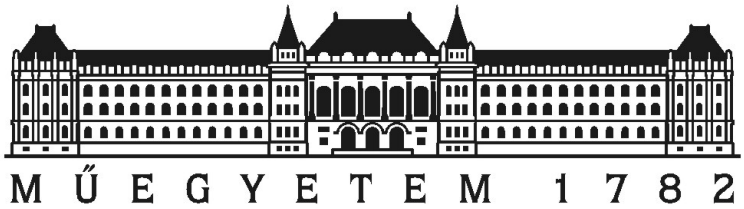


MATLAB-SIMULINK RENDSZEREK MODELL-ALAPÚ VALIDÁCIÓJA

Búr Márton
III. évf. Mérnök informatikus

Konzulensek:

Hegedüs Ábel
BME-MIT
tudományos segédmunkatárs

Horváth Ákos
BME-MIT
tudományos segédmunkatárs

Méréstechnika és Információs Rendszerek Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Budapest, 2012.

Búr Márton: MATLAB-SIMULINK rendszerek modell-alapú validációja

Összefoglalás

A beágyazott, biztonságkritikus rendszerek fejlesztése során kiemelt fontosságú a tervezett rendszer működésének tervezés idejű ellenőrzése. Ezek a rendszerek általában a környezetükkel interkacióba 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 TDK keretében egy olyan módszert valósítottunk meg, amelyben a SIMULINK modellekre vonatkozó rendszervalidációs kényszereket deklaratív módon definiálhatók és ezen kényszerek teljesülése tetszőleges SIMULINK modellen ellenőrizhető. A megvalósítás során 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 módszer használhatóságához szükséges, hogy a megszegett kényszerek esetén a kapcsolódó modell elemek azonosíthatóak legyenek a SIMULINK szekresztőfelületén is. Ennek érdekében a kialakított eszköz képes a modell lekérdező által szolgáltatott eredményeket automatikusan visszavetíteni a SIMULINK modellre, amely eredményeképp a kapcsolódó modellrészek egyértelműen azonosíthatók.

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 approach.

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 method 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. The approach can scale up to SIMULINK models with hundreds of thousands of elements thanks to the efficiency of the model representation and the incremental evaluation techniques of the query engine.

In order to ensure the usability of the approach, the corresponding model elements of violated constraints should be identifiable on the user interface of SIMULINK as well. Therefore, our technique can automatically annotate the results provided by the model query engine back to the SIMULINK models, thus the corresponding parts of the model can be properly identified.

Tartalomjegyzék

1. fejezet

Bevezetés

A beágyazott, biztonságkritikus rendszerek fejlesztése során kiemelt fontosságú a tervezett rendszer működésének tervezés idejű ellenőrzése. Ezek a rendszerek általában a környezetükkel interkacióba 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 tanusítványozási követelményeknek (pl. DO-178C [18]). 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 hatékony. Ezen túlmenően nehézségeket okoz az is, hogy a szakterület-specifikus követelmények a SIMULINK általános célú modellezési nyelvében csak bonyolult módon definiálhatóak.

Megközelítés. A TDK keretében egy olyan módszert valósítottam meg, amelyben a SIMULINK modellekre vonatkozó rendszervalidációs kényszereket deklaratív, szakterület-specifikus módon definiálhatók és ezen kényszerek teljesülése tetszőleges SIMULINK modellen hatékonyan ellenőrizhető.

A megvalósítás során a SIMULINK modelleket *átalakítom az Eclipse Modeling Framework által használt reprezentációra*, majd egy *automatikus leképezés* segítségével szakterület-specifikus modellek generálódnak. Ezen modellek már tömören, az adott szakterület számára fontos részleteket tartalmazzák csak ezáltal egyszerűsítve a deklaratív kényszerek definícióját.

A *validációs kényszerek* nagy hatékonyságú *kiértékelését* az EMF-INCQUERY modell lekér-

dező eszköz segítségével valósítottam 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* a szakterület-specifikus modellre való leképezés nyomonkövethetőségének köszönhetően.

Kapcsolódó munkák A Simulink rendszerek modell-alapú validációját más megoldások is részben támogatják, amelyek közül a MATE eszköz [8, 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 leírásásra. Végül a [7]-ban bemutatott megközelítés a VMTS transzformációs keretrendszerre épülve képes komplex modelltranszformációat és lekérdezéseket végrehajtani Simulink modellek felett.

Az említett megoldások mind modell-alapon működnek, azonban az általam megvalósított módszer két lényeges pontban különbözik: (i) támogatja az automatikus absztrakciót a szakterület-specifikus modellekre és (ii) inkrementális megközelítést használ a validációs szabályok hatékony kiértékelésére.

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 általam megvalósított modell-alapú validációs megközelítésről.
- A 4. fejezetben részletezem a megvalósítás legfőbb lépéseit: (i) a Simulink modellek alapú EMF alapú reprezentációját (4.1.3. fejezet), (ii) az automatikus leképezést szakterület-specifikus modellekre (4.2. fejezet), (iii) a validációs szabályok definiálásának módját (4.4. fejezet) és végül (iv) az eredmények visszajelzését az eredeti modellbe (4.5. fejezet).
- A megvalósított módszert értékelem a 5. fejezetben, míg a dolgozat zárásaként a 6. fejezetben összefoglalom a leírtakat.

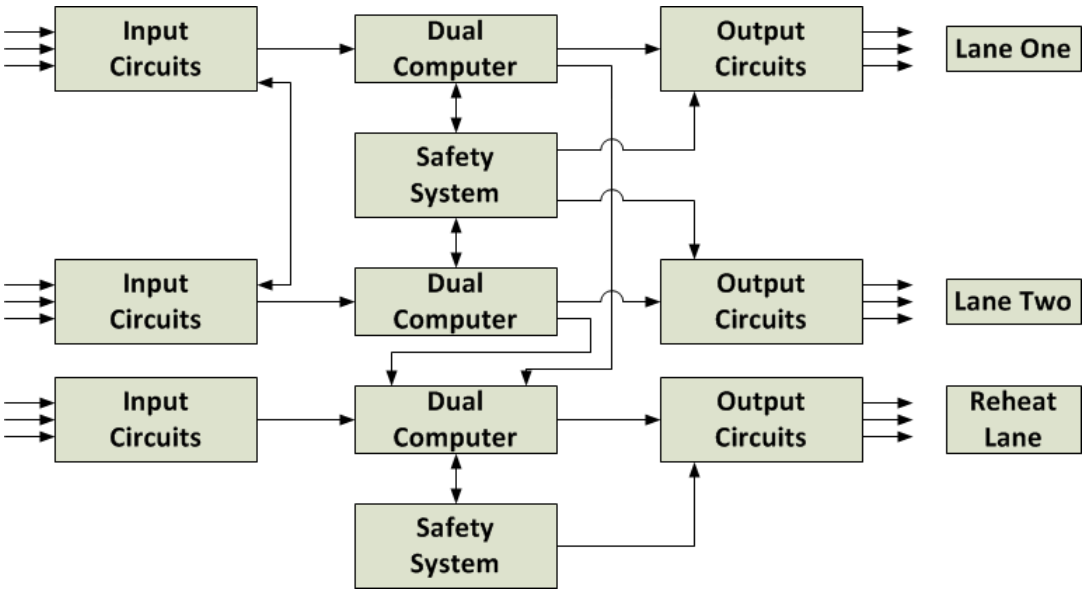
2. fejezet

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 példán keresztül bemutatom a MATLAB-SIMULINK modellézési megközelítést, továbbá az modell-vezérelt rendszerfejlesztésben széleskörben használt Eclipse Modeling Framework keretrendszert és az arra épülő EMF-INCQUERY inkrementális modell lekérdező eszközt.

2.1. Mintapélda: Sugárhajtómű motorvezérlő

A biztonságkritikus rendszerek alkalmazásának egyik fő példája a repülőgépek. Ezek nagyok sok komponensből álló bonyolult rendszerek, amik tervezésére szigorú szabályozások érvényesek. A Rolls-Royce cég gyártotta RB199-es sugárhajtómű motorvezérlő blokkvázlata [15] a 2.1. ábrán látható. Ez a típusú vezérlés a Tornado típusú vadászgépekben az RB199 ma is megtalálható [17].



2.1. ábra. Az RB199-es sugárhajtómű motorvezérlő blokkvázlata.

A fenti blokkvázlaton láthatók a vezérlést megvalósító komponensek és azok csatlakozásai. Az érzékelőktől és a repülőgép egyéb kapcsolódó egységetiől beérkező jeleket jelkondícionáló áram-

körök erősítik (**Input Circuits**), majd továbbítják a számítógépek (**Dual Computer**) felé feldolgozásra. Egyes bemeneti áramkörök egymással is összeköttetésben állnak, egymásnak ellenőrző, úgynevezett *heartbeat jeleket* küldve, ezzel is javítva a hibadetektálást és a biztonságosságot.

A számítógépek a beérkező jeleket feldolgozzák, majd a kimeneti áramköröket (**Output Circuits**) irányítják, illetve jeleket küldhetnek egy másik számítógépnek, mely a működésüket, továbbá saját eredményeit ellenőrzi. Minden számítógép csatlakozik egy biztonsági rendszerhez (**Safety System**), mely szintén a rendszer hibatűrését növeli. Ezek ellenőrzik a kimeneti áramkörök működését. A kimeneti áramkörök jelei határozzák meg a redundáns vezérlőcsatornákba (az ábrán **Lane One** és **Lane Two**), illetve az utánégető csatornába (**Reheat Lane**) juttatott üzemanyag mennyiségét.

A rendszerek esetén bizonyos strukturális kényszerek betartása kívánatos a modell helyességének érdekében. Jelen esetben is kimondhatók szabályok, például minden kimenetre jelet adó számítógép csatlakozzon egy biztonsági rendszerhez, minden kimenet egy biztonsági rendszerhez. Erre amiatt van szükség, mert a rendszer kimenetén megjelenő vezérlőjeleknek a lehető legnagyobb biztonsággal kell helyesen megjelennie.

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

2.2. MATLAB-SIMULINK

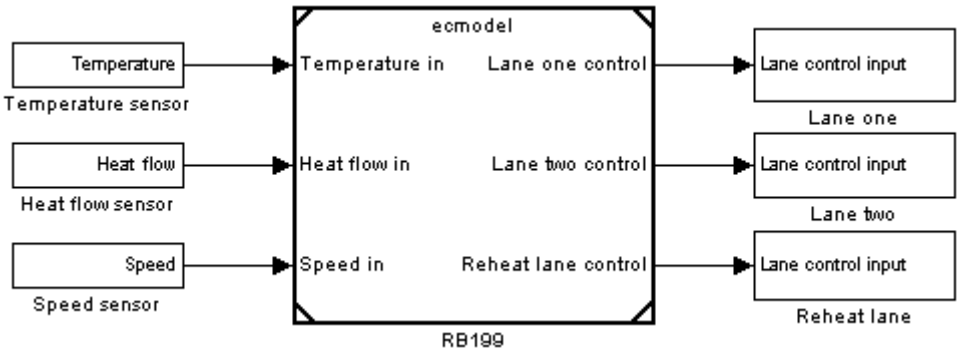
A SIMULINK hivatalosan 1990 óta [13] a MATLAB részét képezi. A világon azóta sokhelyütt elterjedt, számtalan területen alkalmazzák, így nagyon sok mérnök is. Szokták mondani, hogy a világon bizonyos szinten az angol hivatalos nyelvnek tekinthető napjainkban. Ehhez hasonlóan, ha két fél valamilyen rendszer 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 modellek 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 hozhatók létre **rendszerek (system)**, másnéven **modellek (model)**.

Példa: A motorvezérlés szimulációjára alkalmas SIMULINK blokkdiagramon (2.2. ábra) középen szerepel az irányítást végző RB199 modul és a hozzá kapcsolódó elemek. A vezérlés az érzékelőktől beérkező jelek alapján történik, és jelen esetben a motorban elhelyezkedő három csatorna működését befolyásolja. Az érzékelők ezen csatornák jellemzőit, illetve bizonyos környezeti hatásokat mérnek, ezek függvényében változik a vezérlő kimenete.



2.2. ábra. A vezérlő és környezete

Minden, a környezet állapotáról információt szolgáltató jel a *Temperature in*, *Heat flow in* és a *Speed in* port blokkok valamelyikéből érkezik a vezérlőbe, majd a jelerősítő áramköri funkciót ellátó blokkokba futnak be. A blokkok közötti nyilak az összeköttetések, melyek portokból indulnak és portokba mutatnak, ezzel egyben a jelterjedés irányát is meghatározzák. Egy jel több helyre is rákapcsolható. Egy port jelen modell esetén csak egyirányú kommunikáció lebonyolítását támogatja.

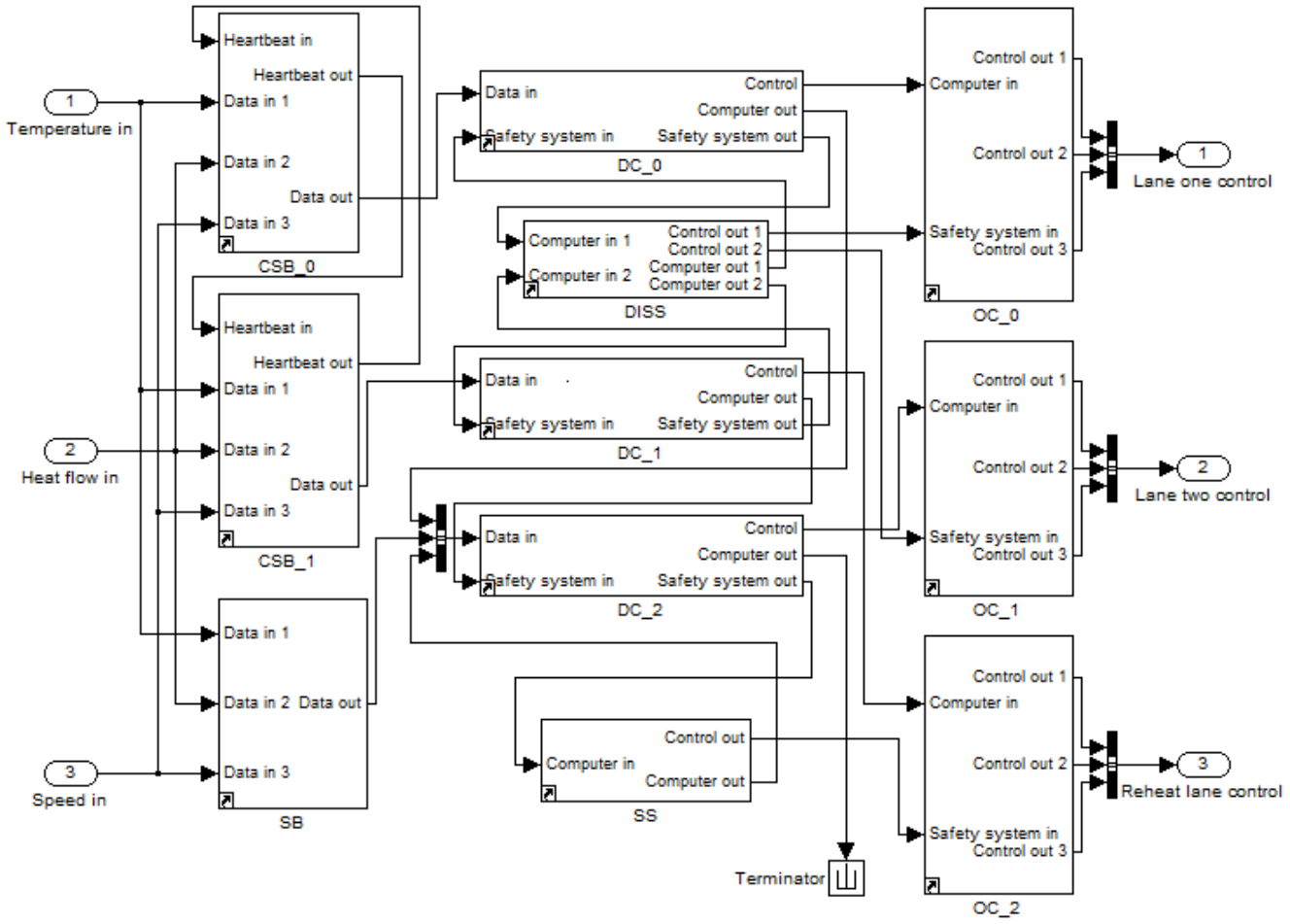
A blokkokat reprezentáló téglalapok belsejében van feltüntetve az egyes portok neve, illetve a blokk alatt szerepel a diagramonként egyedi blokknév.

2.2.1. 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 blokkokat létrehozni. Egy blokk lehet ún. **atomi (atomic)**, vagy **alrendszer (subsystem)**. A 2.3. ábrán például atomi blokk a *Terminator*, és alrendszer a *DC_ 0*. Egy blokk akkor alrendszer, ha a belső felépítésében szerepel blokk, és akkor atomi ha nem alrendszer. 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, ún. **port blokkok (port block)** reprezentálják a belsejében. A SIMULINK mudellt megjelenítő ábrán ilyen port blokk például a *Temperature in* nevű blokk.

Az 2.2. ábrán szereplő RB199 elnevezésű blokk belső felépítése látható a 2.3. ábrán. A 2.1. ábrán szereplő blokkvázlatnak megfelelően minden elemet SIMULINK blokkok reprezentálnak.

A *DC_ 2* elnevezésű blokk bemeneténél található *Bus creator* blokk a jelek megfelelő kezelését szolgálja: összefogja a számítógép *Data in* portjába irányuló kapcsolatokat. Szintén a *DC_ 2* blokkhoz kapcsolódik a *Terminator* nevű blokk, az ábra alján található modellelem. Az ilyen blokkba vezetett jelek nem kerülnek további feldolgozásra. A *CSB_ 0*, *CSB_ 1* és *SB* blokkok jelentik a bemeneti jelerősítő áramköröket. A *DC_ 0*, *DC_ 1* és *DC_ 2* blokkok számítógépek, a



2.3. ábra. Az RB199 motorvezérlő SIMULINK modellje

hozzájuk csatkozó *SS* és *DISS* blokkok pedig biztonsági rendszerek. A kimeneti áramköröket az *OC_0*, *OC_1* és *OC_2* blokkok jelentik

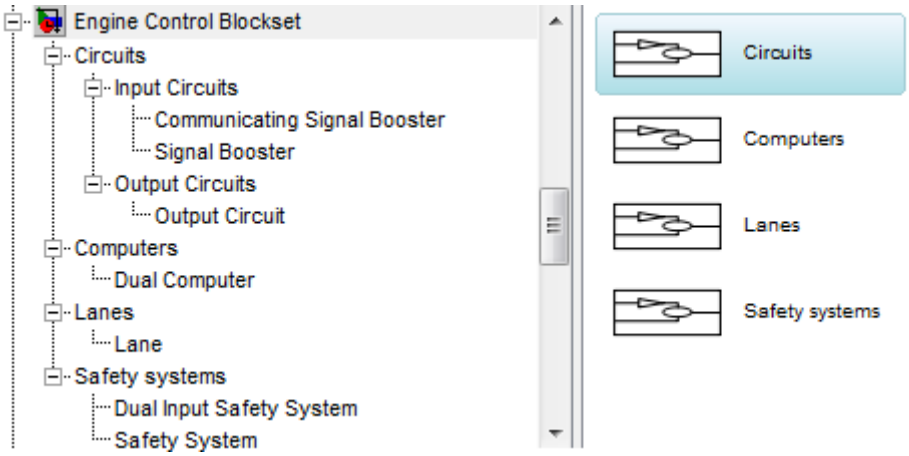
2.2.2. Könyvtár (Library)

Egy könyvtár felépítése hierarchikus, fa struktúrában tárolja az elemeket. Blokkokat tartalmaz, melyek között lehetnek speciális, **alkönyvtárként (sublibrary)** szolgáló blokkok, amik további blokkokat tartalmaznak. Ez a könyvtárstruktúra a rendelkezésre álló, felhasználható blokkok és áttekinthetőségét segíti. Lehetőség van könyvtárak definiálására is, melyhez a rendszerbe beépített, eleve meglévő blokkokat lehet felhasználni. A könyvtárak a SIMULINK-be beépített **Library Browserben** is megjeleníthetők (a 2.4. ábrán is ez látható).Fontos megjegyezni továbbá, hogy a könyvtárak tekinthetők speciális modelleknek is, mivel felépítésükben megegyeznek.

A modell blokkjai alapbeállítás szerint kapcsolódnak ahhoz a blokkhoz, akinek a másolataként létrejöttek. Ezt egy **linknek (link)** nevezett kapcsolat reprezentálja. A linkek segítségével elérhető, hogy egy adott könyvtári elemben végzett módosítás érvényes legyen az összes, belőle másolással létrehozott elemre is, illetve elérhető, hogy egy lokálisan módosított blokk változásait könyvtár szintre emeljük, így minden vele azonos blokkra kiterjeszthetjük. A linkek ugyanakkor lehetővé teszik a modellben szereplő blokk megváltoztatásának lokális tiltását vagy engedélyezését. Adott esetben a linkek tilthatók (tehát figyelmen kívül hagyhatók) és meg is szüntethetők. A link állapotát és jelenlétét egy blokk esetén az egyes blokkokat reprezentáló téglalapok bal alsó

sarkaiban látható nyilak jelzik a 2.3. ábrán.

Példa: A motorvezérlő elemeket tartalmazó SIMULINK könyvtár az 2.4. ábrán látható.



2.4. ábra. A motorvezérlő blokkokat tartalmazó könyvtár a Library Browserben. A fa struktúra leveleiben láthatók az egyes elemeket reprezentáló blokkok

A könyvtáron belüli kategorizálás alkönyvtárak formájában történik a blokkok funkciói alapján: *Circuits (Áramkörök)*, ezen belül találhatók továbbá az *Output Circuits (Kimeneti Áramkörök)* és az *Input Circuits (Bemeneti Áramkörök)* alkönyvtárak; *Computers (Számítógépek)*, *Lanes (Csatornák)* és *Safety Systems(Biztonsági Rendszerek)*. A könyvtár megjelenítése emellett lehetséges a library browser keretein kívül is, ekkor mint egy modell szerkeszthető.

2.2.3. Modellek validációja SIMULINK-ben

Tegyük fel, hogy a modell helyességének egyik feltétele, hogy minden kimenet csatlakoztatva van egy biztonsági rendszerhez, illetve egy számítógéphez, és a csatlakozó számítógép és biztonsági rendszer is összeköttetésben áll. A SIMULINK eszközei között nem található olyan, mely az elkészült modellek esetén ilyen, és ehhez hasonló strukturális kényszerek betartását tudná automatikusan ellenőrizni, így az ilyen típusú feladatok korábban igen nagy emberi erőfeszítéstbe is kerülhettek.

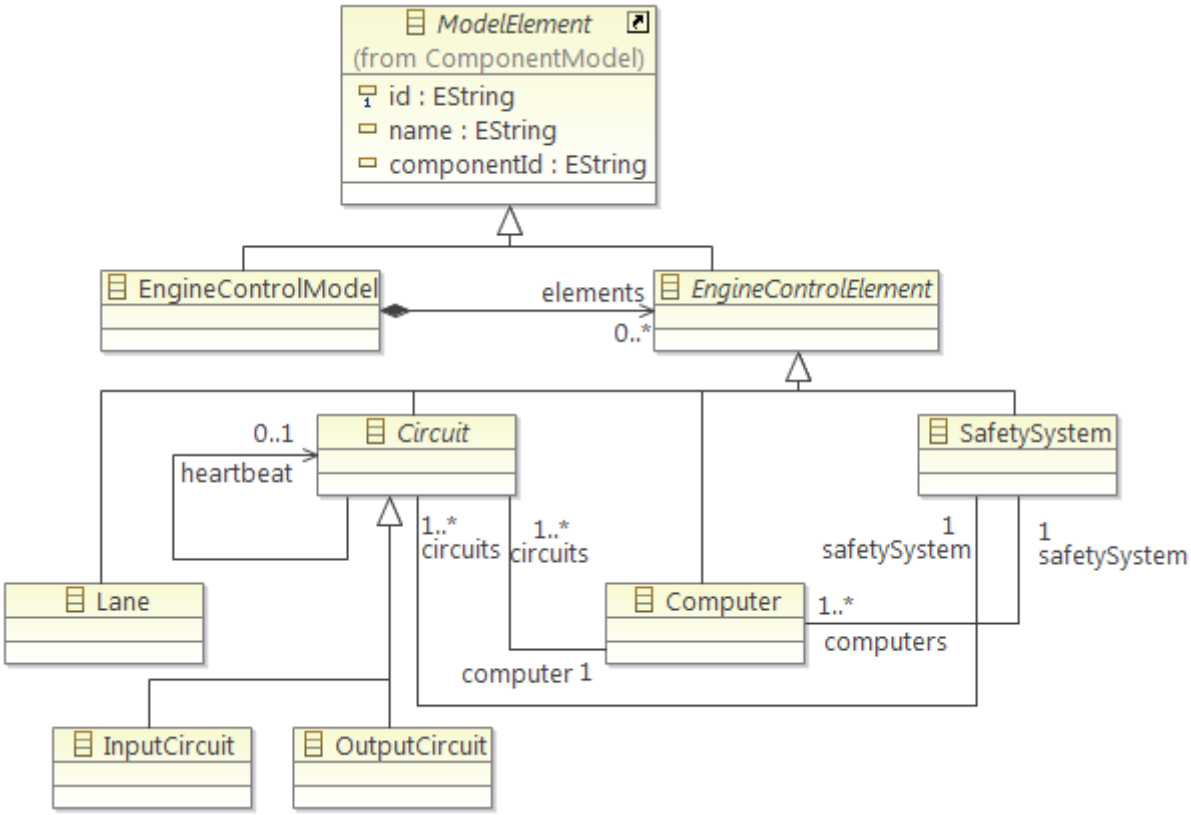
2.3. Eclipse Modeling Framework

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 a 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: A mintapéldához (2.1. ábra) megfeleltetett Ecore modellt a 2.5. ábra mutatja. A felvett modellben megtalálhatók a SIMULINK *Engine Control Blockset* könyvtárban szereplő alkönyvtárak. Az Ecore bemutatása során ez egyes elemek jelentésére a magyarázat ezen példához kötődik.



2.5. ábra. engineControl.ecore modell az RB199 motorvezérlő metamodellje az EMF diagramszerkesztőjében

Az **Ecore**, ami 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 diagramon EClass-ként szerepelnek a SIMULINK könyvtár alkönyvtárai.
- **EAttribute** modellezi az attribútumokat, az objektum adatainak komponenseit. A nevük és típusuk azonosítja őket. A példában ezek a *ModelElement* EClass esetén: *id*, *name* és *componentId*
- **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 *ModelElement* EClass *id* 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 alkamazható, mint például egy *Safety System* esetén a *computers* és a *circuits* kapcsolatok.

Példa: Jelen esetben a referenciák jelentése a modellben az, hogy egy `ModelElement` objektum melyik másik `ModelElement` objektumhoz kapcsolódik, tehát a két motorvezérlő elem milyen viszonyban áll egymással, **milyen fizikai összeköttetésben állnak**. Ennek a jelentősége az, hogy már metamodell szinten meg lehet kötni, hogy egy adott kapcsolat mentén milyen típusú elemek csatlakozhatnak. Mivel az EReference relációk minden esetben irányítottak, ezért a kapcsolat irányát a referenciák irányításával lehet megadni. Ezzel szemben a SIMULINK-ben csak összeköttetések vannak, amik portokhoz csatlakoznak és ezeken keresztül kötnek össze blokkokat. Hátránya, hogy nem korlátozható, hogy mikor köthető össze két blokk egymással, illetve melyik portjukon keresztül kell csatlakozniuk.

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** (@generated), 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 modellvezértel szoftverfejlesztéssel, amíg a Java-s megközelítés azoknak előnyös, akik tisztán Java környezetben fejlesztenek, például Java Development Tool (JDT) [5] 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 [16] 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) [4], 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** [6]. A keretrendszer dinamikájának és a reflektív API-nak a kombinálásával lehetőség nyílik a a metamodell futásidejű megváltoztatására.

2.4. EMF-IncQuery

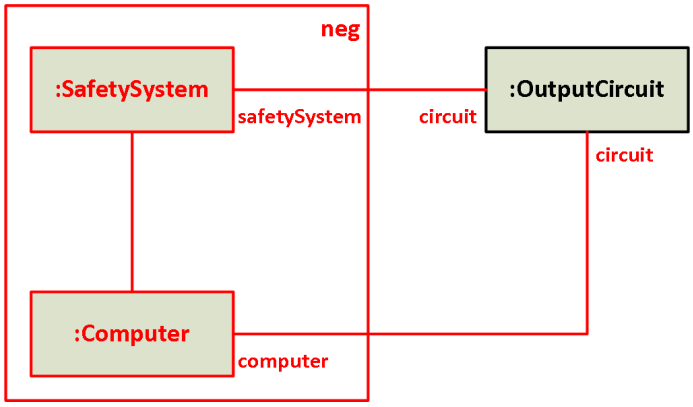
Az EMF-IncQuery [3] egy keretrendszer, mely lehetővé teszi deklaratív lekérdezések (query-k) definiálását EMF modellek esetén. Ezen lekérdezések aztán hatékonyan végrehajthatóak egyéb kézzel írt kód nélkül is.

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 modellel szemben az, hogy minden kimeneti áramkör csatlakozzon egy számítógéphez és egy biztonsági rendszerhez, ahol a számítógép és a biztonsági rendszer egymáshoz is csatlakozzon. Ha van olyan kimeneti áramkör, amely nem teljesíti a feltételt, a modell hibás. Ennek megfogalmazása szerepel minta formájában a 2.6. ábrán.

A feltüntetett kapcsolatok az egyes metamodellbeli EReference-k 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 a konstellációban szerepelnek, a minta illeszkedik és a keretrendszer ez egy hibajelzést generál a felhasználó számára.

A keretrendszer által támogatott deklaratívan megfogalmazott gráfminták definiálásával a modell bejárásának implementálása nem feladatunk, automatikusan megtörténik. Továbbá ilyen módon kényelmesen fogalmazhatóak meg EMF objektumok közötti bonyolult relációk is, melynek számos előnye van:



2.6. ábra. Strukturális kényszer grafikus megfogalmazása az ellenőrzött kimenetre

- 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

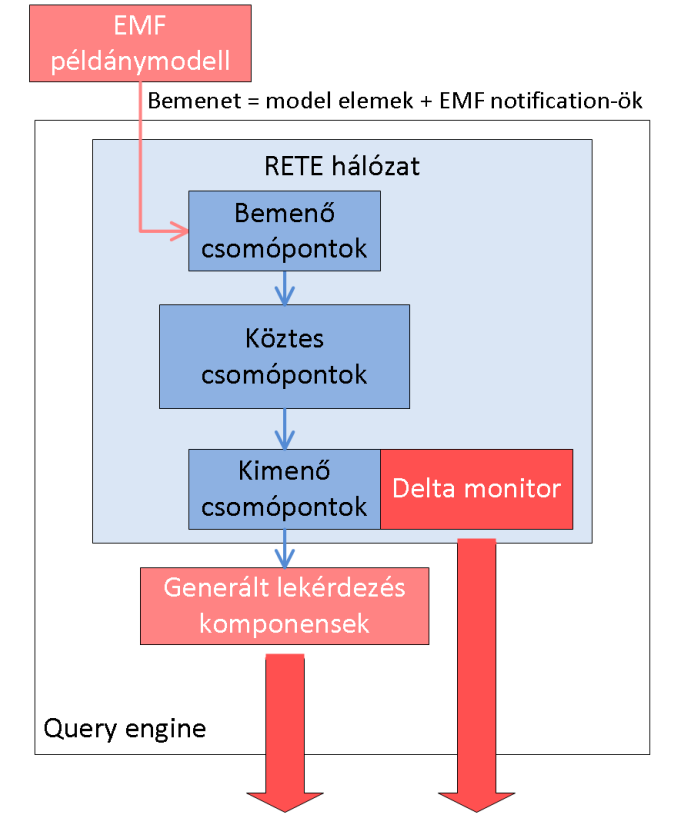
Az inkrementális gráfminta illesztés folyamat bemenete az EMF példánymodell, amire a mintát kell illeszteni és a hozzátartozó notification API. A notification API lehetővé teszi callback függvények regisztrálását a példánymodell elemeihez, amik jelzés objektumokkal (angol elnevezés a notification object, 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 2.3. 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 nagy méretű modelleken is. Ez a speciális teljesítmény karakterisztika, amíg elegendő a rendelkezésre álló memória, jól skálázhatóságot eredményez. Ez az architektúra a 2.7 ábrán látható.

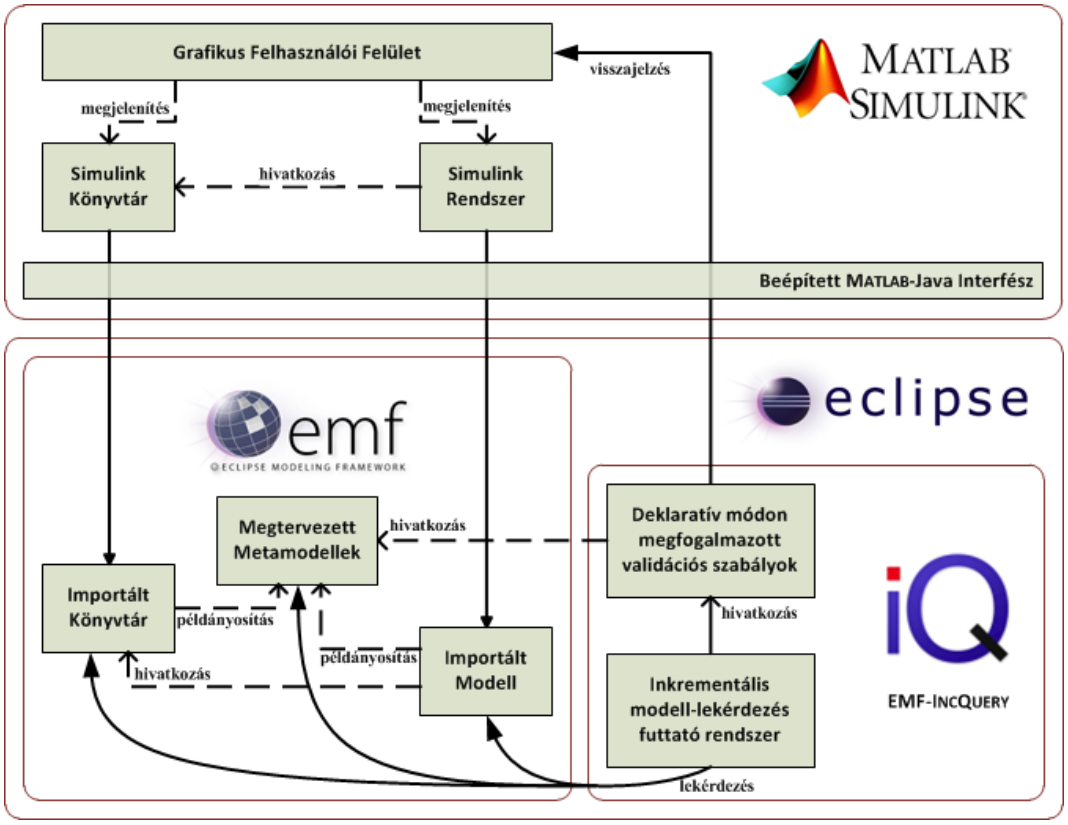


2.7. ábra. Az EMF-IncQuery architektúrája [9]

3. fejezet

MATLAB-SIMULINK modell-alapú validációjának áttekintése

A MATLAB-SIMULINK modellek validációját megvalósító módszer áttekintése a a 3.1. ábrán látható. Az általam megvalósított módszer négy jól elkülöníthető lépésre bontható. A SIMULINK modellek kezelésének előfeltételei, ezen modellek transzformációja szakterület-specifikus modellekké, az előállított modelleken elvégzett validáció és az eredmények visszajelzése a felhasználó felé.



3.1. ábra. A megoldás vázlata

Előfeltételek Annak érdekében, hogy SIMULINK modellek fölött deklaratívan tudjunk validálni, 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 modell ellenőrzés eredményéről visszajelzést kell küldeni a felhasználnak (lásd 4.1.2. fejezet).

Az integráció megvalósításával és metamodell elkszítésével lehetővé válik modellek importálása Simulink-ből EMF alá (lásd 4.1.3. fejezet).

Transzformáció Mivel a Simulink egy általános célú modellezési nyelv, ezért az abban megvalósított rendszerek modelljei a struktúrájukat tekintve megegyeznek. Éppen ezért semmilyen, szakterület specifikus egyszerűsítéssel vagy információval nem rendelkeznek. Ezért, annak érdekében, hogy a validációs szabályokat a szakterületen járatos tervezőmérnökök definiálhassák, a Simulink modelleket leképezése szükséges, hogy előállítsunk egy olyan modellt, aminek az elemei már a szakterületre jellemzők.

Az így létrehozott modellek az **előre megtervezett metamodellek példányai**. A megoldás egyik fontos jellemzője, hogy a Simulink-beli rendszerek és könyvtárak egymás közötti lényeges hivatkozásai továbbra is megmaradnak, továbbá a leképezés nyomonkövethető, így lehetővé téve az eredmények későbbi Matlab-ba való visszavetíthetőségét (lásd 4.2. fejezet).

Validáció Annak érdekében, hogy a validációs szabályokat közvetlenül a Simulink modellek felett tudjuk ellenőrizni, az EMF-IncQuery keretrendszer használtuk. Ez az eszköz képes több, mint 100 000 elemet tartalmazó modellek esetén is az ellenőrzést hatékonyan elvégezni **inkrementális** algoritmusok használatával (lásd 4.4. fejezet).

A modellekre alkalmazandó szabályok **deklaratív módon leírt gráfminták** segítségével definiálhatóak. Ezeket a rendszer értelmezi, majd a modelleket automatikusan ellenőrzi, és megtalálja az esetleges hibákat.

Visszajelzés 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 4.5. fejezet).

4. fejezet

MATLAB-SIMULINK rendszerek modell-alapú validációja

Ebben a fejezetben részletesen bemutatom a kialakított modell-alapú validációs módszer lépéseit. Először a SIMULINK rendszerek EMF alapú modell reprezentációját és ezen modellek MATLAB-ból való importálási módját ismertetem (4.1. fejezet). Ezután a SIMULINK könyvtárak és rendszermodellek automatikus leképezését részletezem szakterület-specifikus metamodellekre és példány modellekre (4.2. fejezet). Végül bemutatom a validációs szabályok deklaratív definícióját ezen előállított modellek felett (4.4. fejezet) és a validáció eredményeinek visszajelzését az eredeti modellen (4.5. fejezet).

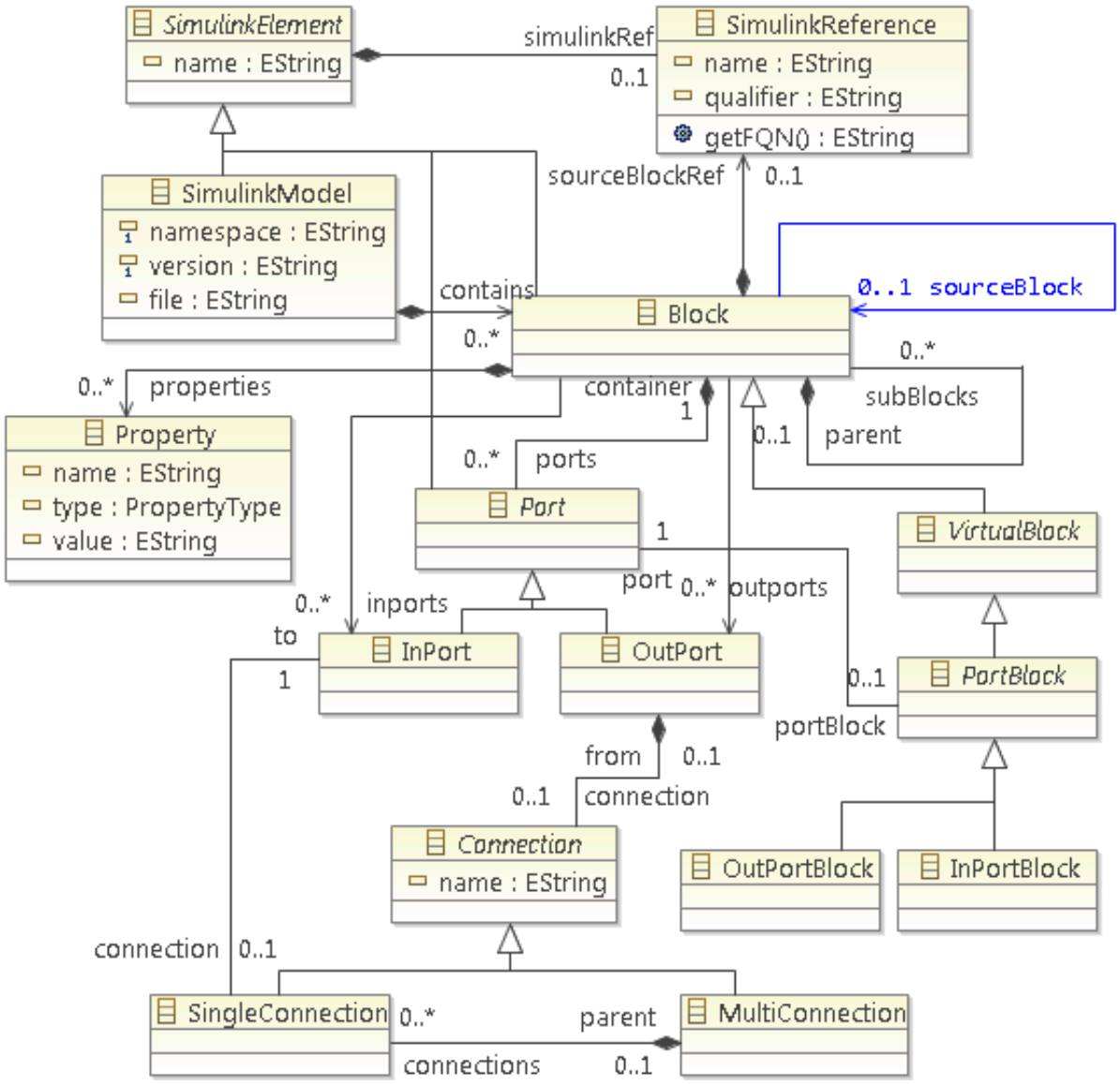
4.1. SIMULINK rendszerek modellezése

Minden SIMULINK modell a struktúráját tekintve azonos: blokkokból épül fel, a blokkok portokkal rendelkezhetnek, ezeken keresztül kapcsolódhatnak egymáshoz a blokkok. Emellett a blokkok további blokkokat tartalmazhatnak, ezáltal egy hierarchikus tartalmazási struktúrát létrehozva. A modell elemek típusai tehát semmilyen információt nem hordoznak azzal kapcsolatban, hogy pontosan mit reprezentálnak. Erről mindössze az adott elem neve és a könyvári blokkhoz kapcsolódó link tájékoztat.

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 4.1. ábrán látható, amey tartalmazza a leglényegesebb, és a példa megértéséhez szükséges elemeket. A pontosabb EMF leképezést is lehetővé tevő metamodellről megtalálható a modeldiagram az A függelékben.

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, rövidítve FQN).



4.1. ábra. A SIMULINK metamodell részlete

A SIMULINK-beli a 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 al-blokkokat, ezért ennek kifejezésére a `Block` objektumok között lévő gyermek-szülő viszonyt jelentő `parent` és `subBlock` referenciák állíthatók be.

A `Block` referenciát tartalmaz egy olyan `SimulinkReference`-re is, ami annak a blokknak a teljes nevét tárolja, amelyik blokkal a könyvtári link összeköti. Ennek alapján egy EMF-INCQUERY-beli, **lekérdezés alapú számított kapcsolatnak (query based feature)** [9] nevezett mechanizmus automatikusan és azonnal kiszámolja a `sourceBlock` referencia értékét minden `Block` példány esetén. Ilyenkor a `sourceBlock` EReference egy úgynevezett **számított érték (derived feature)**, ennélfogva a 4.1 ábrán megkülönböztetésül eltérő betütípussal szerepel a neve.

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 al-blokk. 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 MATLAB-bal

A MATLAB parancssori felületén keresztül minden olyan funkció is elérhető, melyhez a program grafikus felületet biztosít. Ezalól a SIMULINK-ben elérhető funkciók sem kivételek, így modelleket akár tisztán **parancssori utasításokkal (command)** létre lehet hozni, amik aztán 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 alkalamas 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 egy 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 a **teljes nevével (fully qualified name, 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 al-blokk nevének lekérése
   IV. a fenti lépést rekurzív ismételése az al-blokkokra, amíg az éppen vizsgált blokk nem tartalmaz további al-blokkot
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 azonosíása.
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

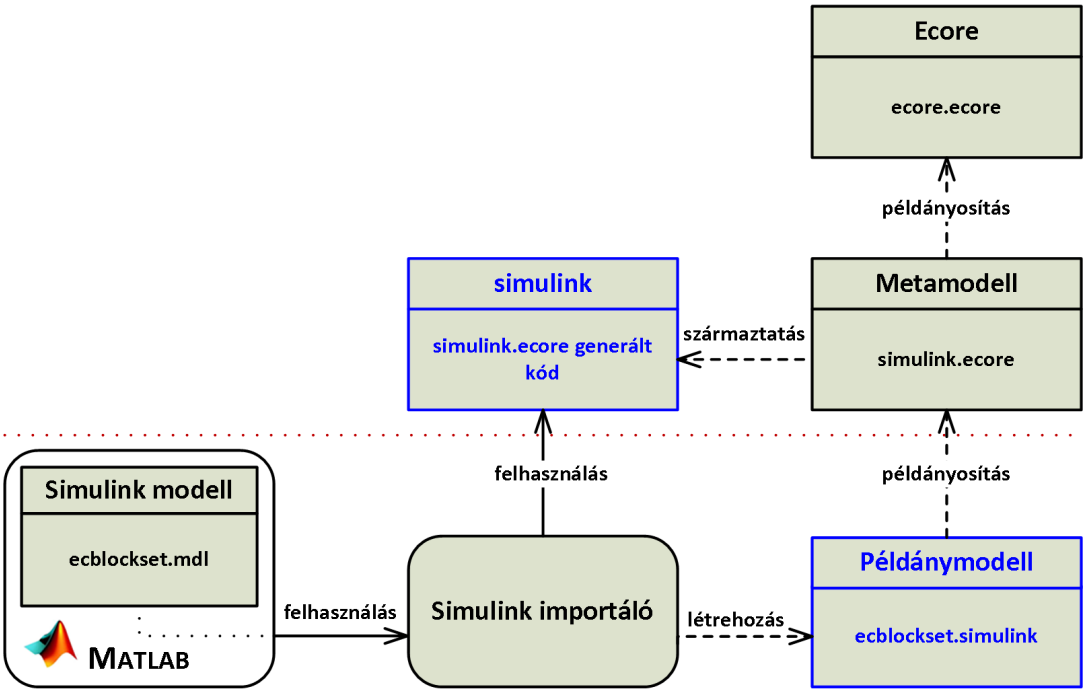
Az EMF reprezentáció létrehozásához szükséges elemek közé tartozik a Simulink metamodellből generált kód és egy bejárást végző modul. A modellhez tartozó kód automatikusan generálható az EMF beépített eszközeinek segítségével. A bejeáró implementálásakor a 4.1.2. fejezetben felsorolt lépések megvalósítása szükséges, melyek ezesetben magukba foglalják a generált kód hasznosítását. A bejárás eredményeképp egy Simulink példánymodell jön létre.

Megjegyzés: Ahogy az már a 2.2.2. fejezetben megjegyzésként szereplet, a könyvtárak tekinthetők speciális Simulink példánymodelleknek, így az importálás folyamata könyvtárak esetén is ezzel azonos.

Példa: A 2.1. fejezet mitapéldájához tartozó ecblockset könyvtár importálásához felrajzolt blokkvázlat a 4.2. 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 **ecblockset.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 mteaszinteknek megfelelő sorrendben helyezkednek el.

4.2. Transzformáció

Az előző fejezetben bemutatott EMF reprezentáció segítségével a SIMULINK modellek leírhatóak. Azonban annak érdekében, hogy a validációs szabályokat szakterület-specifikus módon



4.2. ábra. SIMULINK-beli modell EMF-be történő importálásának vázlata

definiálhassuk, szükség van egy leképezési módszerre, amely a SIMULINK könyvtárakból előállítja a szakterület-specifikus metamodelleket és példánymodelleket.

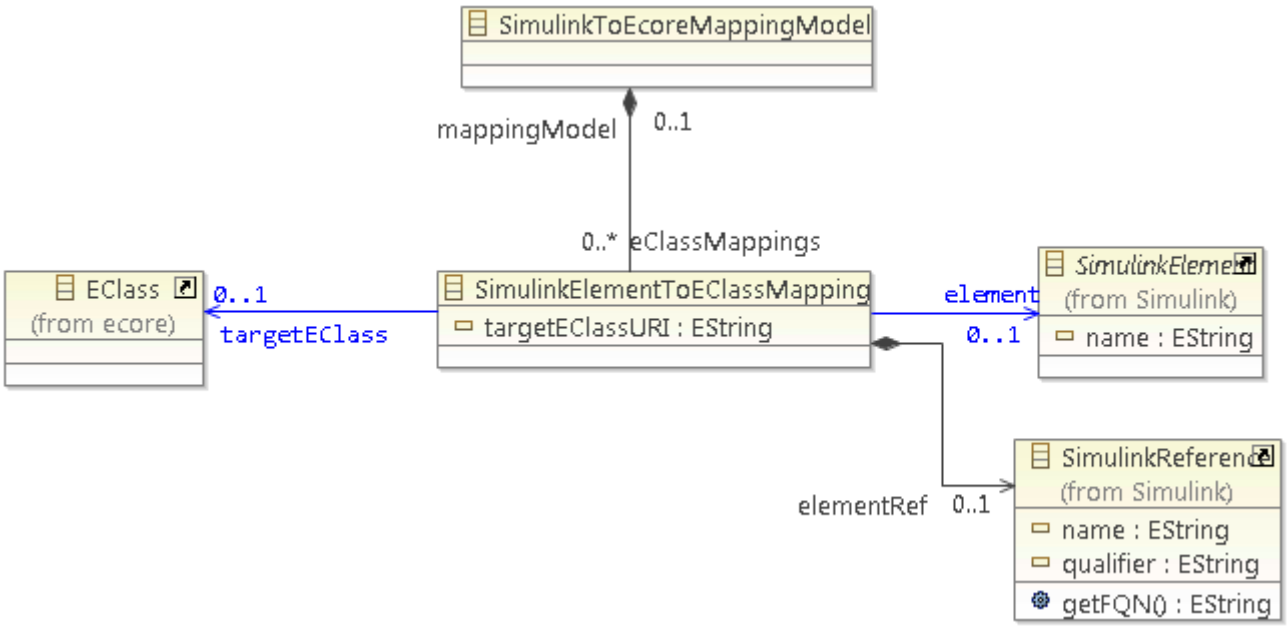
4.2.1. Szakterület-specifikus nyelvek

A szakterület-specifikus nyelvek arra használhatók, hogy egy bizonyos szakterületen felmerülő problémák megfogalmazását egyszerűbbé, hatékonyabbá tegyék. Az angol elnevezésük Domain Specific Languages, az ebből eredő elterjedt rövidítés a DSL.

A SIMULINK modellező nyelvének elemei nem reprezentálnak egyetlen konkrét szakterületet sem, minden szakma specifikus jellemző valamilyen változóban tárolódik. Ennél fogva az előzőekben (4.1.1.) bemutatott, a SIMULINK modellek reprezentációjára alkalmas metamodell megalkotásakor sem volt szempont, hogy szakterület-specifikus elemeket tartalmazzon a metamodell, ellenben minél pontosabban leírható legyen egy tetszőleges SIMULINK modell EMF-ben.

Cél az importált Simulink EMF modellt leképezni (az ehhez tartozó metamodell a 4.1. fejezetben látható) a modell által leírt szakterület specifikus modellé. Erre a könnyebb kezelhetőség és átláthatóság, valamint hatékonysági szempontok miatt van szükség. Ennek módja, hogy egy előre elkészített **megfeleltetés (mapping)** alapján bizonyos feltételeknek eleget tevő, illetve meghatározott jellemzőkkel rendelkező Simulink EMF objektumok leképződnek egy adott DSL példányára.

A megfeleltetés modell a `SimulinkElement`-ekhez egy-egy EClass-t rendel az elem FQN-je, és az EClass beli `EClassURI` attribútum alapján. A célpont `SimulinkElement` teljes nevét a mapping objektumban tárolt `SimulinkReference` példányból nyeri, míg az `EClassURI` értékét a mapping objektum a `targetEClassURI` attribútumában tárolja. A mapping metamodell részlete a 4.3. ábrán látható. A `SimulinkElementToEClassMapping` objektumainak `targetEClass` és `element` referenceiái korábban említett számított értékek (4.1.1. fejezet), amiket az EMF-INCQUERY számol ki a megfelelő attribútumértékek alapján.



4.3. ábra. A simulink-Ecore EMF modellek közötti megfeleltetési metamodell részlete

Annak érdekében, hogy a transzformációt megvalósíthassuk, előre meg kell határozni a megfeleltetési modellt. Ennek a feladata lényegében kettős:

- meghatározza az egyes objektumok **leképezésének szabályát**
- egy jól megválasztott összerendelési modell a **nyomonkövethetőséget** is lehetővé teszi

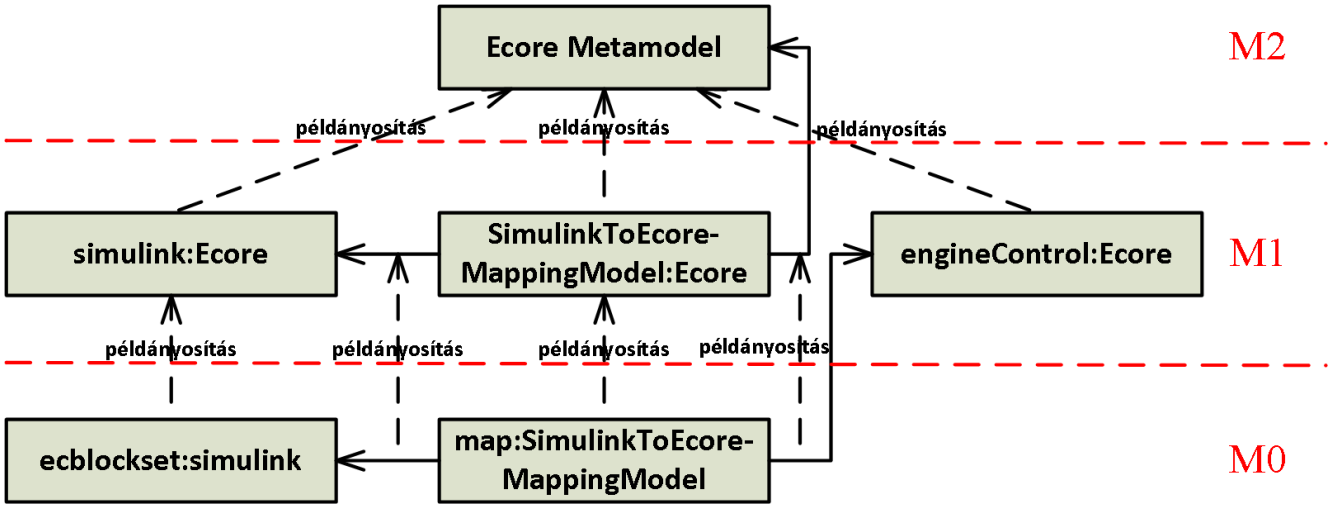
A nyomonkövethetőségnek itt a validáció miatt kiemelt szerepe van, mivel meg kell tudni határozni a transzformáció utáni elemekhez tartozó Simulink elemeket az importált modellen, hogy a problémát az eredeti modellen is jelezni lehessen.

Példa: A korábbi elnevezéseket használva az engineControl.ecore (2.5. ábra) szakterület-specifikus Ecore metamodell és az importált simulink könyvtár között a 4.4. ábrán látható *map* megfeleltetés modellt kell létrehozni. Ezen az ábrán szereplő szaggatott vonallal rajzolt nyilak minden esetben típus-példány kapcsolatot jelölnek, a sima egyenes vonallal bejelölt nyilak pedig hivatkozásokat jelentenek. Fontos kiemelni, hogy a mapping metamodell tartalmaz hivatkozást arra a modellre is, aminek a példányaként létrejött, azaz a saját metamodelljére. A vízszintes szaggatott vonalak és az M2, M1 és M0 feliratok az egyes MOF-ban definiált metaszinteket jelölik.

4.2.2. A DSL hasznosítása

Blokkokat másolva lehetett SIMULINK alatt könyvtári elemekből modellt létrehozni. Emiatt körülményes megkülönböztetni egy rendszert és egy könyvtárat, hiszen mind struktúrájuk, mind az őket alkotó elemek típusai közel azonosak. Ennél a másoláson alapuló kapcsolatnál, ahol egy paraméterben van tárolva, hogy minek a másolataként jött létre az adott elem, kifinomultabb a **típus-példány kapcsolat**.

Az EMF támogatja, hogy egy felhasználó által definiált DSL alapján, annak megfelelő példányokat hozzon létre. Az előző fejezetekben szereplő Ecore metamodellek, például a Simulink



4.4. ábra. A mapping metamodell és kapcsolatai a metamodellekkel, valamint a mapping példánymodell és a kapcsolatpéldányok révén létrejött hivatkozásai a példánymodellekre

(lásd 4.1. ábra) is, az Ecore modell példányai, míg a Simulink példánya egy importált modell (pl. ecblockset.simulink, lásd 4.1.3. fejezet).

Érdemes a blokkokat összekötő kapcsolatokat a SIMULINK-től eltérően kezelni. EMF alatt egy kapcsolatot jelenthet az is, ha két elemet egy referencia köt össze. Jelen esetben minden olyan összeköttetésnek, aminek a modellben a validáció során szerepe egy referenciát feleltetünk meg. Erről a metamodell megalkotásakor kell gondoskodni, hiszen annak ismeretében, hogy milyen elemek között, illetve milyen típusú összeköttetés létezhet kell felvenni a metamodellben a reláció definícióját.

A feladat összefoglalva az, hogy legyen egy könyvtárat valamilyen módon reprezentáló modell, aminek az elemeit példányosítva építhetők meg a SIMULINK rendszerek, vagyis egy könyvtár feleljen meg egy DSL-nek, míg a modell legyen ennek egy példánya. Erre a feladatra a következő két alfejezetben leírt megoldások javasolhatók. Mindkét esetben az importált, Simulink metamodell példányaként létrejött EMF-beli könyvtár reprezentáció adottnak tekintett.

Kiegészítő metamodell generálása

Egyik lehetőség a megoldásra a rendszer modellezésére alkalmas, rendszer specifikus metamodell kiterjesztése. Ez a megoldás lényegében egy újabb metamodell létrehozását jelenti, amely elemei az eredeti metamodell elemeinek specializációi. Fontos, hogy ezt a finomított metamodellt azonban automatikusan hozhat létre a rendelkezésre álló megfeleltetési modell alapján.

Első feladat ezzel kapcsolatban egy új, üres metamodell létrehozása. Amennyiben ez rendelkezésre áll, a következő algoritmus alkalmazandó:

1. Minden olyan elemre, ami a könyvtár importált, Simulink modelljében szerepel meg kell keresni azokat az elemeket, amikhez a megfeleltetés modellben nem tartozik összerendelés (amennyiben tartozik összerendelés az adott elemhez, úgy azt már nem kell figyelembe venni, hiszen az eredeti könyvtár metamodell tartalmazza), és nincs olyan elem, aminek a `parent` referenciája rá mutat.

2. Az összes ilyen elem esetén vizsgálni kell, hogy a szülőjéhez (parent reference) tartozik-e

megfeleltetés objektum.

3. Amennyiben igen, úgy létre kell hozni az új, finomított metamodelben egy olyan EClass-t, ami abból az Ecore osztályból származik le, akire az adott mapping objektum hivatkozik.

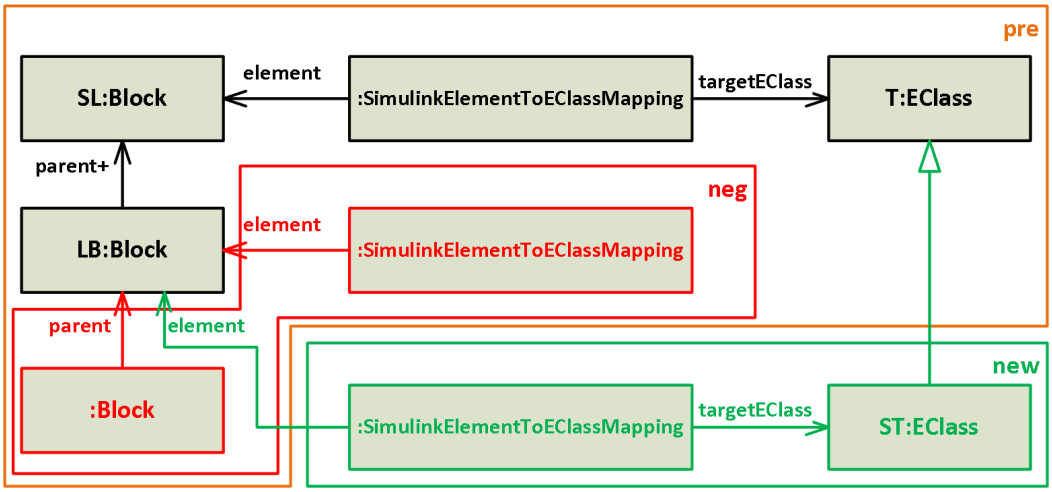
4. Ha nincs megfeleltetve egyetlen Ecore osztály sem a szülőjének, akkor pedig addig kell a szülő kapcsolatokat (a szülőből kiindulva) vizsgálni, amíg nem található olyan példány, akihez található megfeleltetett EClass, és ekkor a 3. pontban leírtaknak megfelelően kell eljárni.

Ez az algoritmus minden esetben működik, ahol az eredeti metamodell egy bizonyos mélységig helyesen leírja a könyvtárat, és a megfeleltetés modellben is szerepel minden metamodellbeli EClass-hoz összerendelés.

A fent leírtak implementálásakor egyes lépéseknél gráfminta illesztés használható. Megfogalmazhatók minták, amik az olyan `Block` objektumokra illeszkednek, amikre egyetlen mapping objektum sem hivatkozik, de a `parent` relációkon keresztül elérhető olyan `Block` objektum, amihez tartozik EClass összerendelés. Ezt fejezi ki a `parent+` kifejezés, ami a `parent` reláció tranzitív lezártját jelenti [2]. Ekkor létre kell hozni egy EClass-t, ami abból az EClass-ból öröklődik, amihez a parent relációkon keresztül elért `Block` van rendelve, illetve el kell készíteni egy mapping objektumot is, ami összerendeli az újonnan létrehozott EClass-t és azt a `Block` objektumot, amihez még nem tartozott mapping. A leírtakat a a 4.5. ábra szemlélteti, a rajta szereplő objektumok általános neveinek rövidítésének feloldásai:

- `SL` - SubLibrary
- `LB` - LibraryBlock
- `T` - Type
- `ST` - SpecializedType

Az ábrán szereplő **pre** jelzésű minta a feltétel, aminek illeszkedése esetén a **new**-al jelölt részben szereplő objektumokat és referenciákat kell létrehozni



4.5. ábra. A keresett minta mint feltétel narancssárgával keretezve, és illeszkedése esetén a tevékenység zölddel jelölve

Példa: Vizsgáljuk azt az esetet, amikor a 4.5. ábrán szereplő minta illeszkedik a következő, a korábbi példákban szereplő (2.1. fejezet mintapéldája) konkrét objektumokra:

- `Signal Booster`, mint `LB`
- `Input Circuits`, mint `SL`
- `InputCircuit`, mint `T`

Itt azt is feltételezzük, hogy az `Input Circuits` objektumnak az `InputCircuit` EClass van megfeleltetve. A végrehajtandó lépések ezúttal a `SignalBooster` (mint `ST`) EClass létrehozása és a hozzátartozó öröklés kapcsolat beállítása, továbbá egy `SimulinkElementToEClassMapping` objektum létrehozása, és a benne szereplő két hivatkozás beállítása az ábrának megfelelően

Példányszintű kapcsolatok

A DSL pontosítására egy másik lehetőség, hogy nem egy új metamodell generálódik, hanem egy olyan modell jön létre, ami a könyvtár metamodellre és a Simulink példánymodellre hivatkozik, de ez utóbbival egy metaszinten helyezkedik el. Tehát az újonnan felvett modell is egy Ecore metamodell (de ez a metamodell természetesen nem azonos a Simulinkkal) példánymodellje.

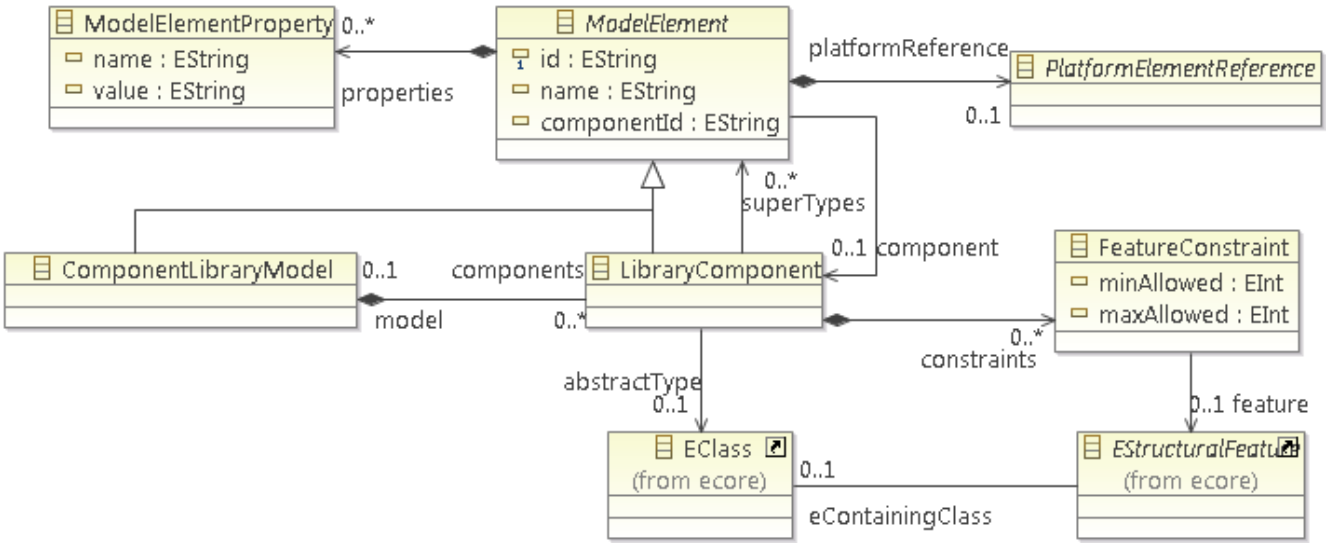
Ehhez egy újabb Ecore metamodell megtervezése szükséges. A cél az előző módszerben (4.2.2) szereplő, automatikusan generált Ecore modellt egy olyan példánymodellre cserélni, ami tartalmazza a szükséges hivatkozásokat az adott DSL elemekre. Ennek megfelelően a metamodell tervezésekor a kapcsolatoknak hasonló viszonyokat kell képviselni, mint amik az előző megoldásban már szerepeltek. A metamodel diagramja a 4.6. ábrán látható

Ezek alapján szerepelnie kell egy olyan absztrakt elemnek, ami egy könyvtár vagy modell tetszőleges elemét reprezentálja, legyen ez a `ModelElement` EClass. Tartalmazhat bizonyos könyvtár specifikus jellemzőket, amiket a tartalmazott `ModelElementProperty`-ben tárol. Annak érdekében, hogy egyértelműen meg lehessen feleltetni egy könyvtár elemének, kell tartalmaznia egy, a megfeleltetési információkat tároló `PlatformElementReference`-t is. Ezzel kapcsolatosan a 4.7. ábra mutatja a nyomonkövethetőséghez szükséges elemek viszonyait. A Simulink metamodell osztályaira a `PlatformElementReference` két specializációja, a `SimulinkElementReference` és a `SimulinkModelReference` osztályokban szerepel hivatkozás.

Ebben az esetben a `simulinkModel` és az `element` referenciák értékei származtatott értékek, hasonló módon számítódnak, mint a a 4.1.1. fejezetben tárgyalt `sourceBlock` EReference értéke.

Az absztrakt `ModelElement` Ecore osztálynak két specializációja szerepel, amik példányosíthatók is. Az egyik a példánymodell modellek gyökérelemeként szolgáló `ComponentLibraryModel`, a másik pedig az egyes könyvtári blokkokat reprezentáló `LibraryComponent` EClass. Ez utóbbi a példányai fogják meghatározni, hogy egy könyvtári elem melyik szakterület-specifikus modellelemnek feleltethető meg az `abstractType` EClass-ra mutató referencia révén. Ezek alapján a (a könyvtári komponensből eredően) Component Model-nek nevezett metamodell Ecore diagramja a 4.6 ábrán látható.

Szerepel még ezen a diagramon egy `FeatureConstraint` EClass, ami az `EStructuralFeature` Ecore metamodellbeli EClass-ra tartalmaz referenciát. Az `EStructuralFeature` az `EAttribute` és az `EReference` ősosztálya. Ezzel például a 2.5. ábrán szereplő `SafetySystem` objektumokra



4.6. ábra. A componentModel.ecore diagramja

köthető ki, hogy minimum, illetve maximum hány `Computer` EObject-re tartalmazhatnak referenciát. Ennek segítségével írhatók le a DSL modellpéldány egyes elemeinek referenciáira vonatkozó kardinalitással kapcsolatos kényszerek.

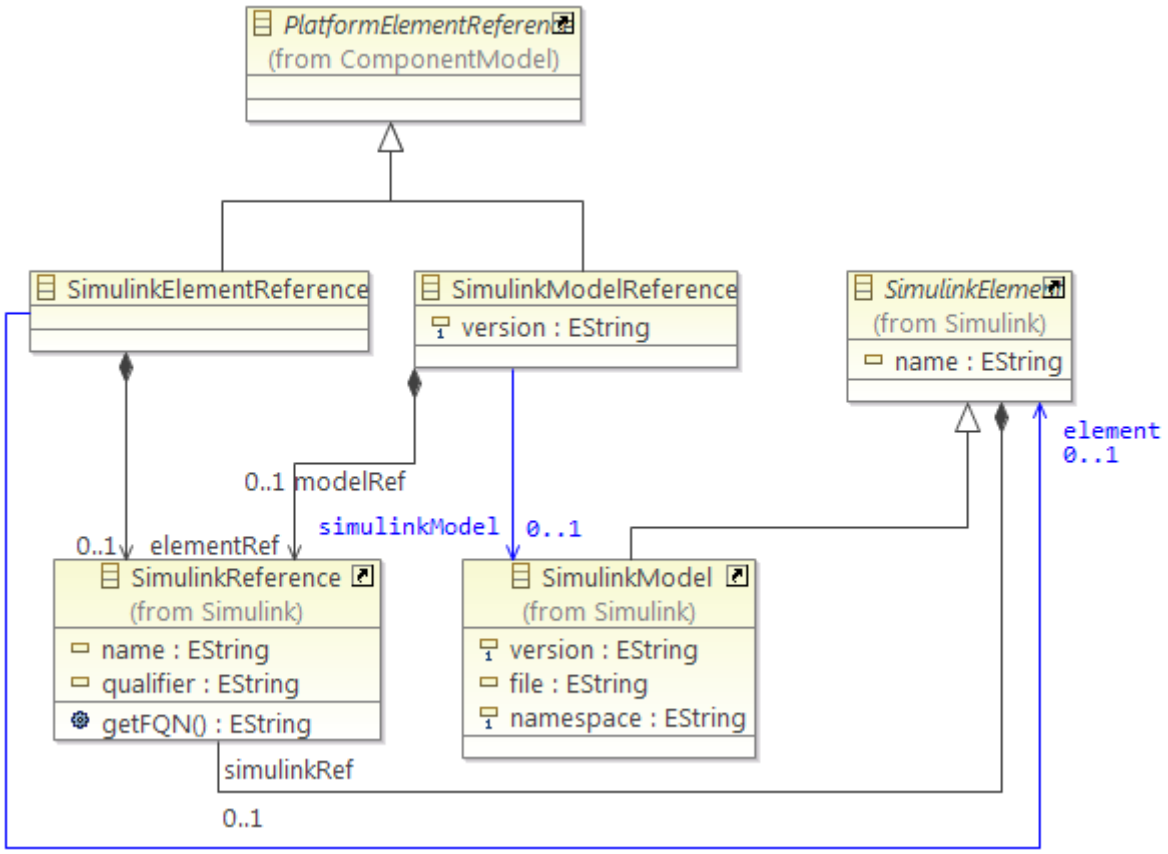
A könyvtár modell leképezése az imént definiált metamodell egy példányára az alábbi algoritmus szerint tehető meg:

1. Egy `ComponentLibraryModel` objektum létrehozása, és a neki megfelelő `SimulinkModelReference` létrehozása.
2. Azon `Block` objektumok megkeresése az importált Simulink modellben, amikhez nem tartozik mapping objektum, és nincs olyan `Block`, aminek a `parent` referenciája rá mutat.
3. Minden így megkeresett objektumhoz a `parent` referencián keresztül egy olyan `Block` keresése, amihez tartozik mapping objektum.
4. Egy új `LibraryComponent`, és egy új `SimulinkElementReference` objektum létrehozása és hozzáadása a modellhez.
5. A `Block`-ból a szülőkön keresztül elért `Block`-hoz tartozó mapping objektum által hivatkozott EClass, mint `abstractType`, és az előző lépésben létrehozott `SimulinkElementReference`-re mutató referencia beállítása a `LibraryComponent` objektumnál.
6. A `SimulinkElementReference` objektum esetén az `element` reláció beállítása.

Helyesen megalkotott DSL esetén a fenti algoritmus minden könyvtári blokkot megtalál és létrehozza a neki megfelelő `LibraryComponent` objektumot.

Az előző módszerhez hasonlóan ebben az esetben is jól alkalmazhatók a gráfminták a megfelelő `Block` Ecore objektumok és EClass-ok keresésére. A 4.5. ábrán leírt minta minta a 4.8 ábrán láthatóra módosul a rajta szereplő objektumnevek az alábbi általános esetekhez használt rövidítéseket jelentik:

- `SL` - SubLibrary



4.7. ábra. A `PlatformElementReference` és a belőle származó elemek, illetve kapcsolataik

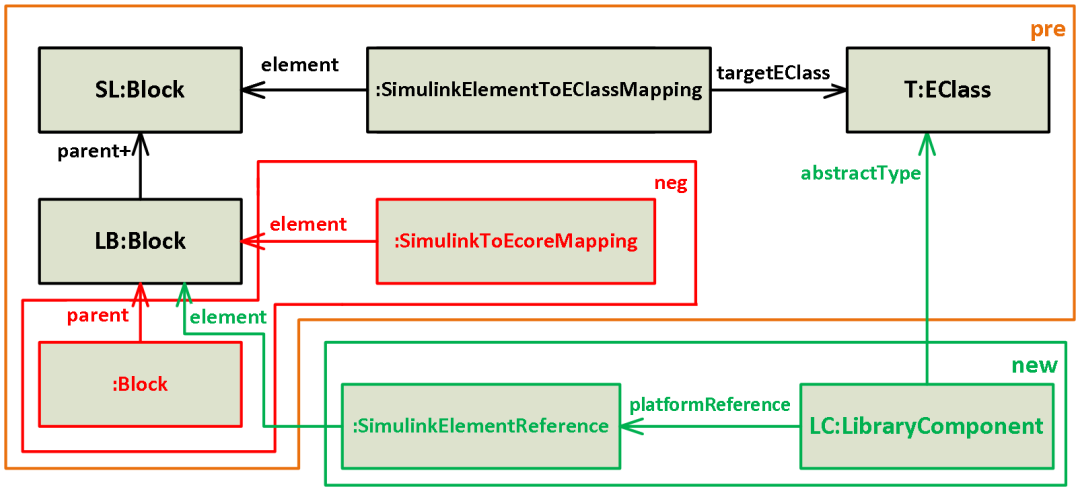
- `LB` - LibraryBlock
- `AT` - AbstractType
- `LC` - LibraryComponent

Példa: A 2.1. fejezet mintapéldájája alapján illeszkedjen a 4.8. ábrán szereplő minta a következő objektumokra:

- `Safety System` (`LB`)
- `Safety systems` (`SL`)
- `SafetySystem` (`AT`)

Ekkor egy `LC` `LibraryComponent`, és egy `SimulinkElementReference` típusú objektum létrehozása a feladat. Az `LC` objektumnak meg kell adni az `AT`-t, mint `abstractType`. Ezután be kell állítani az `LC`, a `SimulinkElementReference` objektum és a `Safety System` közötti relációkat az ábrának megfelelően.

A bemutatott módszerek közül mind a kettő alkalmas volt arra, hogy egy általános célú modellező nyelven definiált modell **elemeihez típusokat rendeljen**, és ez az **összerendelés nyomonkövethető** legyen. Ezzel elérhetővé vált a SIMULINK nyelvén megfogalmazott modellek transzformációja egy szakterület-specifikus modellre.



4.8. ábra. A keresett minta, és illeszkedés esetén a létrehozott új objektumok és beállított referenciák

A SIMULINK modellek blokkjainak jellemzőit tároló **blokk tulajdonságok (block property)** a fenti módszerekkel analóg módon **megfeleltethetők bizonyos EAttribute-oknak**. Ugyan ez igaz a SIMULINK **kapcsolatok** és az EMF-beli **referenciákák** közötti megfeleltetésre is. Ezeknek a megvalósításához mindössze a már meglevő metamodelleket kell bővíteni, azonban ezek nem hordoznak magukban lényeges újításokat.

A kapcsolatok és tulajdonságok leképezéséhez fel kell venni egy-egy új EClass-t a mapping metamodellen, amelyek képesek az EReference és `Connection`, valamint az EAttribute és `Property` megfeleltetésre. Ezekre az EClassokra a könnyű kezelhetőség fenntartása végett a `SimulinkElementToEClassMapping` osztályokból kell hivatkozni.

4.2.3. A bemutatott módszerek előnyei és hátrányai

Bármelyik leírt lehetőséget választva elérhető tehát, hogy a modellelemekhez típusok rendelődjenek. Ha az Ecore metamodell kiegészítését alkalmazzuk, akkor minden könyvtári blokkhoz külön típus jön létre. Ebben az esetben nem kell a szakterület-specifikus elemek megkülönböztetésére szolgáló paramétereket fenntartani, hiszen a típus egyértelműen meghatározza a hozzátartozó blokkot. Emellett minden kiinduló kapcsolat csak akkor vehető fel, ha a kapcsolathoz tartozó típuskényszereket a résztvevő elemek típusai kielégítik.

A nehézséget ebben az esetben viszont az okozza, hogy ha egy könyvtár akármilyen mértékben is megváltozik, akkor újra létre kell hozni a generált modelleket, aminek következtében az előző példánymodell sok esetben nem lesz betölthető, hiszen a metamodell elemei megváltoztak. Ezekben az esetekben is, amikor az alap DSL még helyes maradt, a modell újra importálására van szükség, ami a lépéseket sorrendben összefoglalva az alábbiakat jelenti:

1. A könyvtár importálása simulink modellként EMF-be
2. Ennek felhasználásával az új metamodell kiegészítés generálása
3. Ha a modell is változott, akkor a modell ismételt importálása az EMF-be a SIMULINK-ből
4. A modell EMF reprezentációjának transzformálása

Ezeket a lépéseket végrehajtva az újragenerált modell betölthető lesz.

A másik megoldás nem érzékeny a könyvtár kisebb változásaira. Mivel ebben az esetben csak elemek kötzötti referenciákról van szó, ezért a létrehozott EMF modellek mindaddig betölthetők maradnak, amíg a könyvtár metamodelljének tekintett DSL nem változik. Ez annak köszönhető, hogy a transzformált modellobjektumok két típussal is rendelkeznek. Az egyik típus, az az Ecore tekintetében vett típus, ami onnan jön, hogy melyik elem példányaként jön létre. A másik típust a modellelem által hivatkozott `LibraryComponent` objektum határozza meg. Ez abból áll, hogy a `LibraryComponent` objektum egy `SimulinkElementReference` objektumra mutat, ami a konkrét könyvtári `Block` objektumra hivatkozik, ami itt a másik típust jelenti.

4.3. Példánymodell leképezése

A könyvtár transzformációja után a könyvtár elemeit használó modellt is le kell képezni az adott DSL egy példányára. Erre a hatékony validáció érdekében van szükség. A szakterület specifikus modelleken megfogalmazhatók típuskényszerek, míg az általános célú modell esetén csak attribútumokra vonatkozó megkötéseket lehet felírni, amelyek megfogalmazása sok esetben körülményes és nem olyan hatékony (4.11. ábra).

Mindkét, az előző részekben (4.2.2) tárgyalt lehetőségekre meg kell vizsgálni, milyen modell transzformációs lépésekre van szükség. Hasonlóan a könyvtáraknál tárgyaltakhoz, a modellek leképezése esetén is alkalmazhatók gráfminták. Mindkét módszer feltételezi, hogy a SIMULINK modell importálva van EMF-be, és a neki megfelelő Simulink könyvtár modell elérhető.

A kiegészített metamodell példányosítása

A 4.2.2. fejezetben bemutatott módszer segítségével a könyvtárak alapján létrehozott, pontosabban automatikusan kiegészített DSL példányosítása a cél. Ehhez az importált **Simulink modellre**, a létrehozott **Ecore könyvtár modellre** van szükség, illetve a könyvtár transzformációja során elkészített **megfeleltetés modell** példányra.

A modell egyes blokkjainak típusai a következő algoritmus segítségével határozhatók meg:

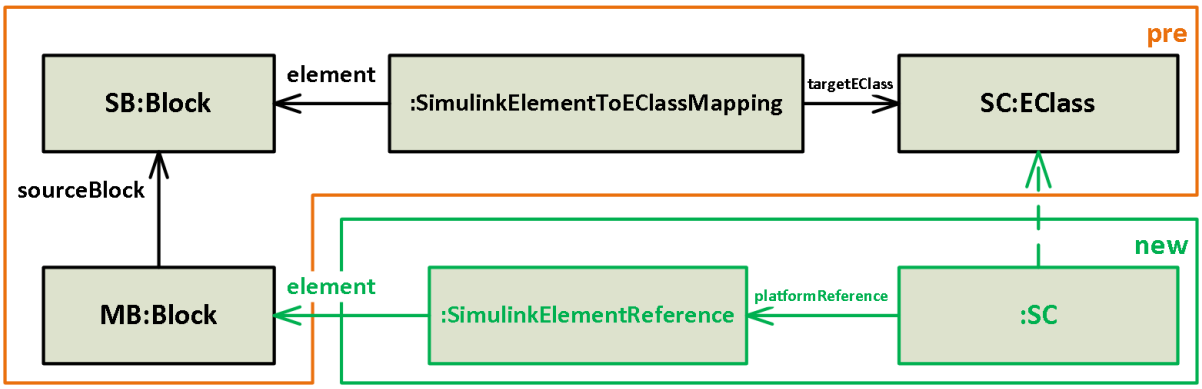
1. Minden blokk esetén ki kell választani a `sourceBlock` referenciájában hivatkozott, a könyvtár importált modelljében található `Block`-ot
2. Ezen könyvtári `Block`-ok esetén, a generált mappping modell alapján meg kell keresni a nekik megfeleltetett EClass-t
3. Ebből az EClass-ból létre kell hozni egy példányt, amely példány a `Block` transzformált reprezentációja
4. A nyomonkövethetőség érdekében ezt az EClass példányt valamilyen módon kötni kell ahhoz a blokkhoz, amelyiknek a DSL-beli képeként létrejött

A blokkok transzformációja után a közöttük lévő kapcsolatok felvehetők, azaz ennek megfelelően az egyes EObjectekben szereplő EReference értékek beállíthatók, illetve az egyes osztályokhoz tartozó EAttribute-ok is létrehozhatók a `Property`-k alapján.

A simulink modell `Block`-jainak transzformációs lépéseinél ismét felhasználhatók gráfminták. Erre alkalamas mintát jelenít meg a 4.9. ábra. Az egyes elemek általános elnevezései:

- `SB` - SourceBlock
- `MB` - ModelBlock
- `SC` - SourceClass

Az ábrán szereplő szaggatott vonallal jelölt nyíl példányosítást fejez ki, az `SC` EClass egy példánya jön létre a minta illeszkedése esetén. A sima egyenes vonallal berajzolt nyilak pedig az EReference kapcsolatokat reprezentálják.



4.9. ábra. A példánymodell transzformációjához használt minta a finomított metamodellt használó módszernél

Példa: Legyenek a mintában szereplő blokkok a korábbi példának (2.1. fejezet mintapéldája) megfelelő elemei:

- Az `SB` legyen a `Communicating Signal Booster`
- Az `MB`-nek feleljen meg a SIMULINK modellen szereplő `CSB_0` blokk importált megfelelője
- `SC` pedig legyen a `CommunicatingSignalBooser` EClass

Ekkor létrejön két EObject, egyik típusa `CommunicatingSignalBooser`, a másik pedig egy `SimulinkElementR`
Az ábrán szereplő EReference értékeket pedig a jelölt irányba kell beállítani.

A példányszintű kapcsolatok felhasználása

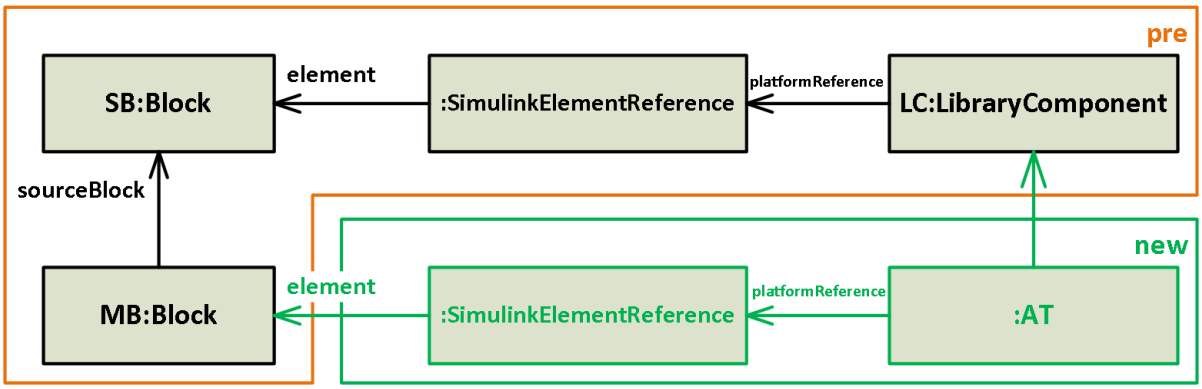
Ennél a megoldásnál a 2.5. fejezetben definiált **DSL példányosítása** történik az importált simulink könyvtár modell alapján létrehozott `ComponentLibraryModel` **példány felhasználásával**. Az SIMULINK modell transzformációjához ezeken kívül még magára az **importált simulink példánymodellre** van szükség.

A modellben szereplő blokkok tanszformációjának lépései:

1. Minden modellbeli `Block` objektumra a belőle induló `sourceBlock` referencián keresztül elérhető `Block` objektum megkeresése
2. Az így megtalált, úgynevezett *Source Block*-hoz hozzárendelt `LibraryComponent` objektum megkeresése
3. Egy, a `LibraryComponent` `abstractType` referenciájával hivatkozott EClass példány létrehozása
4. Egy `SimulinkElementReference` példány létrehozása, majd az újonnan létrejött objektumokhoz tartozó objektumok referenciáinak megfelelő beállítása

Az algoritmushoz kapcsolható minta és illeszkedése esetén a létrehozandó objektumok és kapcsolatokat a 4.10. ábra szemlélteti. Az ábrán szereplő általános elnevezések rövidítéseinek feloldásai:

- `SB` - `SourceBlock`
- `MB` - `ModelBlock`
- `LC` - `LibraryComponent`
- `AT` - `AbstractType`



4.10. ábra. A példánymodell leképezéséhez alkalmazott gráfminta a példányszintű kapcsolatok esetén

Példa: A korábban is használt (2.1. fejezet mintapéldája) elemek neveit helyettesítve a mintába az alábbi módon:

- Az `SB` legyen a `Communicating Signal Booster`
- Az `MB`-nek feleljen meg a SIMULINK modellen szereplő `CSB_0` blokk importált megfelelője (ugyan úgy, mint az előző példában a 4.3. fejezetben)
- `LC` egy `LibraryComponent` példány, aminek az `abstractType` referenciája a DSL-ben szereplő `InputCircuit` EClass-ra mutat, ahol most az `InputCircuit` helyettesítendő az `AT` helyére

A minta illeszkedése esetén létre kell hozni egy `InputCircuit` objektumot és egy `SimulinkElementReference` objektumot, majd a referenciáikat a fent leírtaknak és a 4.10 ábrának megfelelően kell beállítani

4.4. Validáció

A szakterület specifikus metamodellek és példánymodellek felett már felírhatók kritériumok a modell helyességére. Erre ismételten az EMF-IncQuery használható. 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. A mintákat az előbb leírt módszerekkel előállított modellek esetében három féle módon is meg lehet fogalmazni és ezáltal validácós célra felhasználni:

- Az importált Simulink EMF modellek esetén
- A kiterjesztett metamodell példánymodelljeire
- több példánymodel és az elemeik közötti kapcsolatokra

Az egyes esetek vizsgálatakor bemutatott példa általános leírása a 2.6.ábrán szereplő minta.

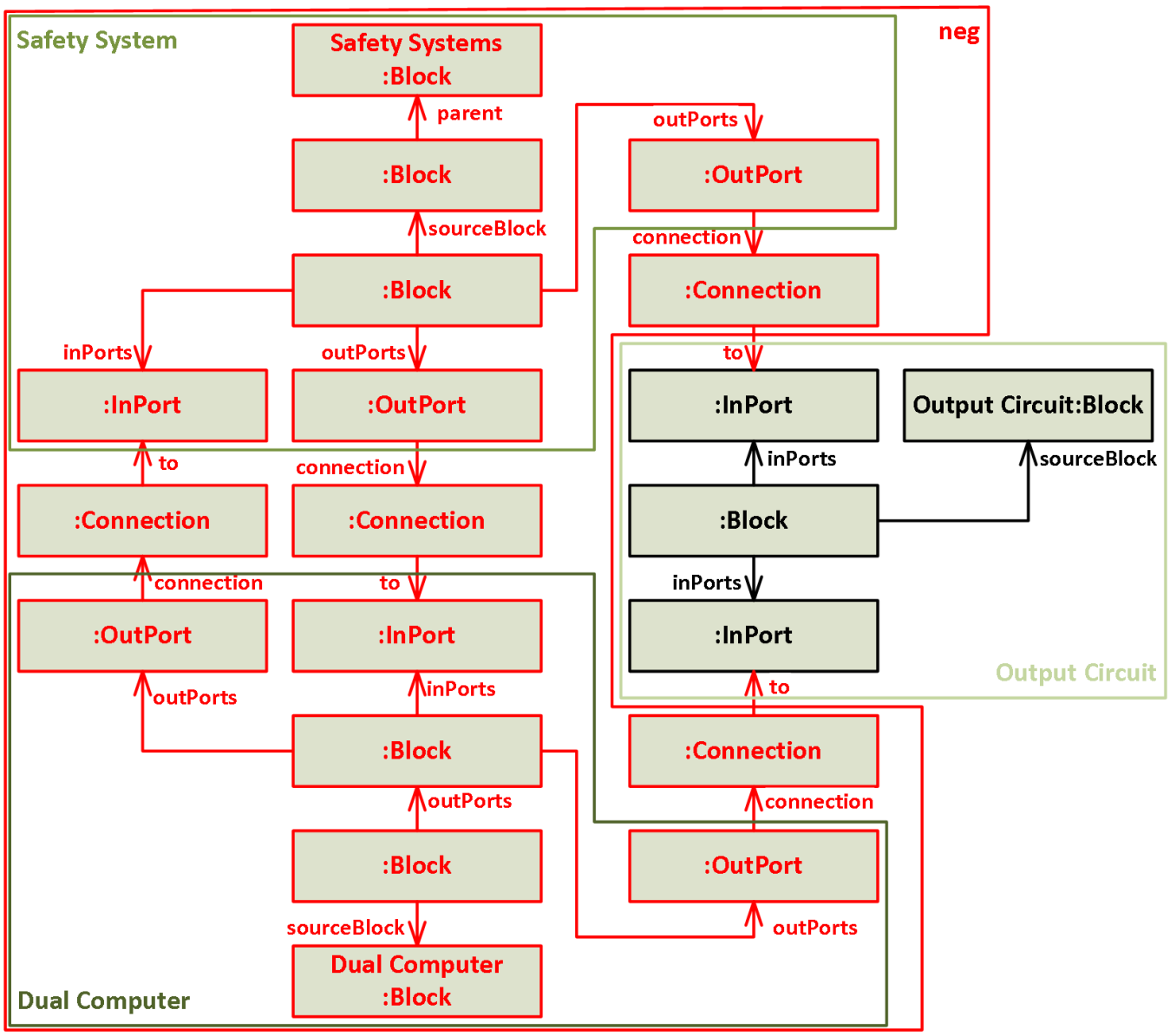
Könyvtár, mint EMF simulink modell

Az EMF-be importált, Simulink modellek alapján létrehozott Simulink modellekre is illeszthetők minták validáció céljából. Minden Simulink modell célja, hogy megpróbálja minél pontosabban, a Simulink modellek jellemzőihez 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.

Ennek a következménye, hogy a Simulink modellek felett gráfminták megfogalmazásakor minden körülmény figyelembe vehető, az illeszkedési feltételek megfogalmazhatók. Ennek a módszernek a hátránya azonban, hogy a simulink metamodell bizonyos esetekben túl részletes, sok olyan információ is szerepelhet benne, amire nincs szükség. Mindemellett a modellben szereplő elemek attribútumai vehetők csak figyelembe, hiszen az elemek típusai nem hordoznak információt az adott elem funkcionalitásával kapcsolatban.

Ebben az esetben tehát a minták megfogalmazása körülményes és túlságosan terjengős, a modellel szembeni struktúrális kényszerek viszont biztosan megfogalmazhatók. A példában szereplő minta grafikus leírása szerepel a 4.11. ábrán. Ebben az egyszerű esetben is egy igen összetett mintát lett a kényszer megfogalmazásához leírni. Az ábrán az áttekinthetőség kedvéért be vannak keretezve, illetve el is vannak nevezve az egyes logikailag összetartozó, a Simulink-ben egy blokkhoz tartozó elemek. Az egyes blokkok „típusainak” azonosítására jelen esetben a nekik megfeleltetett könyvtári elemek használhatók. A típus szó azért került idézőjelbe, mert ennél a megoldásnál valójában nincs típushierarchia, és ez is okozza a validációs szabályok megfogalmazásának nehézségét.

A 4.11. ábrán az Output Circuit blokk, Safety Systems alkönyvtárból származó blokk a Dual Computer blokkot alkotó objektumok vannak jelölve. A minta akkor fog illeszkedni, ha a modellen a feketével jelölt OutputCircuitet alkotó objektumokhoz nem találhatók meg az ábrán specifikált elrendezésben az objektumok és a közöttük futó kapcsolatok. A nyilakra írt elnevezések a keresett EReference-ek nevei, ahol a nyilak a kapcsolat irányát is jelölik. Ez a minta abból a szempontból hiányos, hogy nincs megkötve, pontosan milyen nevű portokon keresztül csatlakozzanak a blokkok.

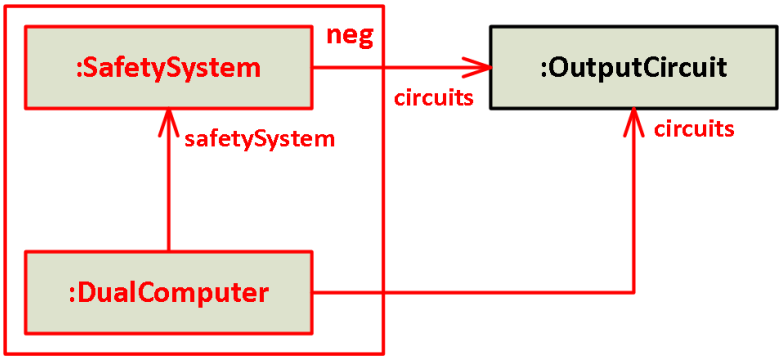


4.11. ábra. A Simulink modellpéldányok validációjához használt minták már néhány blokk esetén is igen bonyolultak tudnak lenni

Könyvtár, mint metamodell

A könyvtárat reprezentáló DSL egy példányára leképzett Simulink modell előnye az, hogy minden elem típusához egyértelműen megmondható az adott elem feladata. Továbbá az eredeti modellben található összeköttetések referenciákra képződtek le, ezzel is növelve az egyértelműséget.

Ezekben az esetekben tehát igen könnyen és hatékonyan meg lehet fogalmazni a modellhez kényszereket. A kényszereket jelentő, hibás modellrészletekre illeszkedő minták tömörek és kifejezők. Ezt mutatja a 4.12. ábra is, ami a Simulink modellre illesztett mintát bemutató diagramhoz képest jóval egyszerűbb, 20 általánosnak mondható elem helyett 3 típussal ellátott elem viszonyát kell leírni, és a típuskényszerek hatékonyan ellenőrizhetők. A minta értelmezése továbbra is az, hogy a kimeneti áramkörökhöz csatlakoznia kell valamilyen biztonsági rendszernek, illetve egy Dual Computer elemnek. A kapcsolatokat helyett a modellen referenciák szerepelnek, amik szintén jelentős egyszerűsítést eredményeznek.



4.12. ábra. Típusszármaztatás után a minta sokkal egyszerűbbé és ezáltal átláthatóbbá válik

A módszer hátránya azonban akkor jelentkezik, ha az importált SIMULINK könyvtárat jelentő modell akármilyen mértékben megváltozik. Ekkor ugyanis az összes előzőleg generált modelleket és a hozzájuk tartozó kódot ismét létre kell hozni. Emellett ha egy olyan esetre szeretnénk validációt végrehajtani, amikor a használt mintában valamilyen kiegészítő metamodellbeli osztály is szerepel, akkor a modell változásának esetén előfordulhat, hogy a mintát is újra kell írni.

Ezáltal a megoldás azokban az esetekben működik jól, ahol a könyvtár egyáltalán nem, vagy csak nagyon ritkán változik.

Könyvtár, mint DSL példánymodell

Ebben az esetben az előre megtervezett metamodellnek elég megfelelnie egy bizonyos tartalmazási szintig az adott SIMULINK-beli könyvtár felépítésének. Itt egy olyan, a könyvárat reprezentáló példánymodellel egészül ki a megoldás, ami hivatkozik a DSL-re, de ezek a hivatkozások példány szinten vannak jelen, nem pedig modell szinten.

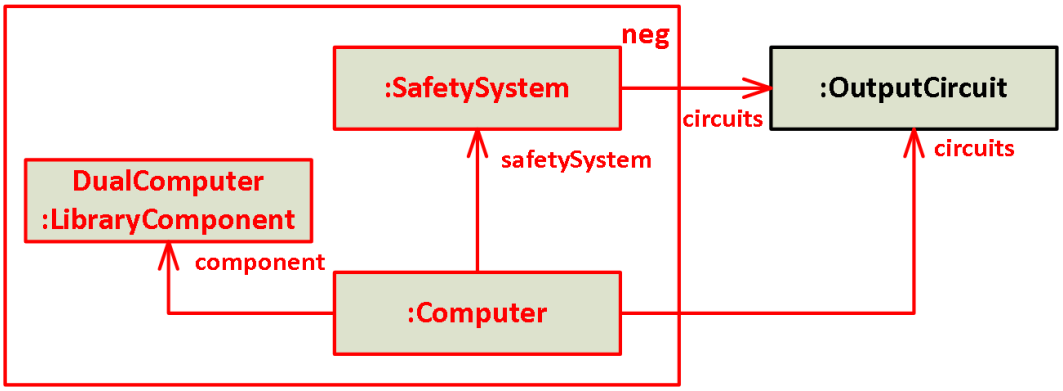
Ennek köszönhetően az összes modell és a belőlük létrejövő kód újragenerálására csak azokban a kivételes esetekben van szükség, amikor a könyvtár szerkezetében lényeges változás történik.

Amivel kevesebbet nyújt ez a módszer az az, hogy nincs olyan sok típus, mint a tisztán DSL-t példányosító módszer esetén, azaz itt vannak az elem funkcionalitásra és tulajdonságaira utaló típusok, de nem olyan kifejezők. Validációkor azonban a gráfminták leírásánál sokat egyszerűsítenek. A 4.13. ábrán látható a kapcsolódó minta leírása. A számítógéppel szemben támasztott elvárás, hogy pontosan DualComputer számítógép legyen, de ez nem szerepel a metamodellen, így ezt külön, a kapcsolódó LibraryComponent objektum nevén ellenőrizni kell.

4.5. Visszajelzés a validáció eredményéről

Az absztrakt elemeken végzett validáció eredményét vissza kell jelezni a felhasználónak. Lehetőség szerint azon a felületen kell ezt megtenni, ahol a modellek szerkesztésére lehetőség van, jelen esetben a SIMULINK rendszerében az eredeti modellen kell jelezni a hibát. Ennek érdekében ismerni kell az egyes transzformált elemek transzformáció előtti ősképét, ezért is fontos a leképezés során a **nyomonkövethetőség (traceability)** fenntartása. Fontos a visszajelzés esetén az is, hogy minél pontosabban lehessen a hibát az eredeti modellen jelezni.

A nyomonkövethetőséget a bemutatott megoldások esetén a mapping modell, illetve `Com-`



4.13. ábra. Az egyes kapcsolódó elemeknek külön típusa van, de ez nem teljesen specifikálja, melyik elemről is van szó

`ponentLibraryModel` példányok teszik lehetővé. Egy adott DSL példánymodellen szereplő objektumhoz visszakereshető az importált simulink modellen szereplő `Block` típusú ősképe, ami 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 2.1. fejezet példája alapján egy `OutputCircuit` típusú objektum szerepel a DSL példánymodellen. Meg kell keresni, melyik `Block` objektum felel meg ennek az EObjectnek.

4.6. Megvalósítási részletek

4.6.1. A validáció előkészületei

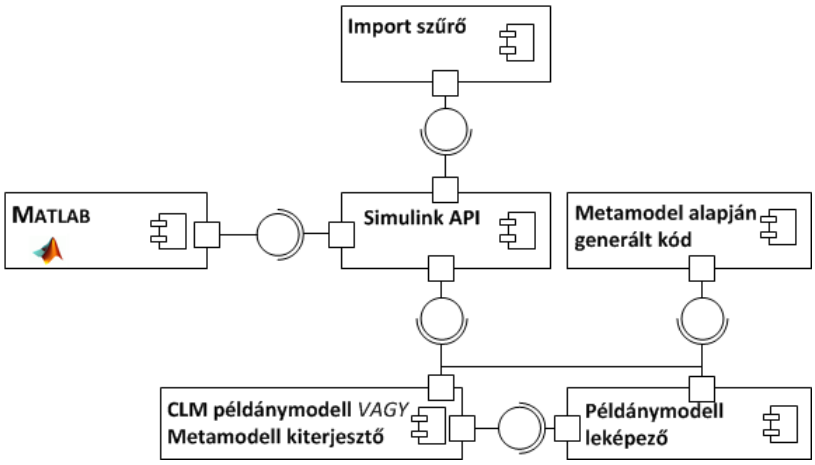
A SIMULINK modellek validációjának lehetővé tételéhez az egyes metamodellek gondos megtervezésén kívül az alábbi konkrét lépések voltak szükségesek:

- Kommunikáció a MATLAB-bal
- SIMULINK modell importálása, EMF-beli megjelenítés
- Az importált EMF modell transzformációja

Az elvégzett munka során elkészült illetve felhasznált szoftverkomponensekről és kapcsolatukról ad áttekintést a 4.14. ábra.

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

Ehhez elérhető egy MatlabControl [10] nevű, Java nyelvet használó kommunikációhoz 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 bocsájta.



4.14. ábra. A modellek importálását és transzformálását megvalósító egyes szoftverkomponensek

Ez az API lehetővé tette alapvető MATLAB parancsok kiértékelését. A használta bizonyos szempontból nehézkes, mivel egyszerű szövegként kell megadni a MATLAB saját nyelvén a kívánt utasítást. Cserébe segítségével tetszőleges parancs kiadható, ezért ezt felhasználva a jelen dolgozathoz kapcsolódó munka keretében elkészült egy erre épülő újabb csomagoló osztály, ami a jelen kontextusban használt MATLAB parancsokat megfelelően paraméterezezve 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 implementáltam 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 például szükség, mert a könyvtárak esetében előfordulhat, hogy egy alkönyvtár olyan blokkokat is tartalmaz, amik eleve alrendszerek, de nem alkönyvtárak. Ekkor az importált modellben kívánatos lehet, hogy csak az alrendszer jelnjen meg, mint könyvtári blokk, és ne szemantikailag hibásan alkönyvtárként látszódjon. 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 kiahsználja az Elipse plug-in struktúrájának modularitását, sokféle felhasználást és könnyű módosíthatóságot eredményez.

A feladathoz szükséges volt további modellek megalkotására:

- A SIMULINK rendszerek reprezentációját lehetővé tevő metamodellre
- Az adott szakterülethez tartozó specifikus metamodellre
- A transzformáció megvalósításához és ennek nyomonkövethetőségéhez használt megfeleltetés metamodellre, illetve ennek egy előre létrehozott példányára is
- Az egyik megoldásban (4.2.2) egy könyvtári komponens metamodell definiálására szükség volt a könyvtárakban szereplő blokkok reprezentációjához

Ezeket a metamodelleket és a belőlük generált kódot jelenti a komponens diagramon **Metamodell alapján generált kód** nevű komponens.

Ezt követően el kellett készíteni azokat a komponenseket, melyek a könyvtárak simulink modelljeiből a 4.2. fejezetben tárgyaltaknak megfelelően az adott DSL-t annak kiterjesztésével pontosítja, vagy egy alkalmas segédmodellt hoz létre. Ez az első megoldásban egy **Metamodell kiterjesztő**, vagy a második esetben egy **CLM példánymodell** és a hozzátartozó kód generáló komponens.

Ez a komponens tehát felhasználja a rendelkezésre álló példány- és metamodelleket, ezekre az EMF-IncQuery segítségével előre definiált mintákat illeszt, amely minták illeszkedésekor a 4.2.2. fejezetben leírtak szerint hoz létre a metamodellekből generált kód segítségével példánymodelleket.

Az ábrán szereplő, ezidáig nem részletezett elem a **Példánymodell leképező**. A feladata felhasználni az eddig elkészített modelleket illetve modell kódokat, és magát a rendszer modelljét transzformálni. A 4.3. fejezetben szereplő lépéseket végrehajtja, amihez előre definiált EMF-IncQuery minták szükségesek. Mindkét felsorolt megoldás esetén mindössze a minták illeszkedésekor végrehajtott lépéseket kell megvalósítani, amihez a megfelelő modell és modell elemek példányosítása szükséges.

4.6.2. 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: A 4.15. ábrán látható `safeOutput` minta leírja, hogy minden `OutPort` milyen esetben lesz biztonságos, azaz mikor van helyesen bekötve a modellen. Az EMF-IncQuery szöveges nyelvet használ a minták definiálására, így az eddigi példákhoz tartozó ábrákon megjelenített minták leírása látható az ábrán. Azokban az esetekben, amikor egy `OutPort` typusú EObjectre nem illeszkedik a minta, akkor ott hibát kell jelezni, amely jelzésben fel kell tüntetni a hibásan bekötött `OutPort` nevét.

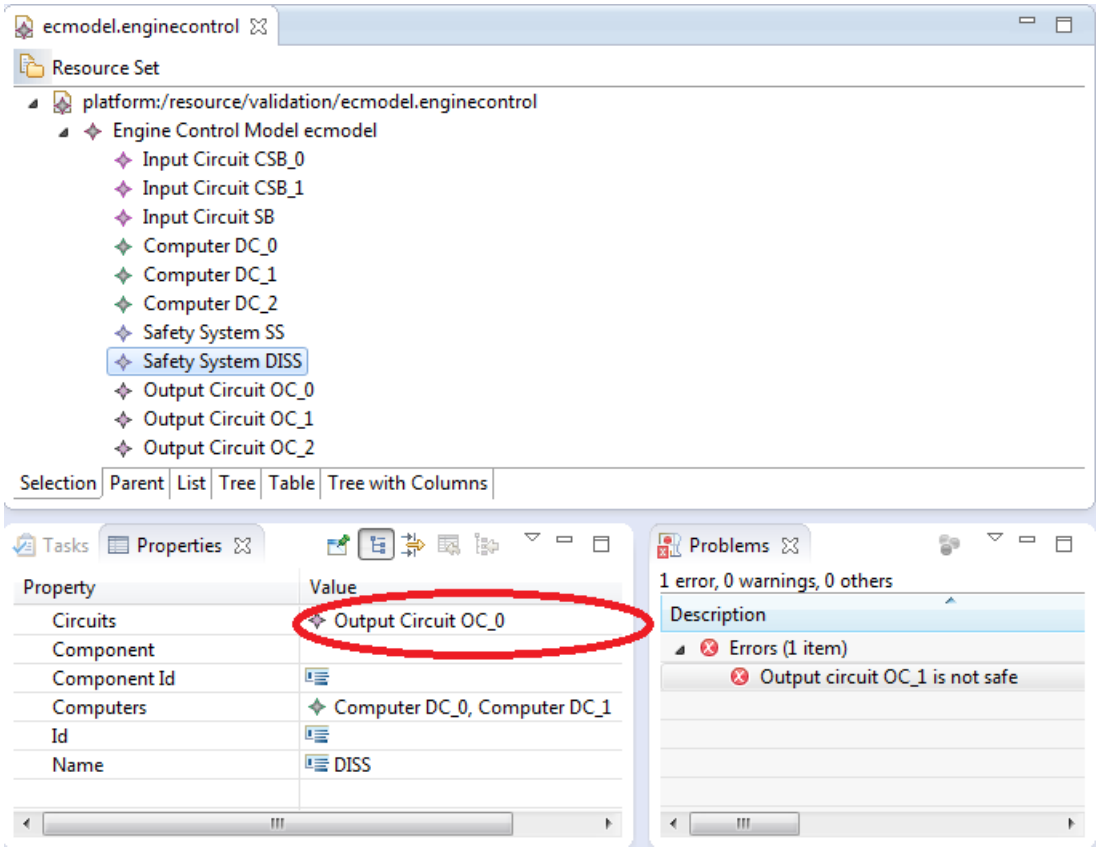
```
private pattern safeOutput(Out : OutputCircuit) {
    OutputCircuit.computer(Out,Computer);
    Computer.safetySystem(Computer,SafetySystem);
    SafetySystem.circuits(SafetySystem,Out);
}

@Constraint(location = "Out", severity = "error", message = "Output circuit $Out.name$ is not safe")
pattern unsafeOutput(Out :OutputCircuit) {
    neg find safeOutput(Out);
}
```

4.15. ábra. Minta definíciója EMF-IncQuery használatával

Egy hibásan összekötött esetről készített kép és a hibajelzés a 4.16. ábrán szerepel. Látha-

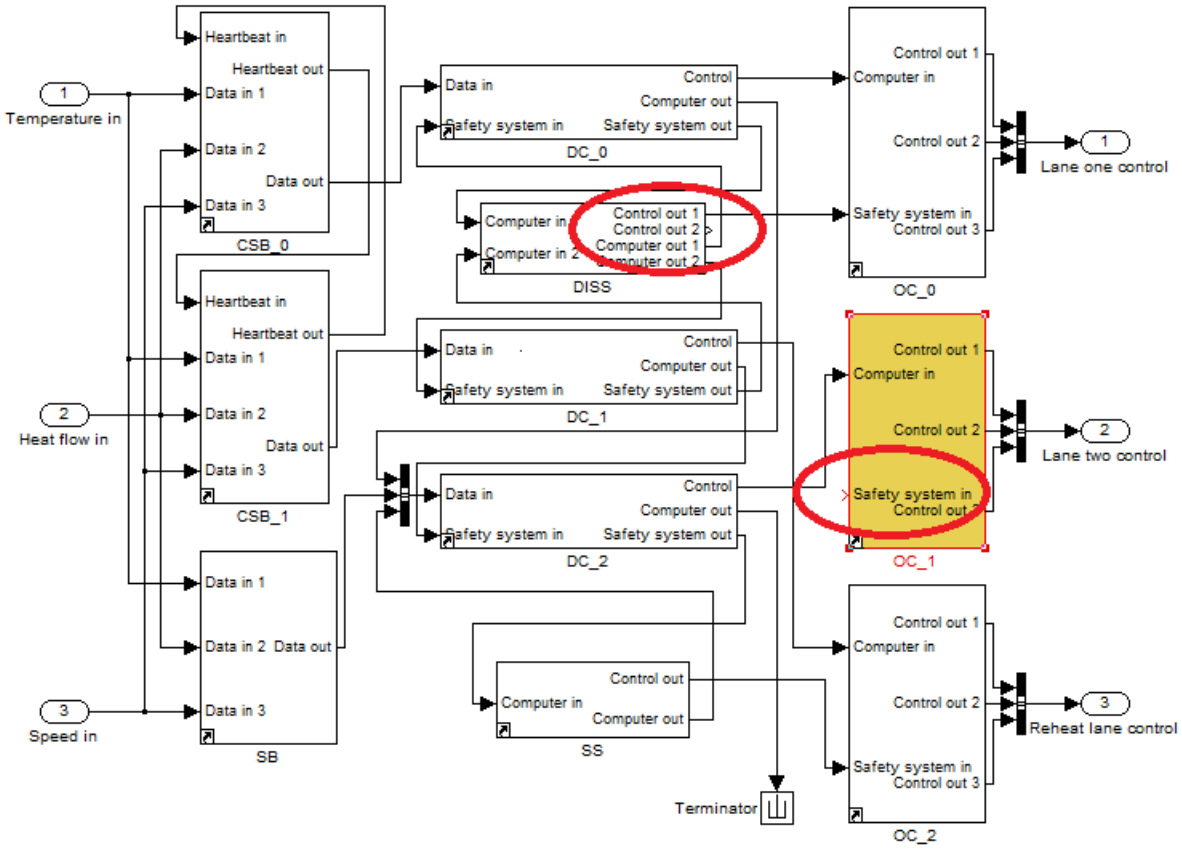
tóan a *DISS* biztonsági rendszer nincs összelötve az *OC_1* nevű kimeneti áramkörrel. Az ehhez kapcsolódó SIMULINK modell blokkvázlatánál a 2.3 a blokkok nevei az itt látható modell elemek neveivel egyeznek.



4.16. ábra. A modellszerkesztő és a hibajelzés

A hibajelzés hatására a SIMULINK felhasználói felületén is láthatóvá válik a hibásan bekötött blokk, amit egy automatikusan a MATLAB kiadott parancs vált ki. Ezt a 4.17. ábra mutatja be.

A mefelelő összeköttetés esetén a `DISS` objektum referenciája az `OC_1` objektumra is mutat, a 2.1. fejezetben szereplő mintapélda alapján. Ezzel analóg módon a SIMULINK blokkvázlat is akkor helyes, ha a `DISS` blokk `Control out 2` kimenete az `OC_1` blokk `Safety system in` bemenetére csatlakozik.



4.17. ábra. A SIMULINK grafikus felületén kiemelve szerepel a hibás bekötésű blokk

5. fejezet

Értékelés

Annak érdekében, hogy a bemutatott módszer használhatóságának validációjához a szintetikus példamodellek mellett lehetőségünk volt egy, a repülőgépipari partnerünktől kapott, több mint 6000 blokot tartalmazó Simulink modellen is tesztelni, ami egy civil repülőép hardver architektúráját írta le.

A mérések elvégzéséhez egy Intel Core i5-2500 3,3 GHz-es és 8 GB RAM-os gépet használtunk 64-bites Windows7-tel, Matlab R2011b és Eclipse 4.2-vel. Az alábbi eredményeket kaptuk, ahol a mérési eredmények 5 független mérés átlagaként adódnak:

- **Importálás**

  A motorvezérlő mintapélda esetén a modell importálása 1,36 másodpercbe tellt. Ebből a kommunikáció 84,6%-ot, azaz 1,15 másodpercet tett ki.

  A komplexebb reülőgép architektúra modell importálási ideje átlagosan 54,66 másodperc. A Matlab kommunikáció ebből összesen 42,57 másodpercet tett ki, azaz a 77,9%-ot. A maradék időben az EMF modellek létrehozása és szerializálása történik.

- **Validáció**

  A validációs szabályoknak való megfelelés a legnagyobb modell esetén is kevesebb, mint 1 másodperc alatt ellenőrizhető volt (0,68 másodperc). Ez az EMF-IncQuery eddig ismert teljesítményét figyelembe véve[1] az elvárt eredménynek megfelel.

A mérések eredményeiből az alábbi két fontos következtetést vonhatjuk le:

- A modellek importálási idejének mérésekor az derült ki, hogy a futási idő megközelítőleg 80%-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 modell importálás igényli a legtöbb időt, maga a validáció szinte azonnali módon számolódik. Ebből is látszik, hogy további teljesítménynövekedés a modell importálásának gyorsításával érhető el.

Összességében elmondható, hogy ezen egyszerű mérésekkel bizonyítottuk, hogy a rendszer képes elvégezni a validációt akár komplex Simulink rendszerek esetén is, azonban látható, hogy a simulink modellek elemszámának növekedése esetén skálázódási plafonba ütközhetünk.

6. 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 struktúlis kényszereket a rendszer komponenseire. Azonban a SIMULINK keretrendszer az ilyen kényszerek definiálását és hatékony kiértékelését jelenleg nem támogatja.

A dolgoztaban erre a problémára adtam megoldást egy olyan modell-alapú validációs módszer kialakításával, amely képes komplex struktúrális kényszerek definiálására és kiértékelésére MATLAB-SIMULINK rendszerek felett. Ezen validációs módszer kialakításához az alábbi részfeladatokat oldottam meg:

- Tetszőleges SIMULINK rendszer modell feldolgozható és importálható az EMF keretrendszerbe.
- Automatikus, deklaratív szabályokat alkalmazó transzformációval az általános SIMULINK modellek szakterület-specifikus modellekké leképezhetők. A leképezésre két eltérő megoldást adtam, attól függően, hogy a SIMULINK könyvtárat metamodellé vagy példánymodellé alakítottuk.
- Mind a szakterület-specifikus, mind a SIMULINK metamodellek felett felirhatók komplex, struktúrális kényszerek deklaratív gráfminták segítségével, amelyek az EMF-INCQUERY segítségével hatékonyan kiértékelhetők.
- A kiértékelés eredménye az eredeti SIMULINK rendszerre visszavetíthető és a MATLAB felhasználói felületén megjeleníthető.

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

Továbbfejlesztési irányok Nagymértékben segítené a validációs hibák javítását, ha a szakterület-specifikus modell módosításai közvetlenül visszavezethetőek lennének a SIMULINK modellekre. Mivel a szakterület-specifikus modelleknél általában már az egyes elemek típusai összefüggésben állnak az adott elem funkcionalitásával, illetve a modellek kifejezőbbek, így adott esetben könnyebben eszközölhetők a javítások a magasszintű modellen, mint a SIMULINK-beli modelleken. A másik

előnye lenne a funkciónak, hogy nincs szükség a modell ismételt importálására, szerkesztés közben azonnal látható, hogy az eszközölt változás megfelel-e a modellel szemben megfogalmazott kényszereknek.

A nagy modellekre történő skálázódás érdekében kulcskérdés az importálás minnél hatékonyabban történő megvalósítása. Amennyiben meghatározható, hogy a megváltoztatott modell mely részét kell újra importálni, a modell többi részének megtartása mellett. 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. Emellett nem csak az importálás történhet inkrementális technikával, hanem a metamodellek transzformációja is.

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] Gábor Bergmann, István Ráth, Tamás Szabó, Paolo Torrini, and Dániel Varró. Incremental pattern matching for the efficient computation of transitive closure. In *Sixth International Conference on Graph Transformation*, volume 7562/2012, pages 386–400, Bremen, Germany, 09/2012 2012. Springer, Springer.

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

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

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

[6] 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`.

[7] 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.

[8] 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.

[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] I. Moir and A. G. Seabridge. *Aircraft systems : mechanical, electrical, and avionics subsystems integration*. Wiley, 2008.

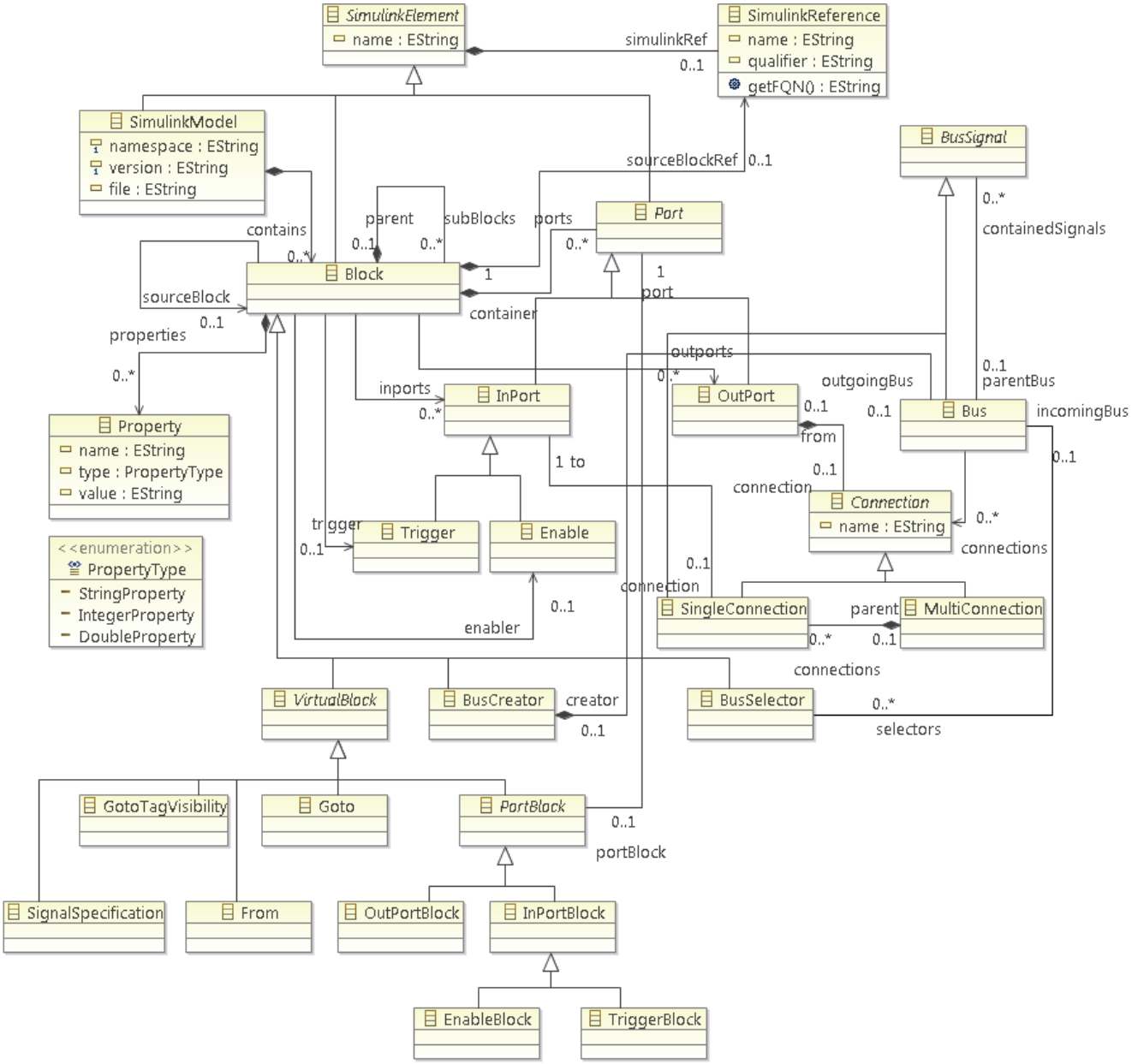
[16] Eclipse project. Ecore API. `http://download.eclipse.org/tools/emf/2.0.5/javadoc/org/eclipse/emf/ecore/package-summary.html`.

[17] Rolls-Royce. RB199 Power for the Tornado. `http://www.rolls-royce.com/Images/rb199_tcm92-6699.pdf`.

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

A. függelék

A teljes SIMULINK metamodell



A.1. ábra. A teljes SIMULINK EMF metamodell