# Természetes nyelven megadott vasúti követelmények transzformációja formális tulajdonság leíró nyelvre elágazó idejű temporális logika felhasználásával

**Lukács Gábor*. Dr. Bartha Tamás****

**Budapesti Műszaki és Gazdaságtudományi Egyetem,*
*1111 Stoczek u. 2. (e-mail: lukacs.gabor@edu.bme.hu).*
***Budapesti Műszaki és Gazdaságtudományi Egyetem,*
*1111 Stoczek u. 2. (e-mail: bartha.tamas@mail.bme.hu).*

Absztrakt: A követelmény-specifikációk megfelelő elkészítése a legtöbb szakterületen kihívást jelent – beleértve a közlekedést is. A rendszerekkel kapcsolatos elvárások helyes formalizálása azért döntő fontosságú lépés, mert a fejlesztés és a követelmények elemzése során elkövetett hibák következményei a fejlesztési életciklus későbbi szakaszaiban egyre súlyosabbak. A hibák elkerülésére a szabályozások és szabványok félformális és formális követelményleíró nyelvek használatát javasolják a rendszerfejlesztés során. Egy-egy szakterület szakértőinek azonban nehézséget okozhat a szakterülettől független, formális módszer alkalmazása, részben a szükséges háttérismeretek hiánya, részben a formális leírások nehezen olvashatósága miatt. Érdemes tehát kompromisszumra törekedni és mind az adott területhez illeszkedő, mind a természetes nyelvhez közelebb álló formalizmust kialakítani. Ez a cikk egy lehetséges módszertant (transzformációs folyamatot) mutat be egy ilyen köztes nyelv fejlesztésére és alkalmazására. Célunk elsősorban a vasúti biztosítóberendezési szakértők munkájának támogatása az életciklus során, a rendszerfejlesztés szintjén.

## 1. BEVEZETÉS

A biztonságkritikus közlekedési rendszerfejlesztés jelenlegi gyakorlatában kiemelten fontos a magas szintű (működési) biztonságot is magában foglaló rendszerek kialakítása, ezért erős a motiváció az előírások formalitásának és szigorúságának növelésére, beleértve a követelmény-specifikációkat is. A formális módszerek alkalmazását a szakterületi szabványok is előírják. Például az EN 50129 szabvány (CENELEC, EN 50129, 2018) ezeket a technikákat SIL3 és SIL4 biztonsági integritási szinten (*Safety Integrity Level, SIL*) „erősen ajánlott” (*highly recommended, HR*) kategóriába sorolja. A fejlesztés során az igényelt, valamint szükséges/elvárt tulajdonságok és viselkedés meghatározása a követelményspecifikációban a szakterületi mérnökök feladata. Rendszertervezési szinten azonban nem szívesen alkalmaznak formális módszereket, mert azok absztrakt/homályos természetük és számításelméleti hátterük miatt jelentős plusz ismereteket igényelhetnek. Természetesen az életciklus szoftvertervezési szakaszában is találkozhatunk a formális módszerek gyakorlati alkalmazásaival, mivel a szoftver-mérnökök jellemzően rendelkeznek megfelelő szintű ismeretekkel ezekről a technikákról (pl. Schneider, 2001). Azonban a közlekedési (esetünkben a vasúti) mérnökök ezeket a készségeket általában csak több éves kemény munka után sajátíthatják el.

A fejlesztés közbeni információáramlás középpontjában az adott mérnöki szakterület (domén) mérnökei állnak. Segítik az érintett felek (*stakeholders*) által elképzelt különböző igények fejlesztési bemenetekké (azaz tervezési alapokká) való átalakítását, valamint javaslatokat tesznek a lehetséges műszaki megoldási alternatívákra, emellett pedig a fejlesztők által azonosított akadályokat/korlátokat közvetítik az érintett felek számára. Ezek a tevékenységek sok esetben különösen nehezek lehetnek, sok konfliktushoz, és – nem utolsó sorban – sok iterációhoz vezethetnek a fejlesztés során, ami annak tudható be, hogy a gyakorlatban a felhasználói igények nagyon ritkán felelnek meg (hiába a sokéves tapasztalat) a „jó” követelményspecifikáció kritériumainak. A jó specifikációt (többek között) a helyesség, teljesség, konzisztencia és ellenőrizhetőség tulajdonságokkal jellemezhetjük. Sajnos a végfelhasználók általában nem rendelkeznek megfelelő erőforrással a magas minőségű követelményspecifikációk elkészítéséhez. A beszállítók szakterületi mérnökeinek tehát le kell küzdeniük ezeket a nehézségeket, amelyek egyszerű, gyors és hatékony megoldásokat igényelnek.

Kutatási célunk egy könnyen elsajátítható, széles körben alkalmazható és gyakorlatorientált módszertan megalkotása, amely támogatja a szakterületi mérnököket abban, hogy formális módszereket alkalmazzanak magas minőségi színvonalú funkcionális specifikációk készítésére. Az elkészített specifikáció egy olyan platformot képezhet, amely lehetővé teszi a követelmények egyeztetését mind a végfelhasználókkal, mind a megvalósítást végző mérnökökkel egyaránt. A módszertan további előnye, hogy minimalizálja a szükséges számítástudományi háttérismereteket, és a vasút-mérnökök által jól ismert és széles körben használt technológiákra épít. Ebben a cikkben ennek a platformnak az egyik legfontosabb részét ismertetjük: azt, hogy a (funkcionális) követelmények hogyan fejezhetők ki egy

korlátozott, szakterület-specifikus természetes nyelven (*Restricted Textual Template, RTT*), majd hogyan alakíthatók át formális követelményekké. A temporális logika (Fisher, 2011) egy jól bevált matematikai formalizmus erre a célra, ezért úgy döntöttünk, hogy elágazó idejű számítási fa logikát (*computational tree logic, CTL*, Chatterjee *et al.*, 2016) használunk fel a követelmény-formalizálás alapjául. Azért választottuk a CTL-t, mert jól illeszkedik (az ismert korlátai ellenére) a vasúti szakterület funkcionális követelményeinek leírásához, és számos eszköz (pl. UPPAAL, Behrmann *et al.*, 2001) támogatja a használatát ezen a szakterületen.

Ebben a cikkben először áttekintést adunk a javasolt transzformációs folyamat hátteréről és kapcsolódó tudományos eredményekről (2. fejezet). A 3. fejezetben bemutatjuk az általunk javasolt transzformációs folyamatot, amely támogatja a vasúti szakterület mérnökeit abban, hogy CTL-alapú leírásokat alkalmazzanak magas minőségű, formális, funkcionális követelményspecifikációk elkészítéséhez. A cikk végén a transzformációs folyamatot tágabb kontextusba helyezzük (4. fejezet) és illusztráljuk egy esettanulmány segítségével, és végül az 5. fejezetben levonjuk a következtetéseket.

## 2. TUDOMÁNYOS HÁTTÉR

A javasolt transzformációs folyamat célja, hogy támogassa a vasúti terület mérnökeit a vasúti vezérlő- és biztosítóberendezések magas minőségű, világos, tiszta és pontos funkcionális követelmény-specifikációinak kidolgozásában. Jelen fejezet célja, hogy áttekintést nyújtson a közlekedési terület (beleértve a szóban forgó szakterület) vonatkozó gyakorlatáról és tudományos hátteréről a 3. fejezetben leírt transzformáció szempontjából. Az ebben a részben található háttérismeretek segítenek megérteni a cikk későbbi részét, és a transzformációs folyamat bemutatásakor hivatkozni fogunk rájuk.

### *2.1 Követelmények kezelése a rendszerfejlesztés során*

A gyakorlatban a követelményekkel kapcsolatos tevékenységeket a *Requirements Engineering (RE)* és *Requirements Management (RM)* részeként végzik a fejlesztési életciklus során. Az RE olyan tevékenységek összessége, amelyek célja a fejlesztendő rendszer céljainak, képességeinek, korlátainak és feltételezéseinek feltárása, értékelése és dokumentálása (Lamsweerde, 2009). Az RM (Hood *et al.*, 2007) – mint az életciklust átfogó tevékenységek halmaza – olyan tevékenységeket foglal magában, mint a követelmények nyomon követése, priorizálása, ellenőrzése, karbantartása, stb. Manapság már számos jól definiált RE és RM módszer, technika és eszköz áll rendelkezésre, amelyeket a releváns szabványok (pl. ISO/IEC/IEEE 15288, 2015) is ajánlanak.

A 3. fejezetben javasolt transzformációs folyamat kidolgozásakor az RE és RM mellett a vasúti követelményforrások vizsgálata is kiemelt szerepet kapott. A vasúti vezérlő- és biztosítóberendezések fejlesztésére vonatkozó követelmények számos forrásból származhatnak, és ezeket a követelményeket általában az absztrakció és a formalizáltság különböző szintjein adják meg a különböző érintettek felek. Ezek a fejlesztéshez szükséges bemenetek az életciklus korai szakaszaiban jelennek meg, és a fejlesztőcsapat által írt különböző típusú követelményspecifikációk alapját képezik. A gyakorlatban gyakran számos követelményforrás is azonosítható a kialakítandó rendszerrel kapcsolatban, ami akár több ezer követelményhez vezethet egy összetett rendszer esetében. Az azonosított követelményrendszer tisztázása nehézkes az elemek nagy száma, a köztük lévő összefüggések, az eltérő absztrakciós és formalizáltsági szintek miatt. Ez a „kiinduló” követelményrendszer ezek mellett mindenféle problémával terhelt, amelyek többsége a természetes nyelvű megfogalmazásból ered. Ilyenek pl. a rugalmasság (ugyanazt többféleképpen is meg lehet fogalmazni), többértelműség, félreérthetőség, terjengősség, követelmények keveredése, vagy az ellenőrizhetőség szempontjainak figyelmen kívül hagyása, stb.

### *2.2 Modellalapú rendszerfejlesztés*

A jelenlegi mérnöki gyakorlatban a *Model-based System Engineering (MBSE)* (SEBoK, 2022) és a *Model-driven Development (MDD)* (Parviainen *et al.*, 2009) megközelítések rendkívül népszerűek. *A modell egy rendszer leegyszerűsített ábrázolása az idő vagy a tér egy bizonyos pontján, aminek a célja a valós rendszer megértésének elősegítése* (Bellinger 2004, SEBoK, 2022). Egy modellnek sokféle megjelenési formája lehet, pl. szöveges, formális, grafikus vagy akár vegyes. A vasúti mérnöki gyakorlatban a formális reprezentáció kivételével mindegyik ábrázolási forma széles körben alkalmazott.

Az MBSE a modellezés egy formalizált alkalmazása, amely támogatja az életciklus tevékenységeket (például a követelményspecifikáció elkészítését) a fejlesztés során (Papke, 2020), amelynek célja, hogy egyesítse a különböző mérnöki tudományágak modellközpontú megközelítéseit (pl. közlekedési, villamosmérnöki, informatikai stb.). Az MDD a szoftverfejlesztés olyan megközelítése, amelyben a magas szintű modellek állnak a középpontban (Selic, 2006). Mindkét megközelítésben a modellek központi szerepet játszanak. Számos eszköz (pl. MATLAB, Hunt *et al.*, 2001) támogatja az MBSE-t és/vagy az MDD-t. A mérnökök általában ismerik a saját területükön elérhető modellezési, szimulációs vagy elemzési eszközöket, így, ha problémával találkoznak, azonnal megvizsgálhatják azt a modellezés segítségével.

### *2.3 Formális módszerek, temporális logikák*

A formális módszerek olyan technikák, amelyeket kezdetben elsősorban az információs technológia területén használtak. Ezeket a technikákat ma már számos más területen is alkalmazzák, mint például a szoftverfejlesztés, a programozható logikai áramkörök, a protokollok, a kommunikáció, az áramkörtervezés, stb. Ezek a technikák a matematikán, főként a diszkrét matematikán és a matematikai logikán alapulnak. A formális módszerek jól használhatók a rendszerfejlesztésben, beleértve a szoftver- és hardver-

fejlesztést (Woodcock *et al.*, 2009). A formális modellek szemantikája és szintaxisa jól meghatározott, világos és teljes, így jelentősen csökkenti a kétértelműséget vagy a specifikációk egyéb hiányosságait.

A jelenlegi vasúti mérnöki gyakorlatot a nem formális és félformális (szöveges és ad hoc jelölési rendszerekből álló) specifikációk jellemzik, amelyeket a szakterületi mérnökök készítenek. A fejlesztési folyamat résztvevői közötti együttműködést a félreértések és/vagy a kétértelműség jellemzik. A formális modellek használata a fejlesztési folyamat során csökkentheti ezeket a nehézségeket. A formális módszerek támogatják a szigorú specifikációt, tervezést és ellenőrzést, komplex rendszerek modellezését és a hibák azonosítását az életciklus korai fázisaiban. A végrehajtható modellek már a gyártás korai szakaszában ellenőrizhetők.

A temporális logikák (Demri *et al.*, 2016) a klasszikus logika időre és időbeli tulajdonságokra való kiterjesztésével jöttek létre. Lineáris temporális logika esetén minden időpillanatnak csak egy következő időpillanata lehet (más szóval csak egyfajta jövő lehetséges), míg az elágazó temporális logika esetében minden időpillanatnak akár több egymást követő időpillanata is lehet (azaz több alternatív jövő is lehetséges). Ebben a cikkben az elágazó temporális logikák közül a **CTL**-t fogjuk használni, amely a CTL*-ból származtatható. A **CTL*** temporális logikát a következő specifikációval adhatjuk meg:

- atomi kijelentések: p, q, r, s stb.
- logikai operátorok: ∧ (és), ∨ (vagy), ¬ (nem) stb.
- temporális operátorok:
  1) **F***p*: a *p* kijelentés egy bizonyos jövőbeli állapotban igaz lesz,
  2) **G***p*: a *p* kijelentés minden jövőbeli állapotban igaz lesz,
  3) **X***p*: a *p* kijelentés igaz lesz a következő jövőbeli állapotban
  4) *p***U***q*: egy jövőbeli állapotban igaz lesz *q* kijelentés és eddig az állapotig *p* kijelentés is igaz lesz.
- útvonal kvantorok:
  1) **A***α*: az *α* tulajdonság minden lehetséges útvonalra teljesül,
  2) **E***α*: az *α* tulajdonság legalább egy útvonalra teljesül a kiinduló állapotból induló utak közül.

A CTL a CTL* egy korlátozott változata (részhalmaza). Különbségüket a következő megszorítások jellemzik a CTL-re nézve:

- Egy útvonal kvantort mindig egy temporális operátornak kell követnie (pl. **AG EF**(*p* ∨ *q*)).
- A temporális operátorok nem kombinálhatók szabadon (ezért pl. az **EF X***p* érvénytelen, míg az **AG E**(*p***U**¬*q*) érvényes/helyes kifejezés CTL-ben).

A CTL használata modellellenőrzésre (lásd részletesen a 4. fejezetben) széles körben elterjedt a vasúti szakterületen, lásd pl. (Rakesh and Kadakolmath, 2018) vagy (Mirabadi and Yazdi, 2009).

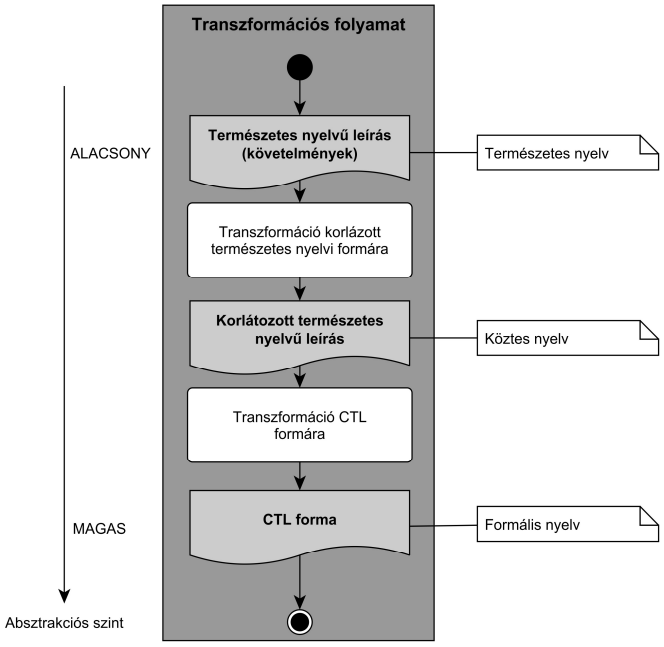
## 3. A TRANSZFORMÁCIÓS FOLYAMAT

Ebben a fejezetben bemutatjuk a vasúti rendszerek szakterületi követelményeinek formális nyelvre – pontosabban CTL nyelvre – való átalakítására javasolt módszertant. A továbbiakban erre a módszertanra egyszerűen "transzformációs folyamat"-ként hivatkozunk.

Az 1. ábra a transzformációs folyamat lépéseit foglalja össze. A folyamat egy bemenetből (*Természetes nyelvű leírás*), egy köztes leírásból (*Korlátozott természetes nyelvű leírás*) és egy kimenetből (*CTL forma*) áll. Ezeken kívül a folyamat két fő tevékenységet foglal magában:

1) *Transzformáció korlátozott természetes nyelvi formára*: ami egy meghatározott szabályrendszerre épített átalakítási lépéssort jelent a természetes nyelvi formáról egy korlátozott természetes nyelvű formára,
2) *Transzformáció CTL formára*: ami egy definiált szabályrendszer melletti átalakítási lépéssort jelent a korlátozott természetes nyelvi formáról CTL formára.

A transzformációs folyamat áttekintéséhez megjegyezzük, hogy a folyamat elejétől a végéig az absztrakció szintje növekszik (azaz egyre kevésbé érthető lesz a követelmény). Ezt az 1. ábrán egy nyíllal is szemléltettük.




1. ábra. Transzformációs folyamat.

A következő bekezdésekben részletesen bemutatjuk a javasolt transzformációs folyamatot, az alkalmazott előfeltételekkel és korlátokkal együtt.

Feltételezzük, hogy a transzformációs folyamat bemenetét természetes nyelven leírt követelmények képezik, és ha ez nem így van, akkor a más formában megadott követelmények természetes nyelvi leírássá alakíthatók. Megjegyezzük, hogy a gyakorlatban a mérnökök – szakterülettől függetlenül – egyre gyakrabban használnak más leírási formákat a követelmények meghatározásához (pl. grafikus formát, lásd 2. fejezet).

A követelményeknek két fő típusa van (Lamsweerde, 2009): *funkcionális* és *nem funkcionális* követelmények. Sokféle nem funkcionális követelmény létezik, de ezek nem tartoznak a jelen írásunk hatókörébe. Így az átalakítás során csak a funkcionális követelményekkel foglalkozunk. Ezek rendkívül fontosak a vasúti biztosítóberendezések területén, amely a funkcionális biztonság elvein alapul (CENELEC, EN 50129, 2018). Ez azonban nem jelenti azt, hogy a folyamatot a jövőben ne lehetne kiterjeszteni a nem funkcionális követelményekre is (természetesen az integrációhoz szükséges feltételek betartása mellett).

A transzformációs folyamat definiálásakor a megközelítésünk az volt, hogy a természetes nyelvi elemek egy korlátozott halmazából álló szöveges követelménymintákat egyfajta köztes specifikációs formaként használjuk. Azt tapasztaltuk, hogy a természetes nyelvi követelményekből könnyebb elérni ezt a köztes formát, és a későbbiekben sokkal könnyebb lesz automatizálni a folyamatot (lásd 5. fejezet). A funkcionális követelmények leírására sok népszerű követelménymintát találhatunk a gyakorlatban (Ugur *et al.*, 2022). Ezek a szakterületi mérnökök számára is jól ismertek. Ilyen egyszerű minta például az „Adott-ha-akkor” (*Given-When-Then, GWT*) (Solis and Wang, 2011). A következőkben egy egyszerű forgatókönyvet mutatunk be a GWT minta alkalmazásával, amit nagyon gyakran használnak a vasúti biztosító-berendezések területén is különféle biztonsági szolgáltatásokhoz (pl. váltóállítási parancs kiadása, a beágyazott biztonsági szoftver konfigurációjának módosítása stb.):

> ***Adott***, hogy van egy felhasználó, aki regisztrálva van, ki van jelentkezve, és a bejelentkezési felületen van.
> ***Ha*** a felhasználó az „Azonosító”, „Jelszó” és a „Szervezet” mezőbe beírja a helyes hitelesítési adatait, és megnyomja a „Belépés” gombot,
> ***akkor*** a rendszer bejelentkezteti a felhasználót (aki hitelesítve van).

Ezen a példán keresztül láthatjuk, hogy ez egy teljesen érthető, természetes nyelvű leírás. Emellett különféle törvényszerűségeket is megfigyelhetünk, melyekre a példa megfelelő felosztásával igyekeztünk rávilágítani.

A korlátozott természetes nyelvű minták lényege, hogy véges számú természetes nyelvi elemet használnak, melynek célja a követelmények egységes megfogalmazása a nyelv kifejezőképességének/érthetőségének lehető legjobb megőrzése mellett (Giannakopoulou *et al.*, 2020).

A következőkben egy olyan szabályrendszert adunk meg, amellyel a természetes nyelven megadott funkcionális követelményeket korlátozott természetes nyelvű leírásokká alakíthatjuk, figyelembe véve a szakterületi mérnökök által is jól ismert követelmény mintákat (pl. GWT).

A funkcionális követelmények alapformája (mintája) a következő:

„*Ha* (*kifejezés1*) *akkor* (*kifejezés2*) *egyébként* (*kifejezés3*)”,

ahol

- a *kifejezés1* egy előfeltétel, amely előírja, hogy a funkció végrehajtása előtt minek kell igaznak lennie (az „*akkor*” részhez) vagy hamisnak lennie (az „*egyébként*” részhez),
- a *kifejezés2* és *kifejezés3* olyan utófeltételek, amelyek meghatározzák, hogy mi lesz igaz a funkció végrehajtása után.

Ezt követően megadjuk azokat a szabályokat, amelyeket az 1, 2 és 3 kifejezés írásakor teljesíteni kell. Ezek a szabályok az elő- és utófeltételekre megegyeznek:

- **általános nyelvtani szabályok:**
  - A következetes szóhasználat/megfogalmazás kötelező.
  - Csak kijelentő mondatok/kifejezések használhatók.
  - Igék nem használhatók (pl. ad, kap, fogad stb.).
  - A névszavak (főnevek, melléknevek, számnevek és névmások) közül csak a főnevek használhatók (azaz nem használhatók pl. a kicsi, három, sok, én, enyém stb. szavak), ill. a gyűjtőnevek sem használhatók (pl. nép, csapat stb.).
  - Határozószavak nem használhatók (pl. egyszer, azonnal, valahol, itt, ott, alig, közben stb.).
  - Igenevek nem használhatók (pl. jönni, járó, elküldött, elvégzendő, futtatva stb.).
  - Viszonyszavak (névelők, névutók, igekötők, segédigék, kötőszavak, módosítószavak) nem használhatók (pl. az, alá, ki-, fog, volna, mert, talán stb.).
  - Mondatszavak nem használhatók (pl. jaj, hess, puff, persze, dehogy stb.).
- **atomi kijelentések:**
  - Az atomi kijelentések szerkezete a következő felépítésű kell legyen:

    *„(objektum) reláció szimbólum (érték)”,*

    ahol a reláció szimbólum lehet: ==, <, >, ≤, ≥ stb. Megjegyzés: az *„objektum”* és az *„érték”* általában a természetes nyelvi leírásból (követelmény szövegéből) származtathatók.
  - Az atomi kijelentések „értékének” megadása azok típusától (logikai érték, egész szám stb.) függetlenül

mindig kötelező. Ezek az értékeket akár szimbolikusan is meg lehet jeleníteni.

- A névterek (OMG UML, 2011) használata megengedett, de nem kötelező. Ily módon meg lehet határozni, hogy egy adott „objektum” milyen állapotban van, illetve azt, hogy milyen értéket vagy tartományt rendelünk egy paraméterhez. Például az „egy felhasználó, aki regisztrálva van” előfeltételt a következőképpen írhatjuk fel:
  `Felhasználó.regisztráció == igaz.`

- **logikai operátorok:**
  - A kifejezések az atomi kijelentésekből logikai operátorok segítségével képezhetők (a logikai kifejezések összetettségére vonatkozóan nem támasztunk külön követelményt).
  - A logikai operátorok szokásos formája használható: ! (nem), && (és), || (vagy) stb.

Ezen a ponton a fenti nyelvtani szabályok segítségével átalakíthatjuk a példában szereplő funkcionális követelményt korlátozott természetes nyelvű formára. Megjegyezzük, hogy hogy az „egyébként” ágat automatikusan transzformálhatjuk "ha-akkor" jellegű kifejezéssé az "egyébként" feltételben szereplő kifejezés tagadásával és a De Morgan azonosságok felhasználásával. Ezzel az automatikus transzformációval támogatni kívánjuk a szakterületi mérnökök munkáját azáltal, hogy a széles körben elterjedt „Ha-akkor-egyébként” (*If-then-else*) formát használhatják igényeik rögzítésére.

Fentebb megadott forgatókönyv ezeket a nyelvtani szabályokat betartva a következő formába írható át:

```
Ha (Felhasználó.regisztráció == igaz &&
Felhasználó.bejelentkezés == hamis &&
Felhasználó.bejelentkezőFelület == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Azonosító) == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Jelszó) == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Szervezet) == igaz &&
bejelentkezés.Gombnyomás == igaz)
akkor Felhasználó.bejelentkezés == igaz.
```

Azonban még mindig hiányzik egy fontos lépés ahhoz, hogy fenti példát helyesen CTL formára alakítsuk. Ehhez az szükséges, hogy megadjuk a követelmény időbeli vonatkozásait a 2.3. fejezetben definiált időbeli operátorokkal és útvonal kvantorokkal egyenértékű szöveges címkékkel. Ezt a lépést a szakterületi mérnököknek kell elvégezniük, mivel ők rendelkeznek az ehhez szükséges információkkal. Ezt követően az „Ha-akkor” struktúrát felváltja a logikai implikáció. Összességében a „*Ha* (*kifejezés1*) *akkor* (*kifejezés2*) *egyébként* (*kifejezés3*)” minta a következőre módosul:

*Temporális operátor1 (kifejezés1) implikálja, hogy Temporális operátor2 (kifejezés2)*, illetve

*Temporális operátor1 (!kifejezés1) implikálja, hogy Temporális operátor3 (kifejezés3).*

Megjegyezzük, hogy a CTL formához nem mindig szükséges a „Temporális operátor2 és 3” megadása, és nem minden esetben létezik az „egyébként”, azaz az „else” ág sem.

A temporális operátorok és útvonal-kvantorok az 1. táblázatban leírt értelmezésükkel használhatók fenti minták esetében.

**1. táblázat. A temporális tulajdonságok szöveges címkéi**

| Címke | Jelentés | CTL forma |
|---|---|---|
| Mindig (invariáns) | a *p* tulajdonság mindegyik lehetséges alternatív jövő (útvonal) mindegyik állapotában teljesül | **AG***p* |
| Előbb-utóbb | a *p* tulajdonság mindegyik lehetséges alternatív jövő (útvonal) valamelyik állapotában teljesül | **AF***p* |
| Minden következő állapotban | a *p* tulajdonság mindegyik alternatív jövő (útvonal) soron következő állapotban teljesül | **AX***p* |
| Addig-amíg | mindegyik alternatív jövőben (útvonalon) a *p* tulajdonság minden állapotban teljesül, mielőtt a *q* tulajdonság teljesülne valamelyik időpillanatban (állapotban) | **A**(*p***U***q*) |
| Egy úton mindig | létezik egy olyan alternatív jövő (útvonal), amelynek minden állapotában a *p* tulajdonság teljesül | **EG***p* |
| Legalább egyszer | létezik egy olyan alternatív jövő (útvonal), amelynek valamelyik állapotában a *p* tulajdonság teljesül | **EF***p* |
| Valamelyik következő állapotban | létezik egy olyan alternatív jövő (útvonal), amelynek valamelyik soron következő állapotában a *p* tulajdonság teljesül | **EX***p* |
| Egy úton addig-amíg | létezik egy olyan alternatív jövő (útvonal), amikor a *p* tulajdonság valamelyik állapotban teljesül, mielőtt a *q* tulajdonság teljesülne valamelyik állapotban | **E**(*p***U***q*) |
| Következménye | *p* tulajdonság teljesülése előbb-utóbb maga után vonja *q* tulajdonság teljesülését | $p \rightsquigarrow q$ |

Amint látjuk, a „Következménye” címke esetében nincs implikációs rész, de ugyanabba az implikációs formába hozható ezzel az azonossággal: $p \rightsquigarrow q \equiv \mathbf{AG}(p \Rightarrow \mathbf{AF}q)$

Az itt leírt lépéssel egy úgynevezett köztes nyelv jön létre (lásd 1. ábra). Ezeket az időre vonatkozó jelzőket használva a példakövetelmény korlátozott természetes nyelvű formája így néz ki:

```
Mindig (Felhasználó.regisztráció == igaz &&
Felhasználó.bejelentkezés == hamis &&
Felhasználó.bejelentkezőFelület == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Azonosító) == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Jelszó) == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Szervezet) == igaz &&
bejelentkezés.Gombnyomás == igaz) implikálja, hogy
következménye Felhasználó.bejelentkezés == igaz.
```

A javasolt transzformációs folyamat utolsó lépése a definiált köztes nyelvi forma átalakítása a modellellenőrző által érthető CTL formává. A példa szempontjából a CTL kifejezés a következő:

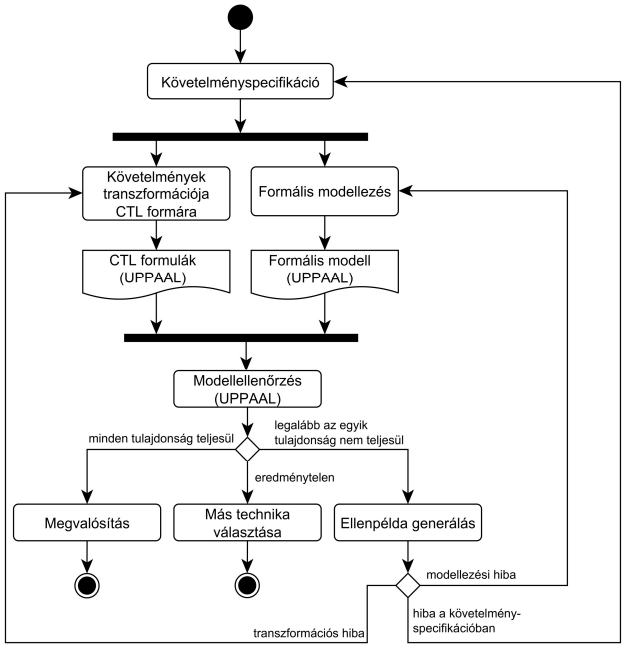
```
AG (Felhasználó.regisztráció == igaz &&
Felhasználó.bejelentkezés == hamis &&
Felhasználó.bejelentkezőFelület == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Azonosító) == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Jelszó) == igaz &&
bejelentkezésEllenőrzés(Felhasználó.Szervezet) == igaz &&
bejelentkezés.Gombnyomás == igaz) imply
AX Felhasználó.bejelentkezés == igaz.
```

Összefoglalva, ebben a fejezetben leírtuk az alkalmazott elveket a javasolt transzformációs folyamathoz és egy példán keresztül lépésről lépésre ismertettük a folyamat lépéseit.

## 4. EREDMÉNYEK ÉS ALKALMAZÁS

A 3. fejezetben bemutatott transzformációs folyamat egyik lehetséges gyakorlati alkalmazása a modellellenőrzés segítségével történő tervezésellenőrzés. A modellellenőrzés (Baier és Katoen, 2008) egy formális módszer, amely arra a kérdésre keresi a választ, hogy egy bizonyos formális modell megfelel-e a rá vonatkozó formalizált követelménynek, vagy egy nem teljesülő követelmény esetén hogyan fordulhat elő ez a helyzet? A modellellenőrzés folyamata a 2. ábra látható.

A modellellenőrzés lehetséges kimeneteit a 2. ábra foglalja össze (lásd a döntési pontot középen). Ha egy formalizált tulajdonság igaznak adódik, akkor a modellezett követelmény (az ellenőrzött tulajdonság) teljesül, és a modell bizonyítottan helyesnek minősül erre a tulajdonságra nézve. Ha van egy eseménysorozat, amely a modell hibás viselkedéséhez vezet, akkor a modellellenőrzés a vizsgált tulajdonság megsértését, sőt ezen felül még egy ellenpéldát is ad eredményül. A „eredménytelen” kifejezés azt jelenti, hogy vagy meghiúsult a modellellenőrzés a rendelkezésre álló memória végessége miatt (állapottér robbanás), vagy az ellenőrzés idő előtt leállt (időtúllépés), mert az túl sok időt venne igénybe.




2. ábra. Modellellenőrzés.

A modellellenőrzés közvetlen bemenetei (pl. a UPPAAL eszköz esetén, Kunnappilly *et al.*, 2021) a CTL formulák és formális modellek, amelyek célja a működőképesség ellenőrzése a tervezett komponens megvalósítása előtt.

A modellellenőrzés bemeneteként szolgáló CTL formulákat pl. egy szabályalapú technikával (lásd jelen cikk tárgya) kaphatjuk meg, korlátozott természetes nyelvű leírások segítségével, míg a komponensek viselkedését (időzített automaták) pl. állapotgépekből generálhatjuk (ami akár automatizált módon is lehetséges). Ebben a cikkben bemutattunk egy lehetséges módot, hogyan juthatunk el a természetes nyelven megadott funkcionális követelmények leírásoktól a CTL formáig.

A következő bekezdésekben egy egyszerű esettanulmány (foglaltság érzékelő elem) példáján mutatjuk be a jelen cikkben leírt eredmények alkalmazását, azaz a modell-ellenőrzést. Az érzékelőelem célja a hatókörében tartózkodó vasúti jármű érzékelése. Az érzékelőelem működését rendszerszinten nagyon egyszerű módon írhatjuk le: ha a vonat az érzékelőelem hatókörében van, akkor az „foglalt”; és amikor a vonat nincs az érzékelőelem hatókörében, akkor az „szabad”. Ezt a viselkedést a vasúti mérnökök a funkcionális specifikáció során az eddigi tapasztalatok és szakterületi ismeretek alapján jelentősen kiegészíthetik (pl. interfészek, időzítések, paraméterek stb.). Az érzékelőelem formális modellje (ami a modellellenőrzéshez szükséges) megtalálható egy másik írásunkban (Lukács és Bartha, 2021), így a modellellenőrzésnek ezt a bemenetét adottnak tekintjük.

Az érzékelőelem modellellenőrzése két lépésből áll:

1) a formális modell validációja,
2) a funkcionális követelmények teljesülésének verifikációja.

Az érzékelőelem komponens modelljének validációja során a 2. táblázatban szereplő tulajdonságokat ellenőriztük. A tulajdonságokat angol nyelven adtuk meg az UPPAAL keretrendszerben való egyszerűbb munkavégzése miatt. Megjegyezzük, hogy a 3. fejezetben leírt transzformációs szabályok angol nyelvre való kidolgozása is folyamatban van, mert a legtöbb modellellenőrző eszköz az angol nyelvet támogatja.

A 2. táblázatban megadott, a modellvalidáció során ellenőrzött követelményekre szintén alkalmazható a 3. fejezetben leírt transzformációs folyamat, azonban jelen cikkben csak a funkcionális követelmények transzformációjának bemutatását tűztük ki célul, így ezeknek a követelményeknek a köztes nyelven történő leírását a 2. táblázatban külön nem szerepeltetjük.

Megjegyezzük továbbá, hogy 2. táblázatban nem szereplő, további követelmények is a modellvalidáció részét képezték. Ezeket a követelményeket 90 különböző konfigurációban ellenőriztük, ugyanis az elérhetőséggel kapcsolatos követelmények konfigurációfüggők, míg a funkcionális követelmények konfigurációtól függetlenek. Az ellenőrzendő konfigurációk kiválasztása a teljes konfigurációs térből egy

nagyon lényeges témakör, aminek a részleteivel jelen írás keretében külön nem foglalkozunk.

**2. táblázat. Modellvalidáció – érzékelőelem.**

| **Tulajdonság** | **CTL forma (az UPPAAL jelölés rendszerében)** |
|---|---|
| Holtpontmentesség. | A□ not deadlock |
| A „PRESENCEHANDLING” automata állapotainak elérhetősége: "Van legalább egy útvonal, amelyen egy adott állapot elérhető." | E◇ presencehandling.free |
| | E◇ presencehandling.occ_wfault_ short_occupancy |
| | E◇ presencehandling.occ_wofault_ non_overflowed_PTomax |
| | E◇ presencehandling.occ_wfault_ overflowed_PTomax |
| | E◇ presencehandling.occ_wfault_ overflowed_CInt8Max |
| | E◇ presencehandling.occ_wofault_ non_overflowed_CInt8Max |
| | E◇ presencehandling.occ_wofault_ overflowed_CInt8Max |

Az érzékelőelem modelljének validációját követően a funkcionális követelmények teljesülését ellenőriztük a modellen (lásd 3. táblázat). A modellellenőrzés eredményét szintén a 3. táblázatban adtuk meg.

**3. táblázat. Modellverifikáció – érzékelőelem.**

| **1. funkcionális követelmény** | |
|---|---|
| **Természetes nyelv** | Ha az érzékelőelem vasúti jármű jelenlétet észlel valamelyik foglaltsági bemenetén, akkor az a foglaltsági kimenet foglalt állapotához vezet. |
| **Köztes nyelv** | (in_presence_p == true \|\| in_presence_n == true) *leads to* out_occupancy == true |
| **CTL forma** | (in_presence_p == true \|\| in_presence_n == true) ⇝ out_occupancy == true |
| **UPPAAL forma** | (in_presence_p == true \|\| in_presence_n == true) –> out_occupancy == true |
| **Modellellenőrzés** | **Tulajdonság teljesült.** |
| **2. funkcionális követelmény** | |
| **Természetes nyelv** | Ha az érzékelőelem hibát észlel valamelyik hibabemenetén, akkor az a foglaltsági kimenetnek a foglalt, a hibakimenetnek pedig hibás állapotához vezet. |
| **Köztes nyelv** | (in_fault_p \|\| in_fault_n) *leads to* out_failure == true && out_occupancy == true |
| **CTL forma** | (in_fault_p \|\| in_fault_n) ⇝ out_failure == true && out_occupancy == true |
| **UPPAAL forma** | (in_fault_p \|\| in_fault_n) –> out_failure == true && out_occupancy == true |
| **Modellellenőrzés** | **Tulajdonság teljesült.** |

Ezzel az esettanulmánnyal bemutattuk a jelen cikkben leírt transzformációs folyamat alkalmazását a modellellenőrzéshez egy speciális szakterületen. Megjegyezzük azonban, hogy a modellellenőrzés során egy – egyébként más szakterületeken is már jól ismert – problémát azonosítottunk, nevezetesen azt, hogy a vasúti mérnökök nehézségekbe ütköznek, amikor valamelyik követelmény nem teljesülését kapják eredményül (lásd 2. ábra). Az érzékelőelem példáját tekintve, amikor az UPPAAL ellenpéldát adott, szinte lehetetlennek bizonyult számukra megfejteni, hol van a „követelménysértést” okozó hiba. Ezt a problémát úgy lehetne megoldani, hogy egy visszafelé irányú leképező/annotáló módszert dolgozunk ki az ellenpélda bemutatására a magas szintű modellben.

## 5. ÖSSZEFOGLALÁS

Ebben a cikkben egy olyan transzformációs folyamatot javasoltunk, amely támogatja a formalizált funkcionális követelmények temporális logikai (CTL) formában történő felépítését a vasúti mérnökök által készített természetes nyelvi leírásokból. Kialakítottunk egy köztes nyelvet (korlátozott természetes nyelvet) a gyakorlatban széles körben használt követelmény mintára („Adott-ha-akkor”) alapozva. A javasolt transzformációs folyamat egy automatizált ellenőrzési keretrendszer része lesz (ez nem szerepel ebben az írásban, lásd pl. Lukács és Bartha, 2019). A javasolt transzformációs folyamat az esettanulmányainkban már hasznosnak bizonyult, de a nyelv finomítására és kiterjesztésére irányuló további kutatások még folyamatban vannak. Eddigi tapasztalataink alapján a javasolt transzformációs folyamat a szabályok egy megfelelően meghatározott részhalmazára automatizálható.

## HIVATKOZÁSOK

Baier C. and Katoen J.-P. (2008), *Principles of Model Checking*. The MIT Press.

Behrmann G., Larsen K., Moller O., David A., Pettersson P., and Yi W. (2001). Uppaal - present and future, in *Proceedings of the 40th IEEE Conference on Decision and Control* (Cat. No.01CH37228), **vol. 3**, pp. 2881–2886 vol.3.

CENELEC (2018). EN 50129, *Railway applications – Communication, signalling, and processing systems – Safety-related electronic systems for signalling.*

Chatterjee K. and Doyen L. (2016). Computation tree logic for synchronization properties, *In Proc. of ICALP: Automata, Languages, and Programming*, **vol. 55**, p. 98:1–98:14, 10.4230/LIPIcs.ICALP.2016.XXX.

Demri S., Goranko V., and Lange M. (2016). *Temporal Logics in Computer Science, Finite-State Systems.* Cambridge University Press.

Giannakopoulou D., Pressburger T., Mavridou A., and Schumann J. (2020). Generation of formal requirements from structured natural language, *in Requirements Engineering: Foundation for Software Quality*, N. Madhavji, L. Pasquale, A. Ferrari, and S. Gnesi, Eds. Cham: Springer International Publishing, pp. 19–35.

Fisher M. D. (2011). *An Introduction to Practical Formal Methods Using Temporal Logic.* Wiley, ISBN 978-0-470-02788-2.

Hood C., Wiedemann S., Fichtinger S., and Pautz U. (2007). *Requirements Management: The Interface Between Requirements Development and All Other Systems*

*Engineering Processes*. Springer Berlin Heidelberg, ISBN 9783540476894.

Hunt B. R., Lipsman R. L., Rosenberg J. M., Coombes K. R., Osborn J. E., and Stuck G. J. (2001). *A guide to MATLAB for Beginners and Experienced Users*. Cambridge University.

ISO/IEC/IEEE (2015). ISO/IEC/IEEE 15288, *Systems and software engineering — System life cycle processes*.

Kunnappilly A., Backeman P., and Seceleanu C. (2021). From UML modeling to UPPAAL model checking of 5G dynamic service orchestration, *in 7th Conference on the Engineering of Computer Based Systems, ser. ECBS 2021.* New York, NY, USA: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/3459960.3459965

Lamsweerde A. van (2009), *Requirements Engineering - From System Goals to UML Models to Software Specifications*, Wiley, ISBN 978-0-470-01270-3.

Lukacs G. and Bartha T. (2019). Construction of formal models and verifying property specifications through an example of railway interlocking systems, *Pollack Periodica: An International Journal for Engineering and Information Sciences*, **vol. 14**, pp. 39–50, 2019.

Lukács G. and Bartha T. (2021). Formal modelling of level crossing system for trams using framework UPPAAL, *Technical Review (EMT)*, 77 pp. 18-37., 20 p.

Mirabadi A. and Yazdi M. B. (2009). Automatic generation and verification of railway interlocking control tables using FSM and NuSMV, *Transport Problems (Problemy Transportu)*, **vol. 4**, no. 1, pp. 103–110.

OMG, OMG *Unified Modeling Language (OMG UML),* Superstructure, Version 2.4.1, 2011. Object Management Group Std., Rev. 2.4.1, [Online]. Available: http://www.omg.org/spec/UML/2.4.1

Papke B. L., Wang G., Kratzke R., and Schreiber C. (2020). Implementing MBSE– an enterprise approach to an enterprise problem, *INCOSE International Symposium*, **vol. 30**, no. 1, pp. 1550–1567, https://doi.org/10.1002/j.2334-5837.2020.00803.x.

Parviainen P., Takalo J., Teppola S., and Tihinen M. (2009). *Model-Driven Development: Processes and practices*, ser. VTT Working Papers. Finland: VTT Technical Research Centre of Finland, no. 114, project code: 7326

Rakesh L. and Kadakolmath L. (2018). Modeling and formal verification of SMT rail interlocking system using PyNuSMV, in 2018 4th *International Conference on Recent Advances in Information Technology (RAIT)*, pp. 1–8.

Schneider S. A. (2001), *The B-method - an introduction*, The cornerstones of computing series. Macmillan Publ. ISBN 978-0-333-79284-1.

SEBoK Editorial Board (2022). *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*, v. 2.6, R.J. Cloutier (Editor in Chief). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. www.sebokwiki.org.

Selic B. (2006). Model-driven development: its essence and opportunities, *in Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing* (ISORC'06), 2006, p. 7.

Solis C. and Wang X. (2011). A study of the characteristics of behavior driven development, *in 2011 37th EUROMICRO Conference on Soft- ware Engineering and Advanced Applications*, pp. 383–387.

Ugur P. Y., Helvacioglu I. Y. A. B., and Asa B. N. (2022). The requirement cube: A requirement template for business, user, and functional requirements with 5w1h approach, *International Journal of Information System Modeling and Design (IJISMD)*, **vol. 13**, no. 1, pp. 1–18, 2022, 17th IFAC World Congress.

Woodcock J., Larsen P. G., Bicarregui J., and Fitzgerald J., (2009). Formal methods: Practice and experience, *ACM Comput. Surv.*, **vol. 41**, no. 4. [Online]. Available: https://doi.org/10.1145/1592434.1592436