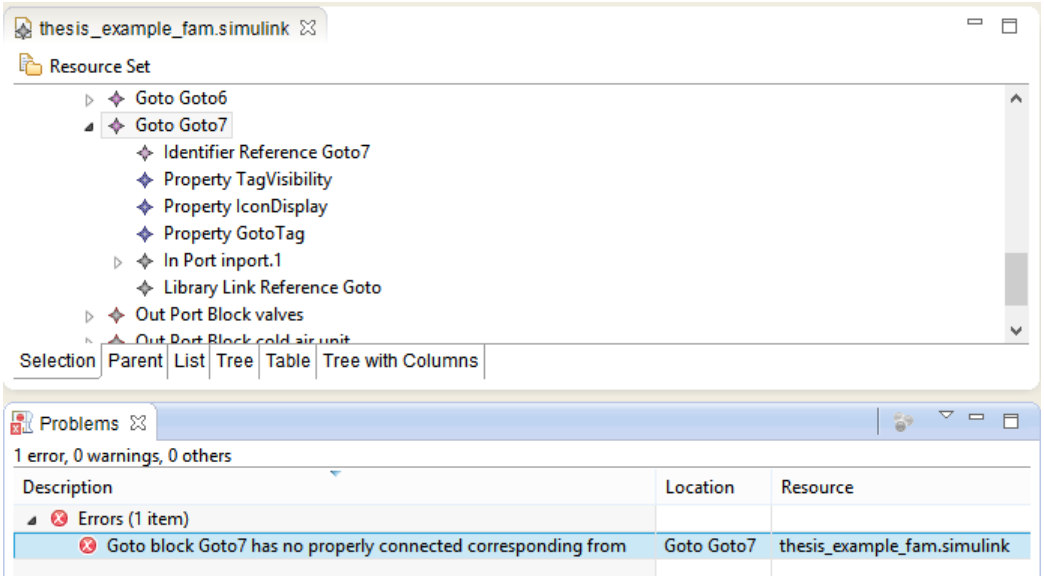
nek bele feleslegesen a modellbe. Az ezt lehetővé tevő komponens neve az *Import szűrő*, ami a Simulink API Eclipse plug-in által definiált kiterjesztési ponthoz (extension point) valósít meg egy kiterjesztést. Ez a megoldás kihasználja az Elipse plug-in struktúrájának modularitását, sokféle felhasználást és könnyű módosíthatóságot eredményez.

A feladat megoldásához szükséges volt a Simulink rendszerek reprezentációját lehetővé tevő metamodell megalkotása. Ezt a metamodellt és a belőle generált kódot jelenti a 4.2. ábrán szemléltetett komponens diagramon a *Metamodell alapján generált kód* nevű komponens.

4.4.2. Validáció és visszajelzés megvalósítása

Az EMF-beli modellekre validációs szabályokat EMF-IncQuery segítségével lehetett megfogalmazni. A megfelelő, hibás esetekre illeszkedő mintákat a `@Constraint` annotációval ellátva az eszköz ezek alapján a validációt végző kódot legenerálja. Az annotáció paramétereként megadható, hogy az adott minta illeszkedése esetén milyen súlyos a probléma (severity), a konkrét illeszkedés melyik elemét tekintsük hibásnak (location), illetve hogy mi legyen a hibaüzenet.

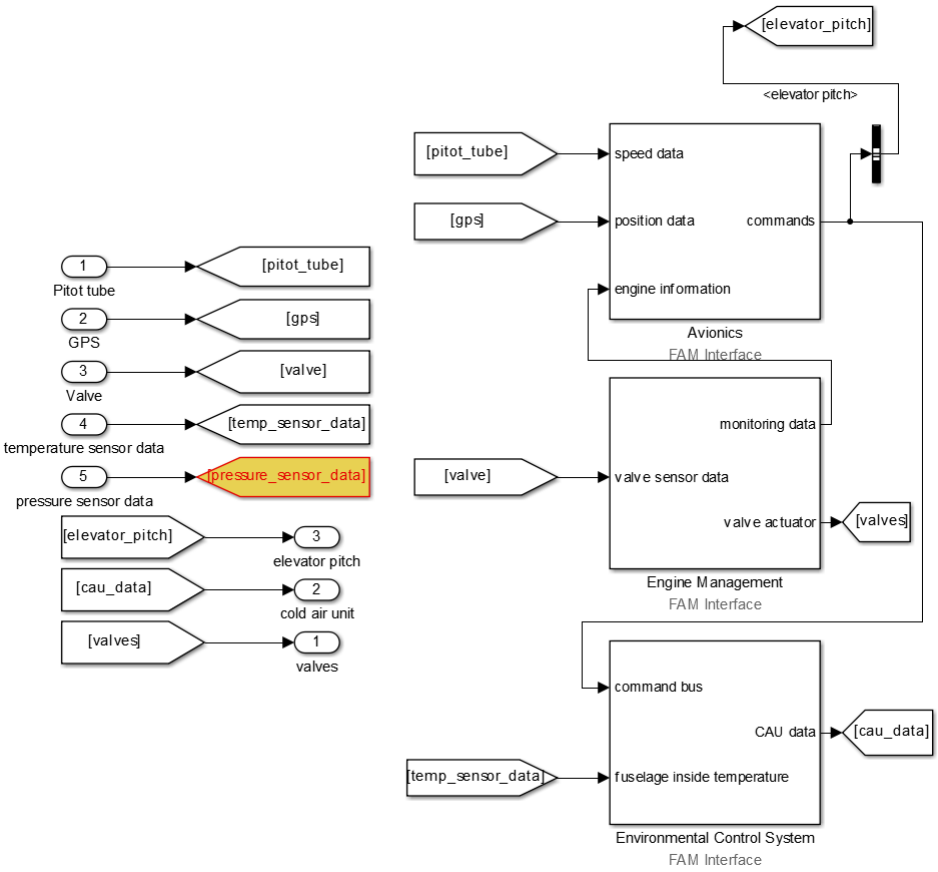
Példa: Egy modellre a 2.7. ábrán EMF-IncQuery-ben, és a 2.6. ábrán grafikusan definiált `notWellformedGoto` mintát illesztettük. A hibásan szereplő `Goto`-ról készített kép és a hibajelzés az EMF-es modellszerkesztő felületén a 4.3. ábrán szerepel. A hibajelzés oka, hogy nem található hozzá helyesen bekötött `From` típusú elem. Azért vesszük ezt hibásnak, mert nem lehet eldönteni ebben az esetben, hogy a modell tervezője szándékosan nem vezette tovább a goto blokkba befutó jelet, vagy hibát követett el, és emiatt nem rendelt hozzá egyetlen from blokkot sem.



4.3. ábra. *Az EMF modellszerkesztő és a hibajelzés*

A hibajelzés hatására a Simulink felhasználói felületén is láthatóvá válik a hibásan elhelyezett blokk, amit egy automatikusan a Matlab-nak kiadott parancs vált ki.

Példa: A Simulink felületén kapott hibajelzést a 4.4. ábra mutatja be. A példában szereplő modellben a legfelső szinten elhelyezett *pressure sensor data* inportból érkező adatok egy goto blokkba futnak, azonban ezek nem továbbítódnak sehova from blokk hiányában. Az ilyen és ehhez hasonló esetekről nem lehet egyértelműen eldönteni, hogy milyen oknál fogva szerepel a modellen az adott elem, így erről a felhasználót tájékoztatni kell.



4.4. ábra. *A Simulink grafikus felületén kiemelve szerepel a hibásan használt blokk*

5. fejezet

Modell alapú validáció kódgenerálás segítségével

A 4. fejezet az EMF Simulink példánymodellek segítségével végzett validációs módszert mutatta be részletesen. Ezen fejezet célja pedig, hogy egy alternatív, a validáció elvégzéséhez túlnyomó részt MATLAB-ot használó megoldást vizsgál meg részletesen, melyhez hasonló módszer ugyanezen technológiák felhasználásával eddig még tudtunkkal nem készült. Az 5.1. fejezetben a gráfminták MATLAB függvényekre való leképezésének lehetőségét mutatjuk be, majd az 5.2. fejezet foglalkozik a megvalósítás sajátosságainak és részleteinek tárgyalásával.

5.1. Gráfminták lefordítása Matlab függvényekre

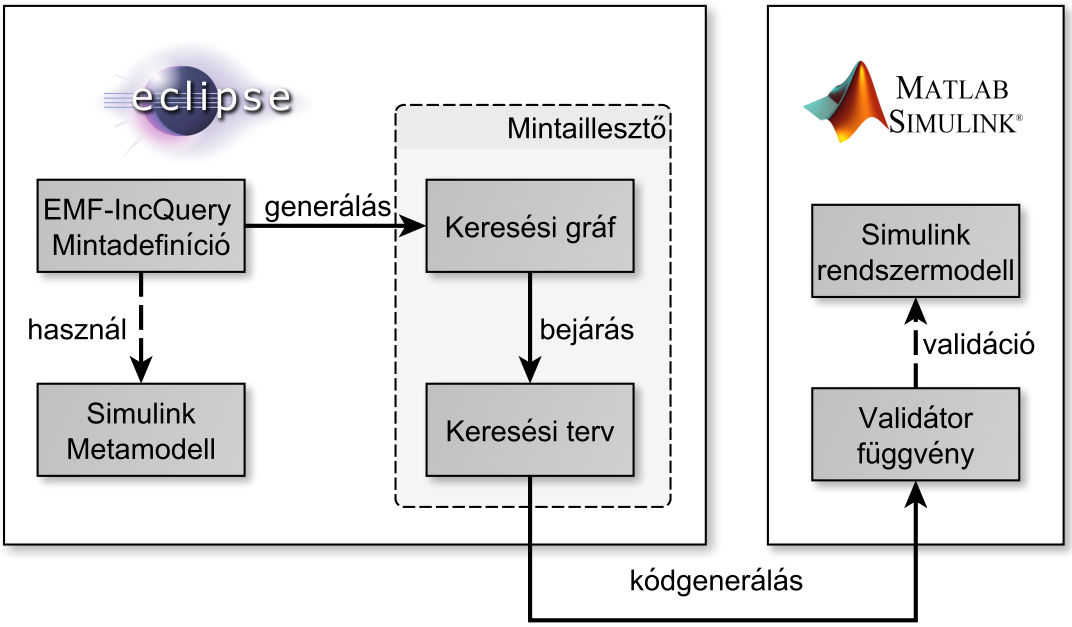
Célunk, hogy az EMF-INCQUERY segítségével definiált szabályok kiértékelését MATLAB környezetben is el tudjuk végezni. Ennek megvalósítása érdekében a szabályokat leíró gráfminták lefordítását választottuk, ahol a fordítás eredményeként a validációt elvégző MATLAB függvényeket kapjuk. Ezt követően egy ilyen függvények a vizsgálandó SIMULINK rendszert paraméterül adva a validációt elvégzi, és a visszatérési értékében megadja a szabályt megsértő modellelemeket. A folyamatot az 5.1. ábra szemlélteti.

5.1.1. Mintaillesztés alapjai

Annak érdekében, hogy a mintákból sikeresen lehessen MATLAB-ban futtatható kódot generálni, meg kell ismerni a minták kiértékelésének módját. A dolgozatban bemutatott módszerek alapjául a hivatkozott irodalomból a [8] szolgált. Ennek megfelelően két alapvető megoldási módszer van minták illesztésére.

Inkrementális illesztési módszer

A módszer lényege, hogy az összes definiált mintához tartozó kezdeti illeszkedéseket a rendszer indulási fázisában valamilyen úton kiszámolja és eltárolja. Ezt követően a modell változásokat figyelve, a változások alapján módosítja az illeszkedések halmazát. Ezzel a megoldással megtakarítja minden modellváltozás után azt a lépést, hogy az egész modellre



5.1. ábra. *A függvények generálásának folyamata*

újra illeszteni kelljen a mintákat, elég csak a megváltoztatásokat figyelembe venni, aminek köszönhetően sok esetben gyorsabb lesz az új illeszkedéshalmaz elemeinek meghatározása.

A módszer azonban nem lesz hatékony olyan esetekben, ahol a modell változtatott részének nagysága összemérhető a teljes modell méretével, hiszen ekkor olyan, mintha az indulási fázisba kerülne az illesztés, és ezzel azt az előnyét veszítené el a módszer, amiért alkalmazzák.

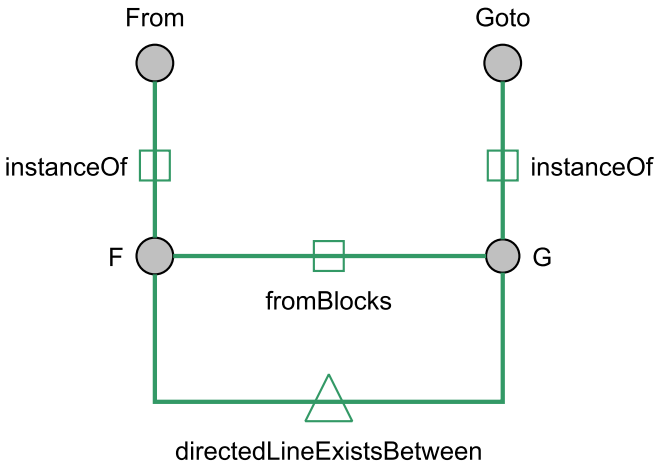
Ezt a megoldást használja az EMF-IncQuery keretrendszer is. A szakdolgozat keretében készített kódgenerátorhoz azonban a másik, szintén az 5.1.1. fejezetben ismertetett módszer használatát találtuk alkalmasabbnak.

Local search alapú mintaillesztés

Az LS módszertanát alkalmazó algoritmusoknál a mintaillesztés első lépése, hogy egy *keresési gráfot* (angolul *search graph*) készítenek el a mintához. Ez a gráf nem más, a minta egy olyan reprezentációja, ahol a mintában szereplő változók és a rájuk vonatkozó megkötések közösen szerepelnek. Ez a gráf valójában egy hipergráf, ahol a csúcsok a változókat, az élek pedig a rajtuk teljesülő predikátumokat, megkötéseket jelentik.

Példa: A `wellformedGoto` mintához rendelt keresési gráf az 5.2. ábrán szerepel. Az `F` és `G` jelentik a mintában szereplő változókat. Az élek az egyes megkötéseket szimbolizálják. A gráf kifejezi, hogy a `Goto` típusnak példánya, az `F`-fel `fromBlocks` kapcsolatban áll és a `(G,F)` párosra illeszkedik a `directedLineExistsBetween` minta. Ehhez hasonlóan `F`-re igaz, hogy `From` példánya, és `G`-vel az előbb említett kapcsolatban áll.

Ezt követően a keresési gráfhoz az illesztő algoritmus elkészíti a *keresési tervet* (angolul

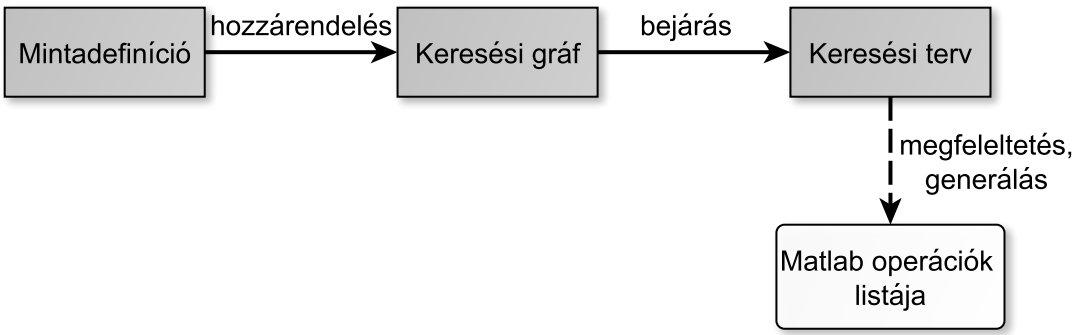


5.2. ábra. *A példában szereplő* `wellformedGoto` *minta keresési gráfja*

search plan). Ez nem más, mint *operációk sorrendezett listája*, amit a keresési gráf egy lehetséges bejárásával kapunk. A benne szereplő műveleteket az adott sorrend szerint minden lehetséges esetre alkalmazva kapjuk az illeszkedések halmazát.

5.1.2. Matlab függvény generálása

A validációt végző MATLAB függvény előállításához a keresési terv szolgál alapul. Ez adja meg a változók behelyettesítésének vagy lekötésének helyét, illetve az egyes szabályok alkalmazásának sorrendjét. Felhasználásával egy elkészített kódgenerátor az EMF-beli Simulink modellek felett megfogalmazott szabályokhoz MATLAB operációk listáját rendeli, majd előállítja a futtatható függvényt. Ehhez szükség volt egy megfeleltetés elkészítésére, amivel meg lehetett adni, melyik EMF-beli megkötéshez milyen MATLAB operáció tartozzon. Ezt az EMF Simulink metamodell alapján, egy iteratív fejlesztési és bővítési folyamatban tettük meg. A kódgenerálás menetének vázlatát az 5.3. ábra mutatja.



5.3. ábra. *A minták alapján* MATLAB *függvény generálása*

Példa: A korábbiakban példák során bemutatott 2.7. ábrán szereplő mintadefiníciókhoz tartozó, egy-egy lehetséges keresési terv szerepel az 5.1. és az 5.2. táblázatokban. A könnyebb érthetőség kedvéért ezen keresztül mutatjuk be a keresési terv kiértékelésének főbb lépéseit.

5.1. táblázat. *Search plan a helyes gotokra illeszkedő mintához*

	Operáció	típusa
1:	G egy `Goto` példány	ellenőrzés
2:	F a `fromBlocks` eleme	kiterjesztés
3:	F egy `From` példány	ellenőrzés
4:	`directedLine` hívás	ellenőrzés

5.2. táblázat. *Search plan a helytelen gotokra illeszkedő mintához*

	Operáció	típusa
1:	Goto egy `Goto` példány	ellenőrzés
2:	neg `wellformedGoto` hívás	ellenőrzés

A `notWellformedGoto` minta a `wellformedGoto` mintát használja, így az utóbbit mutatjuk be hamarabb. A keresési terv kiértékelése kezdetén a minta paraméterébe valamilyen objektum behelyettesítődik. Első lépés, hogy *ellenőrizzük* (az angol szakirodalomban az ilyen lépést nevezik *check operation*-nek), hogy egy `Goto` típusú objektumról van-e szó, hiszen a minta célja, hogy meghatározza a modellben azokat a *goto* blokkokat reprezentáló elemeket, amik helyesen szerepelnek.

Ha az objektum típusa nem `Goto`, akkor a keresést egy másik elem a Goto változóba történő behelyettesítésével folytatja és számon tartja, hogy az előbb próbált elemre a minta nem illeszkedik.

Amennyiben viszont a feltétel teljesül, a kiértékelés folytatódik. A következő lépésben a keresés kiválasztja azokat a szóba kerülő elemeket, melyek az adott gotoval `fromBlocks` kapcsolatban állnak. Az ilyen módon működő műveleteket nevezik kiterjesztésnek (angolul *extend operation*), hiszen a művelet végrehajtása során a minta rész-illeszkedéséhez újabb elemek kerülnek, amikre meg kell nézni, hogy a további, rájuk vonatkozó feltételeket teljesítik-e.

Ezt követi a kiterjesztés során szóberült `F` típusának ellenőrzése. A mostani példában a modell felépítéséből adódóan tudjuk, hogy a `fromBlocks` kapcsolatban egy `Goto` példánnyal csak `From` típusú elemek állhatnak, így az `F` típusának ellenőrzését el is lehetne hagyni.

További feltétel a `directedLineExistsBetween` kétparaméteres minta illesztése vagy másnéven hívása két, konkrét értékkel rendelkező változóra. Amennyiben a `(G,F)` változópárosra a hívott minta illeszkedik, a keresés visszaad egy találatot. Egy találat után a keresést általában nem lehet abbahagyni, mivel a cél az összes lehetséges illeszkedés megkeresése.

A `directedLineExistsBetween` minta szemantikája, hogy a paraméterlistájában szereplő 1. paraméteréből van közvetlen irányított összeköttetés a modellben a 2. paraméterként kapott blokkba. A blokkok közötti összeköttetés létezése a felépített EMF Simulink metamodell sajátosságai miatt csak nehézkesen írható le, így célszerű egy külön mintába kiemelni. Ezt azért is érdemes megtenni, mert a minta áttekinthetőségét növeli, illetve a

minta alapján történő kódgenerálásnál lehetőséget ad optimalizációra. Az általunk megírt és funkcionálisan helyes EMF-IncQuery definíciója az emlegetett mintának az 5.4. ábrán olvasható. A példában helytelen *goto* blokkok megkeresése pedig helyes elemeket

```
pattern directedLineExistsBetween(SrcBlock : Block, DstBlock : Block) {
    Block.outports.connection(SrcBlock, C);
    Block.inports.connection(DstBlock, C);
} or {
    Block.inports.connection(DstBlock, C);
    MultiConnection.connections(MC, C);
    Block.outports.connection(SrcBlock, MC);
}
```

5.4. ábra. *A `directedLineExistsBetween` minta egy lehetésges definíciója*

visszaadó minta felhasználásával történik. Az első lépése a `notWellformedGoto` mintához rendelt search plannek, hogy a paraméter típusellenőrzését elvégzi. Ha ez megfelel, akkor egy negatív alkalmazási feltétel segítségével, az előzőekben tárgyalt `wellformedGoto` mintát, mint predikátumot felhasználja. A negatív alkalmazási feltétel jelentése, hogy a megkötés akkor teljesül, ha a *neg* feltételben szereplő minta nem illeszkedik a megadott paraméterekre. Végeredményeképp tehát azok a blokkok lesznek a minta illeszkedései, melyek `Goto` típusúak és nem felelnek meg a helyes *goto* blokkokat definiáló mintának.

Az előzőekben részletesen bemutatott keresési tervet ezt követően átadjuk a kódgenerátornak, aki a megfelelő Matlab parancsokat beilleszti a generált kódba. A generátorban található logika minimális, a feladata mindössze az, hogy a keresési tervben szereplő operációknak megfeleltetett Matlab hívásokat elhelyezze a kódban. A példában tárgyalt mintákhoz kapott kód alább olvasható.

Példa: A `wellformedGoto`-hoz generált kód egyes lépései:

- Az első lépés, hogy a minta paraméterében szereplő `G` összes potenciális értékét meghatározzuk az 5. sorban. Ennek érdekében azokat a blokkokat keressük ki a paraméterül kapott modellből, mely `BlockType` paramétere `Goto`. Ezek közül a gotok közül kerül majd ki minden lehetséges illeszkedés.
- A 7. sorban kiválasztunk egy sorrakerülő goto-t, majd a 8. sorban ehhez kikeressük a rendszerben található, hozzátartozó fromokat.
- Minden így megtalált from közül egyszerre egyet kiválasztva ellenőrizzük a blokk típusát (10-11. sorok).
- Ha a típus helyes, akkor vizsgáljuk, hogy van-e az aktuális fromhoz kapcsolódó blokk. Ha a visszaadott kapcsolódó blokkok listája nem üres, akkor találtunk egy illeszkedést, azaz az aktuális goto bekerül az eredményhalmazba (13-17. sorok).

wellformedGoto.m

```
1 function returnValue = wellformedGoto( system )
2 %WELLFORMEDGOTO the generated matcher code for the pattern named wellformedGoto
3 % The code is geneated from the corresponding EMF-IncQuery query definition
```

```
	returnValue = 'no matches';
	G = find_system_prepared(system,'BlockType','Goto');
	for j2 = 1:size(G)
		G_element = getListElement(G,j2);
		F = gotoFroms(system,G_element,'');
		for j3 = 1:size(F)
			F_element = getListElement(F,j3);
			if(strcmp(get_param(F_element,'BlockType'),'From'))
				if(~isempty(connecting(system,F_element,'')))
					if(~exist('matches','var'))
						matches = [{getfullname(G_element)}];
					else
						matches = vertcat(matches ,{getfullname(G_element)});
					end
				end
			end
		end
	end
	if(exist('matches','var'))
		returnValue = matches;
	end
end
```

A `notWellformedGoto` kódja a fentivel analóg módon végiggondolható, azzal a különbséggel, hogy a `@Constraint` mintaannotáció miatt a végére bekerül egy visszajelzést megvalósító kódrészlet is, mely automatikusan kijelöli a hibás elemet a SIMULINK szerkesztőfelületén.

notWellformedGoto.m

```
function returnValue = notWellformedGoto( system )
%NOTWELLFORMEDGOTO the generated matcher code for the pattern named notWellformedGoto
% The code is genearted from the corresponding EMF-IncQuery query definition
	returnValue = 'no matches';
	G = find_system_prepared(system,'BlockType','Goto');
	for j4 = 1:size(G)
		G_element = getListElement(G,j4);
		if(length(setdiff(getfullname(G_element),wellformedGoto(system))))
			if(~exist('matches','var'))
				matches = [{getfullname(G_element)}];
			else
				matches = vertcat(matches ,{getfullname(G_element)});
			end
		end
	end
	if(exist('matches','var'))
		returnValue = matches;
		% This is the generated code for error indication on the graphical editor
		% This part is generated because the pattern was supplied with @Constraint
		for j_final=1:length(returnValue)
		returnValueCurrent = getListElement(returnValue,j_final);
			for(jj_final=1:length(returnValueCurrent))
				actual = getfullname(getListElement(returnValueCurrent,jj_final));
				hilite_system(actual,'error');
			end
		end
	end
end
```

5.2. A megvalósítás részletei

A megvalósításhoz használt technológiák közé tartozik az EMF-IncQuery, amely egy kényelmes szöveges szerkesztőt ad a felhasznált metamodell ismeretében a felette megfogalmazott minták definíciójára. A mintadefiníciók alapján előállított keresési tervet egy Xtendes [17] sablon alapú kódgenerátorral fordítjuk le Matlab függvényekre. Emellett természetesen a Matlab rendszert is használtuk, a legenerált mintaillesztő függvények futtatására.

A mostani megoldásunk a keresési tervet egyelőre a megkötések egy egyszerű sorrendezésével állítja elő. Emellett bizonyos fokú optimalizálást jelentett, hogy a gyakran előforduló megkötéseket, illetve az EMF-IncQuery nyelvén bonyolultan megfogalmazott gyakori mintákat különálló Matlab szkriptekbe emeltük ki, és elláttuk a kódgenerátort az ilyen esetekre vonatkozó plusz információval. Ezáltal a generátor képes felismerni azokat a helyzeteket, amikor a kódba az adott megkötés lefordításához a megfelelő speciális függvény hívását kell elhelyeznie. Ezek a függvények szemantikailag ugyan azt az eredményt szolgáltatják, mint a mintában szereplő megkötés, viszont Matlab-ban hatékonyan hajtható végre. Ezt a megoldás során a következőkben felsorolt konkrét esetekre alkalmaztuk.

5.2.1. Kivételes útvonalkifejezések

Az *útvonalkifejezés* szó az olyan megkötéseket jelenti a mintadefinícióban, amik két modellelem közötti viszonyt írnak le. A 2.7. ábrán szereplő `wellformedGoto` minta definíciójában ilyen a `Goto.fromBlocks` kifejezés. A kódgenerálás megvalósítása érdekében mindegyik ilyen kifejezéshez hozzá kell rendelni azt a beépített Matlab függvényt mely az adott útvolankifejezésnek megfelelő Simulink-beli kapcsolatot feltérképezi, azaz alkalmas a keresési tervben szereplő extend/check operációk elvégzésére. Ez a feladat gyakran a `get_param` vagy `find_system` beépített parancsokkal és azok megfelelő felparaméterezésével megoldható. Ezek használata röviden:

- `get_param`: egy modellelem paraméterének értékét kérdezi le, a keresett paraméter neve szövegesen adható meg a függvénynek
- `find_system`: egy rendszerben képes az elemek paraméterértékeke alapján a keresésre

Előfordul azonban, hogy egy EMF Simulink modellbeli referencia értéke valójában a Simulink-ben csak több függvényhívás segítségével kérdezhető le, és az éppen aktuális blokkattribútum értékeken alapszik. A modellek struktúrája szempontjából releváns kapcsolatokat importáláskor külön típusos EMF referenciákra *is* leképezzük, hogy a modellek EMF-IncQuery segítségével kényelmesebben kezelhetők legyenek.

Ebből adódik, hogy kódgenerálásnál viszont fel kell készülni azokra az esetekre is, amikor az ilyen jellegű útvonalkifejezéseket vissza kell alakítani a Simulink-ben alkalmazható ellenőrzésekre. Jellemzően néhány sor kódról van szó minden megoldás esetén, azonban ezek a megoldások már túl komplikáltak ahhoz, hogy érdemes legyen a kódgenerátor sablonját

az ilyen esetek kezelésére is felkészíteni. Ennél egyszerűbb a feladatot megoldó kódrészletet egy függvénybe kiemelni, és ezáltal a többi útvonalkifejezéssel azonosan kezelhetővé tenni az ilyen eseteket is. A fejlesztés során eddig az alábbi rendhagyó esetekre készítettük fel külön a kódgenerátort:

- `InPort.portBlock`: A Simulink modellek esetén csak akkor tartozik egy *port*hoz *port blokk*, ha a port egy *subsystem* portja. Mindazonáltal ha van is megfeleltetett port blokk a porthoz, csak a *portNumber* tulajdonságaik segítségével rendelhetők össze. Ehhez tehát el kellett készíteni egy függvényt, mely a különböző paraméterezések esetén képes a kapott porthoz a port blokk, vagy a megadott port blokkhoz port rendelésére ez alapján az attribútum alapján.

- `Goto.fromBlocks`: Nem minden adat számított adat lekérdezését teszi hivatalosan támogatottá a Simulink, ezek közé tartozik a *goto* blokkhoz rendelt *from* blokkok listája. A Matlab grafikus szerkesztőfelületén elérhető a lista fejlesztés közben, azonban nincs mód ennek programozott lekérdezésére. Ennek érdekében egy olyan függvényt valósítottam meg, mely a goto blokk attribútumainak függvényében megkeresi és azonosítja a hozzá tartozó from blokkokat.

Az említett eseteknél a kifejezések fordított irányát is támogatni kellet, ami a fent leírtakkal gyakorlatilag analóg módon történik.

5.2.2. Hatékony Matlab számítások

Van példa olyan esetre is, amikor az EMF-IncQuery megfogalmazás pontos, a keresési terv szerinti lefordításánál sokkal hatékonyabb megvalósítást jelent egy megfelelően felparaméterezett Matlab beépített függvény hívása. Az ilyen esetek észrevételéhez egyrészt ismerni kell a Matlab nyújtotta függvények alkalmazhatóságát, másrészről pedig a kódgenerátornak is lehetőséget kell adni az ilyen esetek egyszerű felismerésére. Ez utóbbira jelent megoldást, ha ezeket a gyakran alkalmazott részmintákat kiemeljük, és ahol szükség van az alkalmazásukra ott egy `find <minta neve>(<paraméterek>)` megkötést veszünk fel. Ezáltal az ilyen minták nevei és paraméterei alapján van lehetőség a generált kód hatékonyabbá tételére. Ezt a megoldást a következő esetekre alkalmazzuk:

- `directedLineExistsBetween`: A dolgozatban bemutatott példa kapcsán már korábban szerepelt ez a minta. Olyan blokk párosra illeszkedik, ahol a minta első paraméterében szereplő elemből indul ki a közvetlen, irányított összeköttetés a modellben, és a második paraméterként kapott elemben végződik. A minta EMF-IncQuery definíciójában szerepelnie kell a blokkoknak, a hozzájuk tartozó portoknak, illetve a közöttük futó kapcsolatnak. Ezt azonban Matlab-ban egy alkalmasan felparaméterezett `find_system` hívásra cserélve hatékonyabb kódot kapunk, mint a minta keresési tervének alapján történő lefordításával amellett, hogy a kifejezés szemantikája nem változik.

- `isAttributeValueEquals`: ez a minta használható modellelemek attribútumainak vizsgálatára. Az EMF Simulink metamodell lehetővé teszi egy blokk minden attri-

bútumának eltárolását szöveges formában egy map-szerű adatszerkezetben, azonban vannak köztük olyanok, amelyek a hatékony hozzáférés végett az EMF modellben külön EAttribute-okban tárolódnak. Ebből kifolyólag az az EMF-IncQuery minta, ami az értékek összehasonlítását végzi igen komplex, azonban erre a bonyolításra nincs szükség Simulink-ben: minden attribútum értékét a `get_param` paranccsal lehet vizsgálni. Ha az attribútumérték alapján kell egy extend műveletet végezni, akkor az Matlab-ban szintén egyszerűen, a `find_system` alkalmazásával történhet, ahol a keresőfeltétel az attribútumnév és érték.

5.2.3. Visszajelzés a validáció eredményéről

Mivel a validációt a Matlab rendszer végzi, így nem okoz gondot az eredmények megjelenítése: mindössze egy parancs kiadását kell megtenni a modellelem azonosítójával, és a 4.4. ábrán szereplő hibajelzéshez hasonló jelenik meg a modellen. Az ezt elvégző a parancsot a generátor a minta felett elhelyezett `@Constraint` annotáció alapján generálja. Ez látható a korábban ismertetett `notWellformedGoto` kódjában is.

6. fejezet

Értékelés

Annak érdekében, hogy a bemutatott módszerek használhatóságát összehasonlítsuk a szintetikus, példamodellek mellett lehetőségünk volt két, a repülőgépipari partnerünktől kapott, Simulink modellen is tesztelni a validáció hatékonyságát. Ezek egy civil repülőgép hardver architektúráját írják le. A szakdolgozat példáihoz 3 modellt használtam:

- A példák során szereplő szintetikus példa FAM-ot. Továbbiakban a *kicsi* névvel hivatkozom rá. Méretei:
  - 73 blokk
  - 53 összeköttetés
  - 224 elem összesen
- Egy kisebb ipari modellt. Továbbiakban a *közepes* névvel hivatkozom rá. Méretei:
  - 355 blokk
  - 330 összeköttetés
  - 1331 elem összesen
- Egy nagyobb ipari modellt. Továbbiakban a *nagy* névvel hivatkozom rá. Méretei:
  - 1406 blokk
  - 1734 összeköttetés
  - 6055 elem összesen

A validáció hatékonyságának mérésére a dolgozatban korábbi példaként használt `notWellformedGoto` mellett egy másik, mérési célokra kitalált minta illesztését is elvégeztem. Ennek a grafikus definíciója az F.2 függelékben szerepel, az EMF-IncQuery definíció alapján hozzá generált Matlab-ban futtatható kódot pedig az F.3 tartalmazza. Megalkotásakor arra törekedtünk, hogy minél többféle megkötést tartalmazzon, ezáltal pontosabb képet adjon a módszerek hatékonyságáról. A mérési adatok elemzésekor ennek megfelelően a mintáknak a *goto* és *teszt* nevet választottam.

A mérések elvégzéséhez egy Intel Core i5-2430M 2,4 GHz-es és 8 GB RAM-mal rendelkező gépet használtunk 64-bites Windows 8.1-gyel, Matlab R2012b és Eclipse 4.3-mal. Az alább bemutatott eredmények 10 független mérés átlagaként adódnak.

6.1. Validáció EMF modellek felett

Elsőként az EMF modellek feletti EMF-IncQuery-t használó validációs megoldás eredményeit ismertetem:

- **Importálás** A mérés során vizsgáltuk, hogy a blokkok számának függvényében milyen hosszú időt vesz igénybe az EMF reprezentáció elkészítése. Emellett mértük, hogy mennyi időt tölt az importálást végző modul a Matlab-ból érkező eredményekre és válaszokra való várakozással. A kapott e a 6.1. táblázat mutatja.

6.1. táblázat. *A modell méretének függvényében a Java-*Matlab *kommunikáció és a teljes import folyamat ideje*

Modell	Matlab kommunikáció	Importálási idő
kicsi	14,874 s	15,098 s
közepes	234,088 s	241,867 s
nagy	354,719 s	360,878 s

  Az importálás teljes idejéhez mérve a Matlab kommunikáció mindhárom esetben igen jelentős, hiszen ez az összidő több, mint 96%-át jelenti. A maradék időben az EMF modellek létrehozása és szerializálása történik.

- **Validáció** A validációs szabályoknak való megfelelés leellenőrzése az EMF-IncQuery segítségével az importálási időhöz képest szinte elhanyagolható. A mért adatok a 6.2. táblázatban szerepelnek. Ezek az EMF-IncQuery eddig ismert teljesítményét figyelembe véve[1] az elvárt eredménynek megfelelnek.

6.2. táblázat. *Az* EMF-IncQuery *validációs idők mérési eredményei*

Modell	*goto* minta	*teszt* minta
kicsi	0,042 s	0,047 s
közepes	0,185 s	0,212 s
nagy	0,258 s	0,266 s

A mérések eredményeiből az alábbi két fontos következtetést vonhatjuk le:

- A modellek importálási idejének mérésekor az derült ki, hogy a futási idő megközelítőleg 98%-át a Matlab-nak parancsok kiadása és onnan adatok fogadása teszi ki. Egy parancs kiértékelési ideje $0,25$ ms és 35 ms közé esik, és az azonos típusú parancsok esetén a kiértékelési idő a modell méretétől és a visszaadott értékektől független.

- Jól látható, hogy az egész folyamatra tekintve a modellimportálás igényli a legtöbb időt, maga a validáció szinte azonnali módon számolódik. Ebből is látszik, hogy további teljesítménynövekedés a modellimportálásának gyorsításával érhető el.

6.2. Validáció Matlabban, generált kód használatával

A méréseknél feltüntetett MATLAB-os futtatások eredményei mindig a modell betöltése utáni első futtatás eredményét jelenítik meg, mivel jelentős gyorsulás figyelhető meg minden nem első hívás alkalmával. Ennek oka az, hogy a MATLAB-ban bizonyos indexeket hoz létre a modellelemekhez, ami a későbbiekben növelheti függvények kiértékelési sebességét, ezáltal 2 nagyságrenddel is csökkenhet a futási idő.

A minták alapján kódgenerálást használó módszer esetén az eredmények az alábbiak szerint alakultak:

- **Kódgenerálás** A kódgenerátor időigénye nem számottevő, bonyolultabb (15-20 sor körüli) mintákat is kevesebb, mint 100 ms alatt legenerál. A *goto* lefordítása a mért adatok szerint 1 ms, míg a *teszt* mintához tartozó kód 82 ms alatt generálódott.
- **Validáció** MATLAB-ban futtattam le a méréshez alkalmazott mintákhoz elkészült szkriptet, a futási időket a 6.3. táblázat mutatja.

6.3. táblázat. *A* MATLAB*-os függvények lefutási idői*

Modell	*goto* függvény	*teszt* függvény
kicsi	0,990 s	0,414 s
közepes	11,874 s	44,650 s
nagy	27,182 s	991,431 s

6.3. A két módszer összehasonlítása

Az elvégzett mérések alapján az alábbi következtetésekre jutottunk:

- Annak eredményeképp, hogy az EMF-INCQUERY-ben megfogalmazott minták nem minden esetben képezhetők le MATLAB parancsokká, bizonyos dolgok (például tranzitív lezárás) csak EMF modellek felett vizsgálhatók.
- Egyszerű mintákra a MATLAB-ban végzett validáció hatékony.
- Komplex esetekben az EMF-be importálás és EMF-INCQUERY használata az egyetlen út a MATLAB-os validátorfüggvények exponenciálisan növő időigénye miatt.

Ezen egyszerű mérések bizonyították, hogy mindkét kidolgozott módszernek létjogosultsága van a modellméret és a komplexitás függvényében. Minden esetben azt érdemes alkalmazni, ami az igényeknek az adott esetben jobban megfelel.

7. fejezet

Összefoglalás és jövőbeni tervek

A MATLAB-SIMULINK általános célú modellezési és szimulációs eszköz széles körben alkalmazott beágyazott és biztonság-kritikus rendszerek tervezése és fejlesztése során. Ezen rendszerek tervezésekor sokszor ipari szabványokat és előírásokat is figyelembe kell venni, amelyek gyakran írnak elő jólformáltsági és strukturális kényszereket a rendszer komponenseire, amelyre a SIMULINK keretrendszer csak korlátozott támogatást nyújt.

Ehhez kapcsolódóan két megközelítést valósítottunk meg:

- Minták formájában definiált szabályok inkrementális kiértékelését:
  - Tetszőleges SIMULINK rendszermodell feldolgozható és importálható az EMF keretrendszerbe.
  - A kialakított SIMULINK metamodellek felett deklaratív kényszerek definiálása lehetséges, amelyek az EMF-INCQUERY segítségével hatékonyan kiértékelhetők.
  - Az eredményt visszajelezzük a SIMULINK felhasználói felületén.
- A szabályok alapján MATLAB kód generálását:
  - Az EMF-INCQUERY mintedefiníciókból a local search módszertanát használva lehetővé tettük MATLAB függvények generálását, melyek a vizsgálandó modellen elvégzik a validációt.
  - A hibásnak talált modellrészletekről a grafikus felhasználói felületen értesítjük a felhasználót.

A kialakított validációs módszereket megvalósítottam Eclipse komponensként az EMF és az EMF-INCQUERY keretrendszerekre építve, és az elkészült eszközt egy ipari partnertől kapott komplex repülőgépipari példákon értékeltem ki. Azonban a módszer skálázhatóságának részletesebb vizsgálata még jövőbeli feladat.

Továbbfejlesztési irányok A nagy modellekre történő skálázódás érdekében kulcskérdés az importálás minél hatékonyabban történő megvalósítása. Amennyiben meghatározható, hogy a megváltoztatott modell mely részét kell újra importálni, a modell többi része

változtatás nélkül megtartható és használható. A nagyméretű modelleket gyakran elosztva, több fájlban tárolják. Ezt kihasználva az importáláskor azt kell megvizsgálni, melyik fájlok változtak az előző importálás óta, és ekkor csak azokat kell újra feldolgozni.

A jelenlegi, kódgenerálásra használt megközelítés további módszerekkel bővíthető ki. Cél egy új, akár a modell felépítése szerinti adaptív algoritmus alkalmazása, mely nagyban javítja a futási teljesítményt.

Irodalomjegyzék

[1] Gábor Bergmann, Ákos Horváth, István Ráth, Dániel Varró, András Balogh, Zoltán Balogh, and András Ökrös. Incremental evaluation of model queries over EMF models. In *Model Driven Engineering Languages and Systems, 13th International Conference,MODELS'10*. Springer, 10/2010 2010.

[2] Budapest University of Technology and Economics, Fault Tolerant Systems Research Group. EMF-IncQuery. `http://viatra.inf.mit.bme.hu/incquery`.

[3] Frank Budinsky. The Eclipse Modeling Framework: Moving into model-driven development. `http://www.ddj.com/184406198?pgno=1`.

[4] Eclipse Foundation. Java development tool. `http://www.eclipse.org/jdt/`.

[5] Ed Merks Elena Litani and Dave Steinberg. Discover the Eclipse Modeling Framework (EMF) and Its Dynamic Capabilities. `http://www.devx.com/Java/Article/29093/0/page/1`.

[6] Péter Fehér, Pieter J. Mosterman, Tamás Mészáros, and László Lengyel. Processing simulink models with graph rewriting-based model transformation. In *Model Driven Engineering Languages and Systems, 15th International Conference,MODELS'12*, 2012.

[7] H. Giese, M. Meyer, and R. Wagner. A prototype for guideline checking and model transformation in matlab/simulink. In *Proc. of the 4th International Fujaba Days 2006, Bayreuth, Germany*, 2006.

[8] Ákos Horváth, Gábor Bergmann, István Ráth, and Varro Daniel. Experimental assessment of combining pattern matching strategies with viatra2. *International Journal on Software Tools for Technology Transfer*, 12:211–230, 2010.

[9] Dániel Varró István Ráth, Ábel Hegedüs. Derived Features for EMF by Integrating Advanced Model Queries. `http://www.inf.mit.bme.hu/research/publications/derived-features-emf-integrating-advanced-model-queries`.

[10] Joshua Kaplan. MatlabControl. `http://code.google.com/p/matlabcontrol/`.

[11] Elodie Legros, Wilhelm Schäfer, Andy Schürr, and Ingo Stürmer. Mate: a model analysis and transformation environment for matlab simulink. In *Proceedings of*

the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems, MBEERTS'07, pages 323–328, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] MathWorks. External Programming Language Interfaces. `http://www.mathworks.com/help/matlab/external-interfaces.html`.

[13] MathWorks. Simulink getting started guide. `http://www.mathworks.com/help/pdf_doc/simulink/sl_gs.pdf`.

[14] Daniel Merschen, Robert Gleis, Julian Pott, and Stefan Kowalewski. Analysis of simulink models using databases and model transformations. In *8th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES '12)*, 2012.

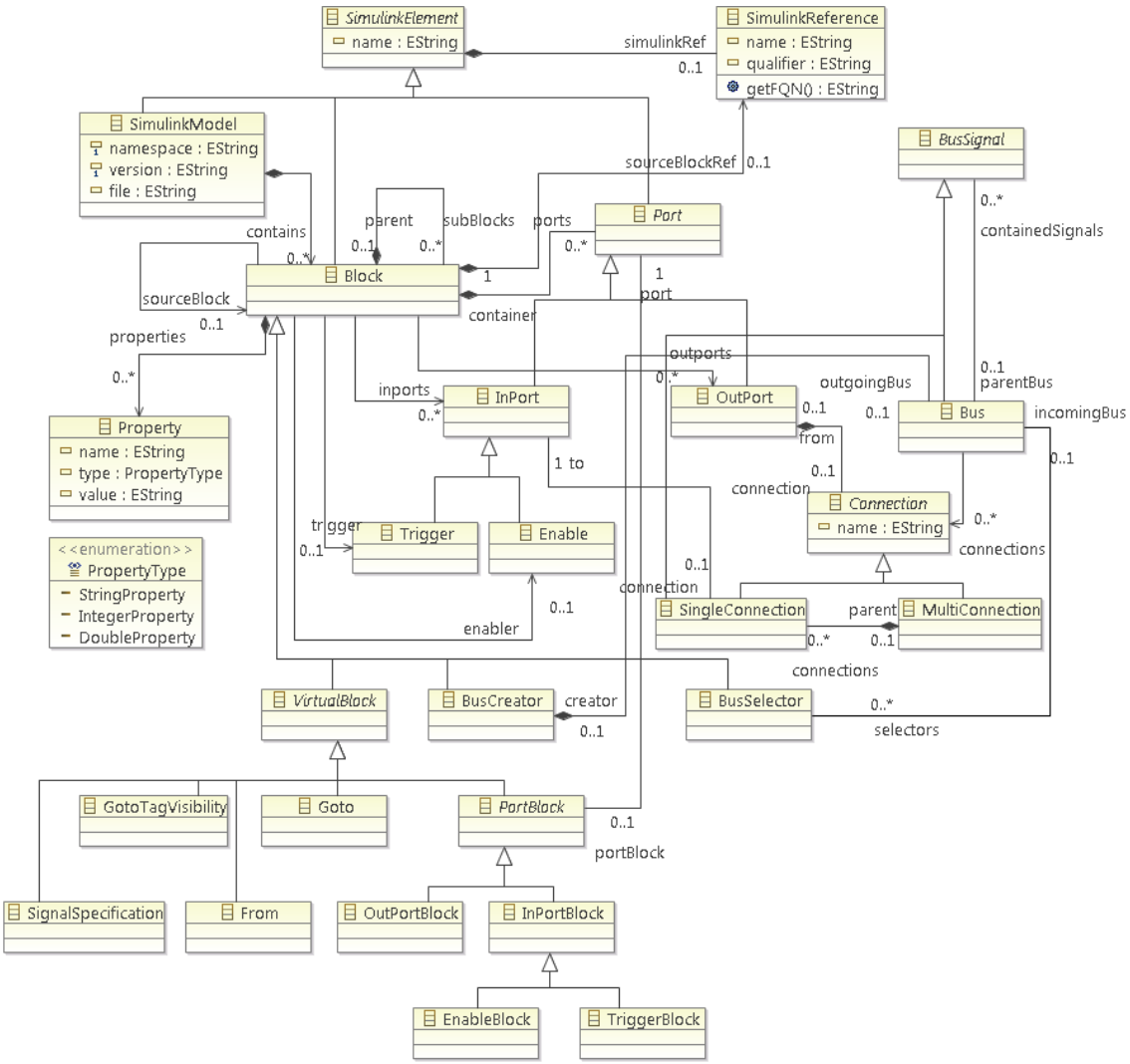
[15] Eclipse project. Ecore API. `http://download.eclipse.org/tools/emf/2.0.5/javadoc/org/eclipse/emf/ecore/package-summary.html`.

[16] RTCA - Radio Technical Commission for Aeronautic. Software Considerations in Airborne Systems and Equipment Certification (DO-178C), 2012.

[17] Sven Efftinge, Sebastian Zarnekow. Xtend. `http://www.eclipse.org/xtend/`.

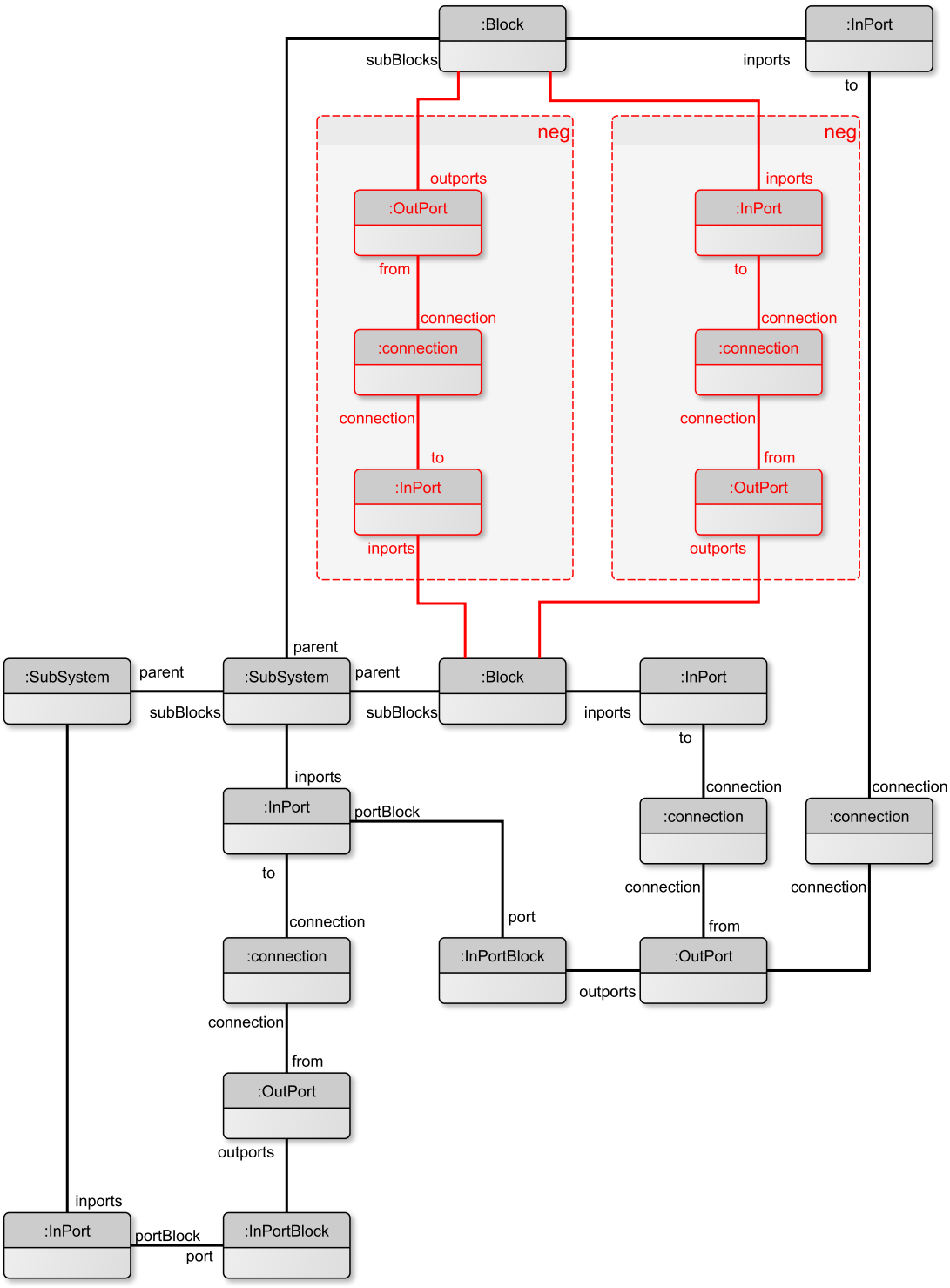
Függelék

F.1. A teljes Simulink metamodell



F.1.1. ábra. *A teljes* Simulink *EMF metamodell*

F.2. A mérés során használt szintetikus gráfminta



F.2.1. ábra. *A méréshez használt gráfminta grafikus reprezentációja*

```
pattern syntheticExamplePattern(Block) {
    SubSystem(Block);

    Block.inports(Block, InPort);
    InPort.portBlock(InPort, InPortBlock);
    Block.parent(SubBlock, Block);
    find directedLineExistsBetween(InPortBlock, SubBlock);

    Block.inports(SubBlock, SubInPort);
    InPort.portBlock(SubInPort, SubInPortBlock);
    Block.parent(SubSubBlock1, SubBlock);
    Block.parent(SubSubBlock2, SubBlock);
    SubSubBlock1 != SubSubBlock2;

    find directedLineExistsBetween(SubInPortBlock, SubSubBlock1);
    find directedLineExistsBetween(SubInPortBlock, SubSubBlock2);
    neg find directedLineExistsBetween(SubSubBlock1, SubSubBlock2);
    neg find directedLineExistsBetween(SubSubBlock2, SubSubBlock1);
}
```

F.2.2. ábra. *A méréshez definiált minta* EMF-IncQuery *definíciója*

F.3. A méréshez készített validációs mintához generált Matlab kód

Az F.2 függelékben található mintához az alábbi kódot generálja az eszköz.

```
function returnValue = syntheticExamplePattern( system )
%SYNTHETICEXAMPLEPATTERN the generated matcher code for the pattern named syntheticExamplePattern
% The code is genearted from the corresponding EMF-IncQuery query definition
	returnValue = 'no matches';
	Block = find_system_prepared(system,'BlockType','SubSystem');
	for j1 = 1:size(Block)
		Block_element = getListElement(Block,j1);
		InPort = find_system_prepared(Block_element,'SearchDepth','0','PortType','inport');
		for j2 = 1:size(InPort)
			InPort_element = getListElement(InPort,j2);
			InPortBlock = portToPortBlock(InPort_element);
			for j3 = 1:size(InPortBlock)
				InPortBlock_element = getListElement(InPortBlock,j3);
				SubBlock = find_system_prepared(Block_element,'SearchDepth','1','Type','Block');
				for j4 = 1:size(SubBlock)
					SubBlock_element = getListElement(SubBlock,j4);
					if(connecting(system,InPortBlock_element,SubBlock_element))
						SubInPort = find_system_prepared(SubBlock_element,'SearchDepth','0','PortType','inport');
						for j5 = 1:size(SubInPort)
							SubInPort_element = getListElement(SubInPort,j5);
							SubInPortBlock = portToPortBlock(SubInPort_element);
							for j6 = 1:size(SubInPortBlock)
								SubInPortBlock_element = getListElement(SubInPortBlock,j6);
								SubSubBlock1 = find_system_prepared(SubBlock_element,'SearchDepth','1','Type','Block');
								for j7 = 1:size(SubSubBlock1)
									SubSubBlock1_element = getListElement(SubSubBlock1,j7);
									SubSubBlock2 = find_system_prepared(SubBlock_element,'SearchDepth','1','Type','Block');
									for j8 = 1:size(SubSubBlock2)
										SubSubBlock2_element = getListElement(SubSubBlock2,j8);
										if(~eq(get_param(SubSubBlock1_element,'handle'),get_param(SubSubBlock2_element,'handle')))
											if(connecting(system,SubInPortBlock_element,SubSubBlock1_element))
												if(connecting(system,SubInPortBlock_element,SubSubBlock2_element))
													if(~connecting(system,SubSubBlock1_element,SubSubBlock2_element))
```

```
                            if(~connecting(system,SubSubBlock2_element, SubSubBlock1_element))
                              if(~exist('matches','var'))
                                matches = [{getfullname(Block_element)}];
                              else
                                matches = vertcat(matches ,{getfullname(Block_element)});
                              end
                            end
                          end
                        end
                      end
                    end
                  end
                end
              end
            end
          end
        end
      end
    end
  end
  if(exist('matches','var'))
    returnValue = matches;
  end
end
```