BIZTOSÍTÓBERENDEZÉSEK SZAKTERÜLETI FORMÁLIS SPECIFIKÁCIÓINAK ELŐKÉSZÍTÉSE

[1]Lukács Gábor, [2]Bartha Tamás

Budapesti Műszaki és Gazdaságtudományi Egyetem,
Közlekedés és Járműirányítási Tanszék, 1111 Budapest, Stoczek u. 2.
[1]e-mail: lukacs.gabor@mail.bme.hu, [2]e-mail: bartha.tamas@mail.bme.hu

Absztrakt: A biztosítóberendezések fejlesztésének területén napjainkban is kihívást jelent a megfelelő minőségű formális specifikációk készítése. A közlekedésmérnökök az oktatási rendszer keretében erre vonatkozó felkészítést csak kis mértékben kapnak, a biztosítóberendezés fejlesztésnek ez egy nagyon speciális területe. A gyakorlatban a formális specifikáció készítése jelenleg több mérnöki terület együttműködésével valósítható meg hatékonyan. Biztonságkritikus rendszerek esetében a vonatkozó szabványok (pl. EN 50128) is erős ajánlást tesznek a formális módszerek alkalmazására. Kutatásunk célja egy olyan specifikációs-verifikációs környezet kialakítása, mely a biztosítóberendezési mérnökök számára megkönnyíti a formális specifikációk készítését anélkül, hogy a háttérben meghúzódó matematikai-számítástudományi ismereteket el kellene sajátítaniuk. Írásunkban egy esettanulmány segítségével foglaljuk össze a szakterületi mérnökök által írt specifikációk automatizált formalizálásának előkészítésében szerzett tapasztalatainkat, amely az alapját képezi egy, a gyakorlatban is használható specifikációs-verifikációs környezet kialakításának.

1. Bevezetés

A biztonságkritikus, mikroszámítógép alapú biztosítóberendezések fejlesztésében egyre nagyobb az érdeklődés a formális módszerek gyakorlati alkalmazhatósága iránt, mert segítségükkel a rendszerek matematikai pontossággal írhatók le. A vasúti alkalmazásokra vonatkozó szabványok erős ajánlást tesznek a formális módszerek alkalmazására (pl. EN 50128, [1]). A formális módszerek elsősorban az informatika területén használt matematikán – főként diszkrét matematikán, matematikai logikán – alapuló módszerek [2]. A formális módszerek alkalmazhatóak a rendszerfejlesztésben, így a szoftver- és hardverrendszer fejlesztésben is [3].

A jelenlegi biztosítóberendezés fejlesztési gyakorlatot a szakterületi mérnökök által írt, nem formális és félformális specifikációk (szöveges és ad hoc jelölési rendszerek) jellemzik. A biztosítóberendezés fejlesztésben a formális specifikáció több mérnöki terület (közlekedésmérnöki, mérnök informatikus) együttműködésével érhető el. A formális módszerek használata a fejlesztési folyamat során csökkentheti ezeket a nehézségeket. A formális módszerek támogatják a szigorú specifikációt, tervezést és verifikációt, a komplex rendszerek modellezését és hibák azonosítását a korai életciklus fázisokban.

Kutatásunk célja egy olyan specifikációs-verifikációs platform (SVP) kialakítása, amely a biztosítóberendezési mérnökök számára megkönnyíti a formális módszerek használatát anélkül, hogy a háttérben lévő matematikai és számítástudományi ismereteket el kellene sajátítaniuk. Az SVP egy olyan környezetet jelent, melynek bemenete a szakterületi specifikáció és a gyári követelmények, kimenete az ellenőrzött formális specifikáció. Az SVP folyamat három fő részből áll: a formális modellek automatizált generálása a szakterületi specifikációkból, a természetes nyelven megfogalmazott gyári követelmények CTL (Computation Tree Logic) nyelvű leírása [4] és a formális modelleken a CTL-modellellenőrzés [5].

Az SVP a gyakorlatban a biztosítóberendezések fejlesztése során a specifikáció ellenőrzésének a folyamatát támogatja azzal a céllal, hogy megnövelje a valószínűségét a specifikációs hibák minél korábbi életciklus fázisokban való feltárásának. A fejlesztés során elkövetett hibák mihamarabbi feltárásának a jelentősége azért fontos, mert egy kései életciklus fázisaiban (pl. üzemeltetés) felfedett hiba javítási költsége számottevően magasabb, mint a korai életciklus fázisokban (pl. tervezés) felfedett hibáké. A biztosítóberendezés fejlesztések területén a költségeken túl a biztonsági kérdések még inkább kiemeltek, különösen a biztonságkritikus rendszerrészek (funkciók) esetében. A biztonságkritikus funkciókkal kapcsolatos hibák mihamarabbi – lehetőség szerint az üzemeltetés előtti – feltárásával, súlyos következményekkel járó baleseteket előzhetünk meg.

Jelen írásunk célja bemutatni a szakterületi specifikációból kiindulva képzett formális modellek kézi előállításának folyamata során szerzett tapasztalatainkat az érzékelőelem – mint esettanulmány – segítségével. Ez az előkészítő munka jelenti az alapját a formális modellek szakterületi specifikációkból való automatizált generálásának. A modellezéshez két formalizmust használtunk: a biztosítóberendezések területén régóta alkalmazott Petri-hálókat és az ezen a területen kevésbé közismert UPPAAL automatákat.

2. Alkalmazott módszerek

A vasút automatikai rendszerek fejlesztéséhez kapcsolódó szabványok (pl. EN 50128, [1]) a V-modell szerinti fejlesztési életciklus fázisokhoz rendelten tesznek ajánlást számos technika, köztük a formális módszerek alkalmazására is. A szabványok a formális módszerek témakörében több technikát is felsorolnak (pl. EN 50128, [1] D.27 melléklete: CSP, HOL, Modellellenőrzés, Temporális logikák, stb.) és elhelyezik azokat a fejlesztési életciklusban (pl. EN 50128, [1] A.2 melléklet: Szoftver követelmény specifikáció vagy A.4 melléklet: Szoftvertervezés és implementáció). Ezeket a technikákat SIL3 és SIL4 (Safety Integrity Level) biztonságintegritási szinteken HR (Highly Recommended) jelölik [1], amely szimbólum azt fejezi ki, hogy ezek a technikák rendkívül ajánlottak a szükséges biztonságintegritási szint eléréséhez.

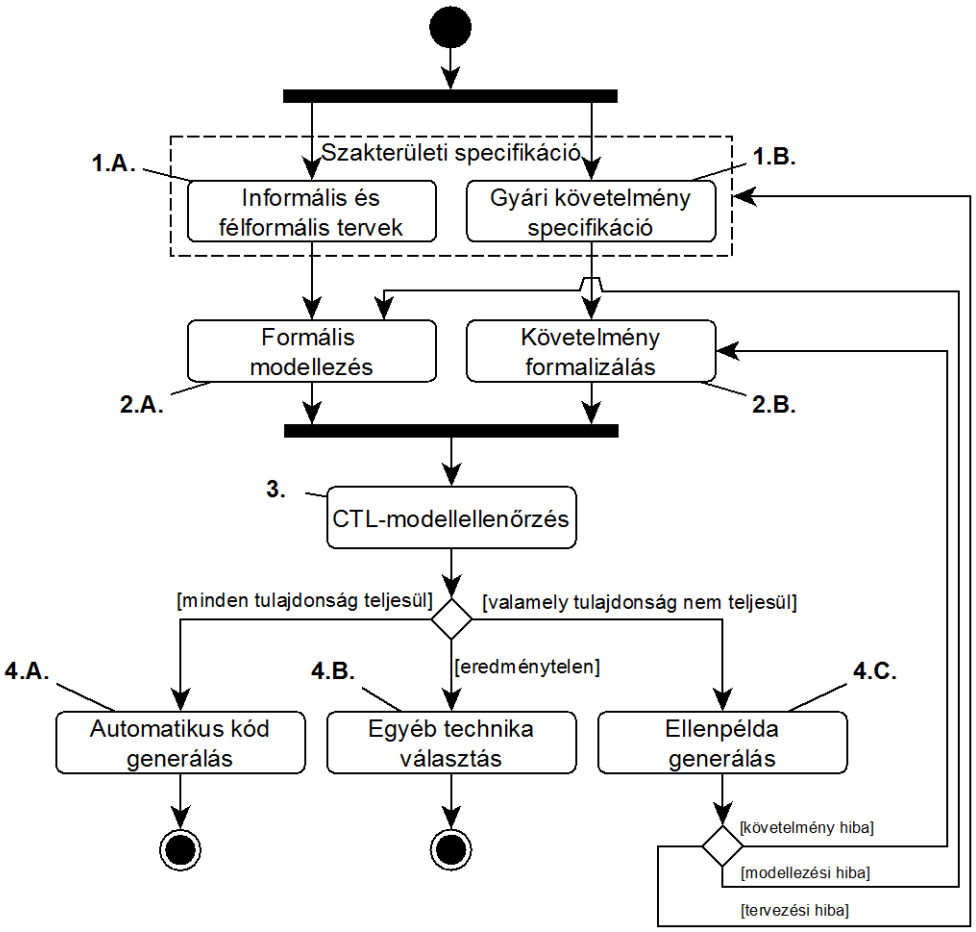
A kutatásunk céljaként megvalósítandó SVP környezet létrehozásához a formális módszerek közül a modellellenőrzést és a temporális logikákat választottuk ki. A választás alapját egy a gyakorlati életből már széleskörűen alkalmazott szakterületi specifikációs és ellenőrzési technológia vizsgálata képezte, melynek segítségével a vizsgált specifikációs-verifikációs

környezetről megállapítottuk, hogy az vegyesen használ formális leírást (pl. igazságtáblázat), félformális leírást (pl. állapottérképet), valamint természetes nyelvű leírásokat. Megállapítottuk továbbá, hogy a specifikációt véges számú elemkészlet (diagramtípus) jellemzi, és azok legtöbbje megfelel az UML (Unified Modeling Language) szabványnak [6]. Ezen kívül a vizsgált környezetben a fejlesztők a specifikáció készítésekor alkalmazzák az MDD (Model Driven Development) alapelveket [7] valamint az MA (Modular Approach) elvét [8]. Ezeket a környezeti feltételeket figyelembe véve célszerűen olyan formális módszereket vizsgáltunk, amely erős modellezési háttérrel rendelkeznek. Kedvező esetben az előbbiekhez kapcsolódó eszközök képesek lehetnek akár a modelleket vizuálisan is megjeleníteni (és/vagy a modell működését szimulálni), amely előnyös tulajdonság lehet a feltárt specifikációs hibák szakterületi mérnökökkel való tisztázásának támogatásában. Az előbbi, a specifikációs technológiára vonatkozó szempontokon túl a verifikációs folyamatok megismerése alapján számos további szempontot figyelembe vettünk, melynek alapjául a gyári követelményrendszer vizsgálata szolgált. Az SVP kialakításához szükséges szempontrendszer összeállítása jelenleg is zajlik, azonban az eddig elért eredmények lehetővé tették, hogy a meglévő szempontok alapján kiválasszuk a formális modellezés és a CTL-modellellenőrzés technológiáját és gyakorlati példákon keresztül is megvizsgáljuk, hogy ezek a módszerek milyen hatékonysággal, korlátok mellett stb. alkalmazhatóak a gyakorlati specifikációs és verifikációs feladatok megoldására.

A modellellenőrzés [5] egy formális módszer (a számítás tudomány egy módszere), amely a rendszer létrehozott formális modelljéről és annak elvárt működését tartalmazó követelmény specifikációjáról a modell állapotterének szisztematikus bejárásával dönti el, hogy a rendszermodell a specifikációt teljesíti-e vagy sem. A modellellenőrzés arra a kérdésre keresi a választ, hogy: Hol nem találkozik a modell a megadott specifikációval? Ha a bejárás specifikációnak való nem-megfelelőséget tár fel, a modellellenőrzés (eszköztől függően) ellenpéldát hozhat a specifikációtól eltérő működésre.

A CTL egy [4], az elágazó idejű temporális logikák közé tartozó leírásmód, mellyel a rendszer tulajdonságmodelljét készíthetjük el. Az elágazó temporális logika egy olyan logika, amely esetén minden időpillanathoz „többféle jövő” tartozhat. A CTL logika atomi kijelentésekből, logikai és temporális operátorokból áll. A CTL logikára jellemzőek az útvonal- és állapotkvantorok.

A CTL-modellellenőrzésnek [3] két bemenete van (1. ábra): a formális modellek és a formalizált követelmények. A szakterületi fejlesztőmérnökök ezeket a bemeneteket közvetlenül nem állítják elő a fejlesztés folyamán, viszont ami minden fejlesztés esetén hasonló formában rendelkezésünkre áll, az a fejlesztett rendszerre vonatkozó gyári követelményrendszer (jellemzően az elvárt tulajdonság leírás) és az informális és félformális tervek (jellemzően az elvárt viselkedés leírás). Célunk, hogy a meglévő dokumentációk alapján egy automatizálható folyamatot definiáljunk és egy azt megvalósító alkalmazást hozzunk létre. Az alkalmazás bemenete a szakterületi fejlesztőmérnökök által készített tervek és a gyári követelményrendszer, míg a kimenete a verifikált formális modell.



1. ábra: SVP folyamat: CTL-modellellenőrzés ([3] alapján)

A modellellenőrzés előkészítéséhez tartozó folyamat egyik része, hogy a szakterületi informális és félformális tervekből formális modelleket állítunk elő (1. ábra 1.A. és 2.A.). Jelenleg kézi módszerekkel gyűjtjük a transzformációs szabályokat, melyek segítségével a szakterületi specifikációból kiindulva formális modelleket generálhatunk. A probléma természetéből fakadóan (leginkább szintaktikai) megszorításokat kell tennünk a szakterületi specifikációk készítésének módjára, melyeket a fejlesztőmérnököknek be kell tartaniuk ahhoz, hogy az automatizált modellgenerálás végrehajtható legyen az SVP környezet segítségével.

A modellellenőrzés előkészítéséhez tartozó folyamat másik része (1. ábra 1.B. és 2.B.), hogy a gyári követelményrendszerből formális követelményeket (CTL-követelményeket) állítunk elő. Ez a folyamat is automatizáltató hasonlóan a formális modellgeneráláshoz, azonban ehhez szigorú megkötések szükségesek a gyári követelményrendszerben szereplő követelmények megfogalmazásának módjára. Az erre vonatkozó szabályrendszer jelenleg definiálás alatt áll. Megjegyezzük azonban, hogy a gyári követelményrendszer nem minden követelménye formalizálható, jellemzően ilyen követelmények a nem-funkcionális követelmények. Emiatt a követelményeket három fő csoportra osztjuk [9]:

1. formalizálható követelmények (funkcionális és biztonsági követelmények),

2. nem formalizálható követelmények (nem-funkcionális követelmények),

3. követelmények, melyek formalizálása függ a modellezés szintjétől.

A 3. csoportba olyan követelmények tartoznak, melyek formalizálhatóak, azonban az aktuális (részletes) tervezési szinten modellellenőrzésre nem felhasználhatóak (még nem lebontott követelmények).

A CTL-modellellenőrzésnek három kimenete lehet (1. ábra 3. tevékenység) [3]:

- Ha a modellellenőrzés sikeres (minden követelmény teljesül), akkor átléphetünk az implementáció fázisba. Az 1. ábrán 4.A. lépésével jeleztük, hogy az implementáció elvégezhető akár egy minősített automatikus kódgenerátorral is. Ekkor a modellek helyességének igazolása a verifikált tulajdonságok körében továbbra is érvényes marad, és a kód helyességének a bizonyítéka.

- Előfordulhat olyan eset is, hogy a modellellenőrzés eredménytelen (1. ábra 4.B.), ilyen esetben célszerű egy másik verifikációs technikát választani. A modellellenőrzés eredménytelensége alatt azt értjük, hogy olyan nagyméretű állapottér keletkezeik, melyet a rendelkezésünkre álló memóriában nem vagyunk képesek eltárolni. Megjegyezzük, hogy az irodalomban a nagyméretű állapotterek kezelésére többféle megoldás is létezik (pl. ROBDD (Reduced Ordered Binary Diagram [10]), vagy kísérletet tehetünk egyszerűen csak az állapottér egy meghatározott részének ellenőrzésére, esetleg egy magasabb absztrakciós szintű modell létrehozására).

- Ha a modellellenőrzés során valamely követelmény nem teljesül (1. ábra 4.C.), akkor az azt jelenti, hogy valahol a modellellenőrzéshez kapcsolódó folyamatban hiba van. A hiba lehet a tervben, a követelményekben és modellezésben egyaránt.

    - Modellezési hiba alatt azt értjük, amikor a modell nem a tervnek megfelelően írja le a rendszert (nem a valóságot modellezi). A legtöbb esetben ez a hibafajta a modell javításával, pontosításával korrigálható, azonban ez azzal a hátránnyal jár, hogy módosított modellre újra el kell végeznünk a korábban már ellenőrzött követelmények vizsgálatát.

    - Ha követelményhibával állunk szemben, akkor a formalizált követelmény nem felel meg teljes egészében az informálisan megfogalmazott követelménynek vagy (kritikusabb ha) már maga az informális követelmény is hibával terhelt. A hiba javítás előbbi esetben a formális követelmény javításával kezelhető, míg utóbbi esetben elengedhetetlen az egyeztetés a szakterületi mérnökökkel.

    - Ha tervezési hibával állunk szemben, akkor a rendszer terveit kell átvizsgálni. A modellellenőrzésnek ez az alapvető célja is egyben. Ebben az esetben a terveket és következményként a belőlük származtatott formális modelleket kell javítanunk.

Megjegyezzük, hogy az SVP környezet célja, hogy előző pontban felsorolt hibalehetőségek javítási folyamatát gyorsítsa a folyamatok automatizálása révén (mivel általában a folyamatot egy-egy hiba esetén teljes egészében meg kell ismételni).

Az SVP környezet gyakorlati megvalósításához kétféle formalizmust választottunk: a Petri-hálókat [11] és a nem-determinisztikus, véges automatákat [3]. Ezek kiválasztását egy döntés-előkészítő folyamat előzte meg, melynek során azonosítottuk azokat az igényeket (pl. a modellezendő rendszer tulajdonságai, időkezelés stb.), melyek szükségesek lehetnek az SVP folyamatok hatékony működéséhez a biztosítóberendezés fejlesztési szakterületen. Mindkét formalizmushoz egy előzetesen összeállított szempontrendszer (pl. beépített modellellenőrző, hordozhatóság stb.) alapján választottunk eszközöket is: PDN-t (PetriDotNet, a Budapesti Műszaki és Gazdaságtudományi Egyetem által fejlesztett tool, [12]) a Petri-hálókhoz és UPPAAL-t (az Uppsala Egyetem és az Aalborg Egyetem által fejlesztett tool [13]) a nem-determinisztikus, véges automatákhoz.

3. Esettanulmány

Jelen írásunk célja bemutatni a szakterületi specifikációból kiindulva képzett formális modellek előállításának folyamata során szerzett tapasztalatainkat (1. ábra, 1.A. > 2.A.). Ehhez felhasználunk egy, a gyakorlati életből származó egyszerűsített példát. A cikkben bemutatott példák során egy egyszerű foglaltságérzékelő elem (objektum) specifikációja képezi az SVP folyamat bemeneti adatát, melyet a biztosítóberendezés fejlesztő mérnökök fogalmaztak meg.

Az érzékelőelem specifikációjának csak egy részletét közöljük (az egyszerűsített logikai viselkedését), amely elegendő a cikkben leírt folyamatok megértéséhez és követéséhez. Nem foglalkozunk többek között az érzékelőelem interfész specifikációjával, a hozzá kapcsolódó időzítésekkel (a feltételek teljesülése esetén minden állapotátlépés azonnal végrehajtódik) és konfigurációs elemekkel. Az érzékelőelem működését az előbbi egyszerűsítések figyelembevételével mutatjuk be.

Az érzékelőelem célja a hatókörében tartózkodó vasúti jármű vagy annak valamely részének a detektálása. Az érzékelőelem kiindulási állapotában (a bemeneteitől függetlenül) hibaállapotban van (kimenetein foglalt-hibásat ad). Az érzékelőelemet ebből az állapotából az oldási folyamattal lehet szabad állapotába helyezni (ahol a kimenetein szabad-nem hibásat ad), amennyiben az oldási feltételek teljesülnek (az érzékelőelem bemenete nem foglalt és egyik hibabemenetén sem kap hibát). Amikor az érzékelőelem alapállapotában a hatókörébe egy vasúti jármű érkezik, akkor az érzékelőelem foglalt állapotba kerül (kimenetein foglalt-nem hibásat ad). Amikor a vasúti jármű távozik az érzékelőelem hatóköréből, akkor az ismét szabad állapotba kerül. Ha az érzékelőelem valamely hibabemenetén (ponált vagy negált) hibát érzékel, akkor hibaállapotba kerül, legyen előzőleg bármely állapotban (alapállapot vagy foglalt állapot). Az érzékelőelem ponált és negált foglaltsági bemenettel rendelkezik, amennyiben ezek a foglaltsági információk értelmük szerint különbözőek, az érzékelőelem hibaállapotba kerül. Az érzékelőelem itt leírt állapotátlépései a bemeneti feltételek teljesülése

esetén azonnal és automatikusan végrehajtódnak. Az érzékelőelem logikai viselkedését az 1. táblázatban foglaltuk össze.

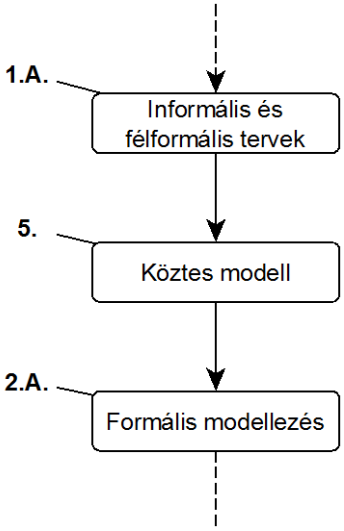
Belső állapot	Bemenetek				Kimenetek		Új belső állapot
BHIBA	**HIBA-P**	**HIBA-N**	**FOGL-P**	**FOGL-N**	**HIBA**	**FOGL**	**BHIBA**
-	nem hibás	nem hibás	szabad	szabad	nem hibás	szabad	nem hibás
nem hibás	nem hibás	nem hibás	szabad	foglalt	hibás	foglalt	hibás
nem hibás	nem hibás	nem hibás	foglalt	szabad	hibás	foglalt	hibás
nem hibás	nem hibás	nem hibás	foglalt	foglalt	nem hibás	foglalt	nem hibás
-	hibás	-	-	-	hibás	foglalt	hibás
-	-	hibás	-	-	hibás	foglalt	hibás
hibás	-	-	foglalt	-	hibás	foglalt	hibás
hibás	-	-	-	foglalt	hibás	foglalt	hibás

1. táblázat: Részlet az érzékelőelem szakterületi specifikációjából (logikai viselkedés leírása)

4. Eredmények

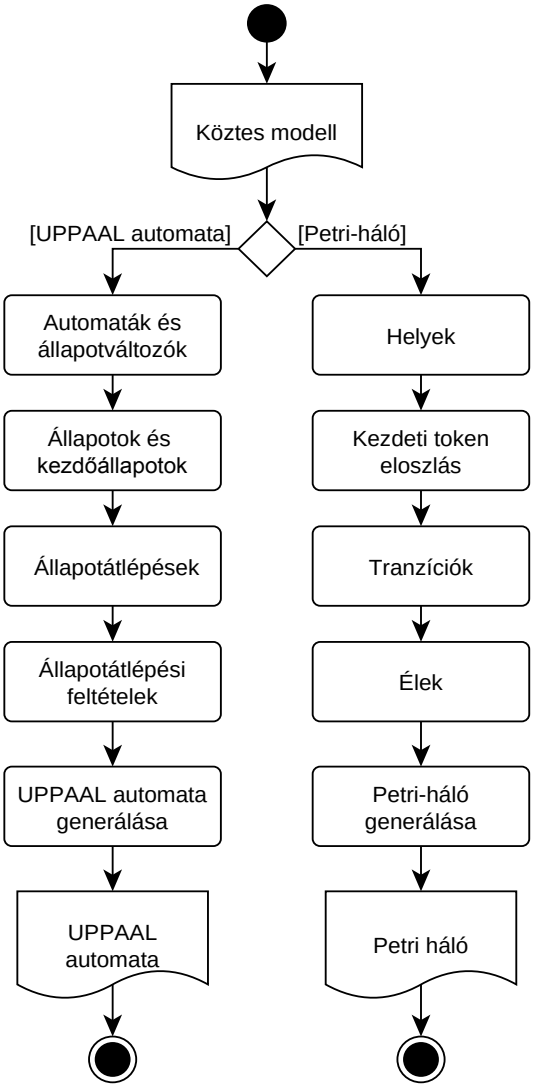
Jelen írásunk középpontjában az 1. ábra alapján a 2.A. tevékenység áll (formális modellezés), ehhez 1.A. (informális és félformális tervek), mint bemenet rendelkezésünkre áll a gyakorlati életből. Célunk, hogy 1.A.-ból kiindulva előállítsuk egy rendszer formális modelljét. Ennek folyamatát a 3. fejezetben bemutatott érzékelőelem objektum logikai viselkedésén mutatjuk be.

Vizsgálataink során arra a megállapításra jutottunk, hogy a szakterületi mérnökök által előállított specifikáció közvetlenül nem transzformálható át formális modellre. Előkészítő tevékenységre van szükség, melynek során előállítunk egy „köztes modellt” [14], amely egy specifikus modell abból a szempontból, hogy a vizsgált szakterületi specifikáció tulajdonságainak hatása mellett hoztuk létre. Ennek figyelembevételével az 1. ábra 1.A. > 2.A folyamata a 2. ábra szerint módosul (a köztes modellt 5. sorszámmal azonosítottuk).



2. ábra: SVP folyamat: Köztes modell helye

A köztes modellt egyértelműen megfogalmazható szabályok alapján állítjuk elő a tervekből (lásd. 2. fejezet, 2. bekezdés: a bemenetként szolgáló szakterületi specifikáció értékelése). A köztes modell – legyen szó bármilyen formalizmusról (pl. Petri háló, UPPAAL automata) – képezi az alapját annak, hogy belőle transzformációs szabályokkal generáljunk formális modelleket. Attól függően, hogy milyen formalizmussal (Pl. Petri-hálók vagy UPPAAL automaták) szeretnénk a rendszer modelljét megadni, különböző tevékenységsorozaton keresztül vezet az út. Ezeket a lépéseket foglaltuk össze a 3. ábrán. Megjegyezzük, hogy ezek a lépések az érzékelőelem egyszerűsített esettanulmányára vonatkoznak (így pl. nem szerepelnek benne az időzítésekre, ill. a konfigurációs elemekre vonatkozó transzformációs szabályok), ezek további tevékenységként kiegészítőleg jelennek majd meg (a 3. ábra bővítésével). A 3. ábrán nem tüntettük fel továbbá a modellek bemeneti gerjesztésének meghatározását. A bemeneti gerjesztés meghatározása általában a folyamat utolsó lépése, és megvalósítása erősen függ a modellezés céljától.



3. ábra: Transzformációs lépések a köztes modellből
Petri-hálók és UPPAAL automaták esetére

Az eddigi eredményeink alapján a biztosítóberendezés fejlesztők specifikációjából kiindulva (azokra célszerű korlátozásokat és formai megkötéseket téve) olyan transzformációs lépéseket azonosítottunk, melyek egy köztes modell előállításán keresztül lehetővé teszik, hogy a szakterületi specifikációkat formális modellekké transzformációjuk. Mivel a transzformációs szabályok – a lehatárolások – mentén mindig érvényesek, lehetőséget teremtenek arra, hogy előbbi transzformációt automatizált módon a tervezett SVP környezet segítségével hajtsuk végre.

A soron következő bekezdésekben bemutatjuk az érzékelőelem példáján keresztül az előállított köztes modellt, valamint a létrehozott Petri-háló és UPPAAL automata modellt. A transzformációs szabályokat itt nem részletezzük, a Petri-hálók esetére részletes összefoglaló található [15]-ben.

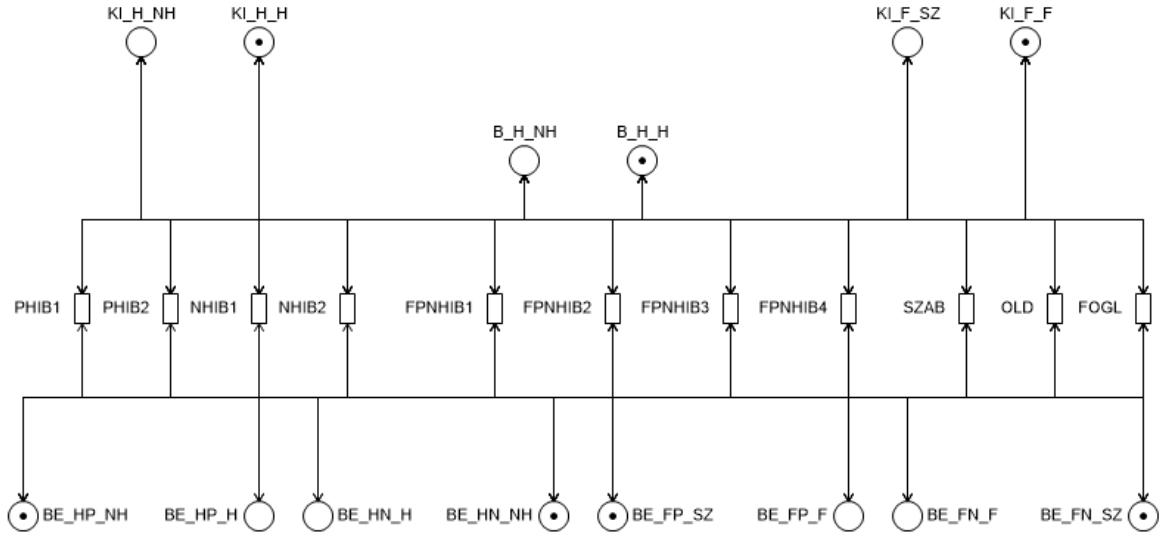
A köztes modell előállítását – melynek alapja az 1. táblázat – az érzékelőelem objektum esetén jelentősen megkönnyíti, hogy a fejlesztők táblázatos formában adták meg annak logikai viselkedését és az 1. táblázat előzetesen egy verifikáción már átesett. A köztes modell elkészítése ezért jelen esetben mindössze a táblázat tartalmának digitális feldolgozáshoz szükséges formátumra való átalakítását jelenti. A 2. táblázatban szereplő 0 és 1 értékek az 1. táblázat segítségével könnyen értelmezhetőek. (Megjegyezzük, hogy a köztes modell előállítása általában nem ilyen egyszerű. A fejlesztők csak néhány egyszerűbb objektum leírását definiálják igazságtáblázat segítségével – ami szintén a formális leírások közé tartozik – a legtöbb esetben viszont a viselkedés leírása állapottérképekkel és az azokat kiegészítő egyéb leírásokkal – pl. idődiagramok – történik.)

Belső állapot	**Bemenetek**				**Kimenetek**		**Új belső állapot**
BHIBA	**HIBA-P**	**HIBA-N**	**FOGL-P**	**FOGL-N**	**HIBA**	**FOGL**	**BHIBA**
-	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
0	0	0	1	0	1	1	1
0	0	0	1	1	0	1	0
-	1	-	-	-	1	1	1
-	-	1	-	-	1	1	1
1	-	-	1	-	1	1	1
1	-	-	-	1	1	1	1

2. táblázat: Az érzékelőelem köztes modellje

A 4. ábrán láthatjuk az érzékelőelem Petri-háló modelljét, melyet a 3. ábra lépésein végighaladva hoztunk létre. A 4. ábra alján szerepelnek az érzékelőelem bemeneteit leképező helyek (BE_ előtaggal), az ábra középső részén az érzékelőelem belső állapotait leképező helyek (B_ előtaggal), az ábra felső részén pedig az érzékelőelem kimeneteit képező helyek láthatóak (KI_ előtaggal). A helyeken feltüntettük a kezdeti token eloszlást. A 4. ábra középső részén találhatóak azok a tranzíciók, melyek az érzékelőelem viselkedését meghatározzák. Ezek két csoportra oszthatóak. Azon tranzíciók, melyek nevében _HIB_ jelölés szerepel, az érzékelőelem valamely állapotából történő hibaállapotba lépését modellezik. A 4. ábrán szereplő tranzíciók száma alapján leolvasható, hogy 8 különféle eset létezik, ahogyan az

érzékelőelem hibaállapotba kerülhet. A tranzíciók másik csoportja az oldást (OLD), az érzékelőelem foglalttá válását (FOGL) és felszabadulását (SZAB) modellezi.



4. ábra: Az érzékelőelem Petri-háló modellje

A 4. ábrán látható érzékelőelem modellhez néhány kiegészítő megjegyzést teszünk:

- A modell összesen 130 élet rejt, melyek egymáson futnak azzal a céllal, hogy átlátható legyen a modell szerkezete, azonban ilyen módon megjelenítve a modellt, az nem értelmezhető Petri-hálóként. A modell olvashatósága azonban nem célunk, mivel a modellek generálása az SVP környezetben automatikusan fog történni, és a gyakorlatban előforduló objektumok mérete (Petri háló esetén különösen) általában olyan nagy, hogy a belőle származó modellek már átláthatatlanok. (Részben ez az oka annak, hogy a Petri-hálók alkalmazása a biztosítóberendezés fejlesztésekben alkalmazási nehézségekkel küzd.)

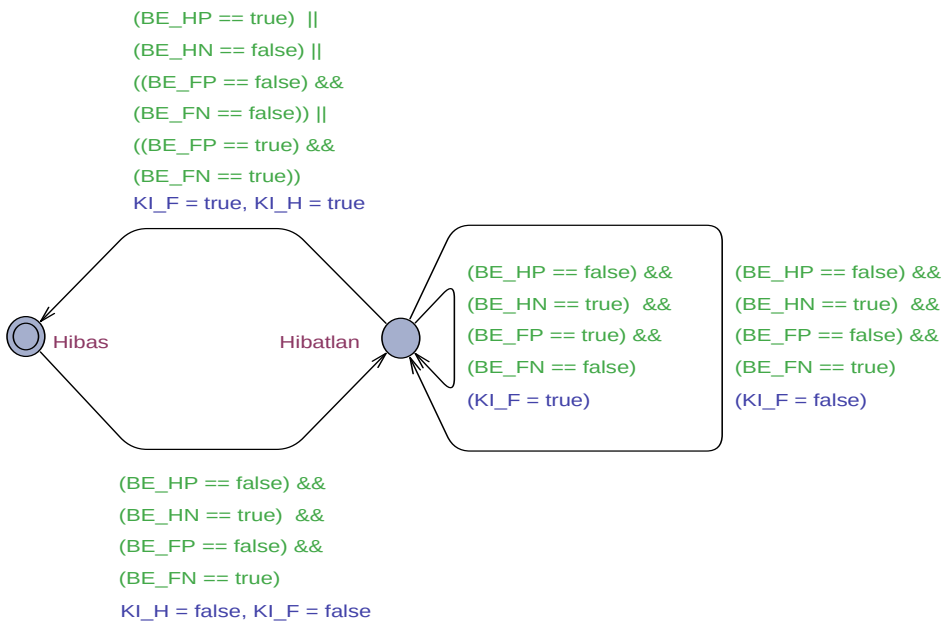
- A modell bemenetei és kimenetei (azaz interfészei) ebben az elrendezésben egyértelműen utalnak arra, hogy az érzékelőelem modellje egy nagyobb rendszerbe integrálható (alhálóként). Ezzel a technikával könnyen képezhetünk hierarchikus Petri-hálókat, melyek segítségével vizsgálhatjuk a fejlesztett rendszer viselkedését integrációs szinten is. Számos tanulmány bizonyítja ugyanis, hogy az egyszerű, könnyen leírható rendszerelemek együttműködése nagyon összetett viselkedést eredményezhet az integrált rendszer szintjén [16].

A 3. ábra transzformációs lépéseit elvégezve az 5. ábrán látható UPPAAL modellhez jutunk, amely az érzékelőelem UPPAAL automata modellje. Az ábra felső részén a be- és kimeneti változók inicializálását láthatjuk, míg az 5. ábra alsó részén az érzékelőelem működését modellező automatát. Az automata két belső állapottal rendelkezik, melyek közül a dupla körrel jelölt állapot a kezdőállapot. Az 5. ábrán szereplő átmenetek két részből állnak: a zöld

színű rész az állapotátlépési őrfeltételeket jelenti, míg kék színű rész a (kimeneti) változókat értékadásait, (utóbbiak az állapotátlépés során kerülnek frissítésre).

```
// Bemenetek állapotát rögzítő változók
bool BE_HP = false; // true: hibás, false: nem hibás
bool BE_HN = true; // true: nem hibás, false: hibás
bool BE_FP = false; // true: foglalt, false: szabad
bool BE_FN = true; // true: szabad, false: foglalt

// Kimenetek állapotát rögzítő változók
bool KI_H = true; // true: hibás, false: nem hibás
bool KI_F = true; // true: foglalt, false: szabad
```



5. ábra: Az érzékelőelem UPPAAL automata modellje

Az 5. ábrán szereplő modell alapján megállapíthatjuk, hogy a Petri-hálónál egy sokkal átláthatóbb formájú modellt kaptunk, az információk sokkal tömörebben ábrázolhatóak. Megjegyezzük, hogy az érzékelőelem Petri-hálós modellje és UPPAAL automata modellje egymásba transzformálható, ami segít a generált modellek összehasonlítás alapú megfelelőségének ellenőrzésében.

5. Összefoglalás

Jelen írásunkban bemutattuk a szakterületi specifikációból kiindulva formális modellek kézi előállításának folyamata során szerzett tapasztalatainkat az érzékelőelem – mint esettanulmány – példáján. Bemutattuk az érzékelőelem Petri-hálós és UPPAAL automata modelljét, melyeket korábban már azonosított transzformációs szabályok segítségével hoztunk létre a PDN és UPPAAL eszközök felhasználásával. Ez az előkészítő munka jelenti az alapját a formális modellek szakterületi specifikációból való automatizált generálásának.

Az automatizált formális modell generálás témakörében eddig elért eredmények kiegészítésre szorulnak az időzítés, valamint a konfigurációs elemek transzformációs szabályaival. Ezen

kiegészítéseket követően lehetőség nyílik az egyedi formális részmodellek egységes modellé való integrációjára, az integrált modell viselkedésének a vizsgálatára.

A CTL-modellellenőrzéshez a formális modelleken túl szükség van a CTL-követelmények megfogalmazására is. Ehhez a formalizálás szempontjából szükséges követelménytípusokat definiáltuk. Azonban szükség van egy szabályrendszer definiálására is, amely a szakterület-specifikus mérnöki leírások nem egyértelmű, informális elemeinek (mind szintaktikai, mind szemantikai) kiküszöbölését, így a gyári követelmények CTL nyelvű megfogalmazhatóságát lehetővé teszi.

A CTL-modellellenőrzés területén a kihívást a modellellenőrzés eredményének a szakterület-specifikus mérnöki leírások fogalomrendszerére való visszavetítése jelenti. Szükség van egy olyan szabályrendszer kidolgozására, melyet felhasználva a szakterületi mérnökök számára a modellellenőrzés eredményét egyértelműen és hiánytalanul lehet közölni (az általunk vizsgált formalizmusok és eszközök körében).

6. Hivatkozások

[1] EN-50128, Railway applications-communication, signaling and processing systems-software for railway control and protection systems, 2011.

[2] Pataricza, A., Bartha, T., Csertán, G., Gyapay, S., Majzik, I., and Varró, D. Formális módszerek az informatikában, (in Hungarian) Typotex, 2004

[3] Ésik Z., Gombás É., Németh L. Z. Verification of hardware and software systems, (in Hungarian), TYPOTEX, 2011.

[4] Kröger, Fred, Merz, Stephan Temporal Logic and State Systems, ISBN 978-3-540-68635-4, Springer, 2008

[5] Clarke, E.M., Henringer, Th.A., Veith, h., Bloem, R. (Eds.) Handbook of Model Checking, ISBN 978-3-319-10575-8, Springer, 2016

[6] Object management group, unified modeling language (OMG UML) V2.5, Object Management Group, 2015.

[7] Bagheri H., Sullivan K. Bottom-up model-driven development, 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18-26 May 2013, pp. 1221–1224.

[8] Sherri L. Jackson Research Methods, A modular approach, ISBN-13: 978-0495811190, ISBN-10: 049581119X , 2011

[9] Farkas B., Lukacs G., Bartha T. Experiences with requirement formalization in the railway interlocking development, In: Innovation and Sustainable Surface Transport, T. Peter (Ed.) Vol. XI, 2017, pp. 197–204.

[10] Henrik Reif Andersen An Introduction to Binary Decision Diagrams, Lecture notes for Efficient Algorithms and Programs, Fall 1999

[11] He X., MurataT. High-level Petri nets extensions, analysis, and applications, In: The Electrical Engineering Handbook, Chen W. K. (Ed.) Academic Press, Burlington, 2005.

[12] Vörös A., Darvas D., Hajdu Á., Klenik A., Marussy K., Molnár V., Bartha T., Majzik I. Industrial applications of the PetriDotNet modeling and analysis tool, Science of Computer Programming, 2017, https://www.sciencedirect.com/science/article/pii/S01676423173
01910, (last visited 08 January 2018).

[13] Gerd Behrmann, Alexandre David, and Kim G. Larsen A Tutorial on Uppaal, Department of Computer Science, Aalborg University, Denmark, http://people.cs.aau.dk/~adavid/publications/21-tutorial.pdf (last visited: 21 February, 2018)

[14] Dániel Darvas Practice-Oriented Formal Methods to Support the Software Development of Industrial Control Systems, Ph.D. Dissertation, 2017, http://mit.bme.hu/~darvas/phd/dissertation.pdf (last visited: 21 February 2018)

[15] Bartha T., Lukacs G. Opportunities of automated transformation of formal specification into a formal model in the interlocking systems area, In: Innovation and Sustainable Surface Transport, T. Peter (Ed.) Vol. XI, 2017, pp. 187–196.

[16] Cai H., Zhang C., Wu W., Ho T. K., Zhang Z. Modeling high integrity transport systems by formal methods, Procedia - Social and Behavioral Sciences, Vol. 138, 2014, pp. 729–737.