

TDK-dolgozat

Szerző: Márczi Brigitta

Technológiai rendszerek diagnosztikai vizsgálata folyamatinformációk alapján generált Petri hálókkal

Szerző: Márczi Brigitta (1. évfolyam)
Témavezetők: dr. Gerzson Miklós (egyetemi docens)
dr. Leitold Adrien (egyetemi docens)
Intézmény: Pannon Egyetem, Műszaki Informatikai Kar,
Villamosmérnöki és Információs Rendszerek Tanszék

Absztrakt

Egy gyártórendszer működése során ismert, hogy mely technológiai műveleteken át juthatunk el a végcélra és ezen műveletek egymáshoz rendelt viszonya. Ezeket az információkat a mért értékekkel kiegészítve a valós rendszer működését jól leíró modell elkészítése válik lehetővé. A modellezés azonban nagyon sok esetben a rendszereknek csak a hibamentes működéséig terjed, kevesebb figyelmet fordítva a hibadetektálásra és a meghibásodás esetén teendő lépésekre. A hibadetektálás és a hibaanalizálás azonban kiemelkedő jelentőségű a legtöbb rendszer működése során, ezért a dolgozat célja egy ezt támogató módszer kidolgozása. Munkánk során a modellezést a diszkrét eseményű rendszermodellek csoportjába tartozó színezett Petri hálók segítségével végeztük el, mivel a technológiai rendszerek széles köre modellezhető diszkrét idejű modellek segítségével.

A dolgozatban egy új eljárást javasolunk diszkrét eseményű rendszerek modell-alapú diagnosztikájára. Módszerünk két elvi lépésből áll: először a mért adatokból előállítjuk az aktuális modellt, majd összehasonlítjuk azt, az ismert hibás vagy hibamentes viselkedést leíró referencia modellekkel. A vizsgálatokhoz szükséges modelleket színezett Petri hálók formájában írjuk le.

A referencia modellek megalkotásakor először az elméleti, hibamentes működést leíró modellt hozzuk létre a rendelkezésre álló technológiai rendszer felépítésére, működésére vonatkozó adatok alapján, majd következő lépésben a lehetséges hibás működési módokat integráljuk a rendszerbe oly módon, hogy a normál működést leíró háló szerkezete nem változzon. A technológiai folyamatok üzemeltetése során végzett megfigyelésekből nyert információkat a naplózás révén kapott log fájlok rögzítik. Hasonló jellegű adatokat kaphatunk a referencia modell szimulációjakor a különböző működési eseteket (normál, ismert és ismeretlen hibás) vizsgálva. Ezeknek a fájloknak az elemzésére adatbányászati, valamint saját fejlesztésű szoftvert használunk. A Converter elnevezésű programunk célja, hogy a referencia modellből és az esetleges hibás működést rögzítő log fájlokból olyan hálószerkezeteket generáljon, amely egyrészt a felhasználó igényeinek megfelelő hibákat megjeleníti a hálóban, másrészt további folyamatbányászati vizsgálatokat tesz lehetővé.

A technológiai rendszer működésének elemzésére gráfok szerkezetét vizsgáló módszert alkalmazunk. A megvalósított hibadiagnosztikai módszer a visszaállított, illetve referenciaként szolgáló, normál és a különböző hibás működéseket leíró Petri hálók összehasonlításán alapul. Hibás működés esetén a hiba súlyosságát mérhetjük a normál működést leíró és az adott hibát megjelenítő Petri hálók távolságával. Ezt az értéket a két gráf maximális közös részgráfjának alapján számolhatjuk. Az eljárás eredményeként meghatározhatóvá válnak a vizsgált rendszer működési állapotai, következtetünk a rendszer működésének helyességére, illetve az esetlegesen előforduló hiba típusára, vagy hatásának mértékére.

A dolgozatban konkrét esettanulmány felhasználásával mutatjuk be az általunk javasolt új hibadiagnosztikai eljárást.

Tartalomjegyzék

1. Bevezetés	5
2. Elméleti áttekintés	7
2.1. Alacsony szintű Petri hálók formális és gráfrepresentációs leírása	7
2.2. Kiterjesztett Petri hálók	13
2.2.1 Színezett Petri hálók	13
2.2.2 Hierarchikus Petri hálók	16
2.2.3 Időzített Petri hálók	16
2.3. Adatbányászat	17
2.4. Gráfok távolságának meghatározása	18
2.5. Modellezést, elemzést segítő szoftverek	20
3. Saját eredmények	22
3.1. Hibadiagnosztika színezett Petri hálók segítségével	22
3.1.1 Normál referencia modell	22
3.1.2 Hibás működés lehetőségét is biztosító Petri háló modellek	23
3.1.3 Naplóállományok és felhasználásuk	24
3.1.4 Gráfszerkezeti vizsgálatok	25
3.1.5 Converter	25
3.2. Esettanulmány	29
3.2.1 Üdítőgyártó modell normál működésének rövid áttekintése	29
3.2.2 Hibák modellbe való integrálása	30
3.2.3 Elemzések	32
4. Összegzés	37
5. Melléklet	40

1. Bevezetés

A modell a valóság – lényeges vonásokat tükröző – leegyszerűsített mása. Ily módon egyrészt szegényebb a valóságnál, hiszen nem hordozza annak minden jellemzőjét, másrészt gazdagabb is annál, hiszen a valóság végtelen sok jellemzőjéből kiemeli az adott szempontból legfontosabbakat, s ezzel olyan összefüggések felismeréséhez is vezethet, amelyek éppen a valóság sokrétősége miatt voltak rejtve [1].

A modellezés fő célja a valóság egy meghatározott formában történő leképezése elemzés céljából. A modellezés a fejlesztés tipikusan korai szakaszának része, a tervezés során vétett hibák és hiányosságok javításának egyik fő eszköze még a valós konstrukció létrehozása előtt. A modellek segítségével szélesebb képet kaphatunk a tervezett rendszerünkről, elvégezhetjük rajta a verifikálás és validálás folyamatait is, tovább tökéletesíthetjük a terveket, vagyis segítségünkre van a vizsgált rendszer minél jobb működésének elérésében. Ennek is köszönhető, hogy a modell alapú technikák [2] nagyon elterjedtek és széles körben használatosak. E dolgozat célja, hogy a technológiai rendszerek során használt Petri háló alapú modellezési formát ismertesse, valamint ezekből kiindulva egy új eljárást ajánljon diagnosztikai elemzésre. Ezen új módszer a rendszerekhez rendelt modellek gráftávolságának meghatározásán alapszik.

I. Problémafelvetés

Tegyük fel, hogy van egy működőképes rendszerünk, melynek működése során kisebb-nagyobb hibák előfordulhatnak, de összességben ellátja a feladatát. Keressük a választ a választ a következőkre: ismerjük-e az összes hiba lehetőségét, meg tudjuk-e magyarázni az egyes hibák okait, tudjuk-e, hogy hol következett be a hiba és milyen hatást gyakorolt az egész rendszerre nézve, valamint melyek a rendszer kritikus pontjai? Amennyiben sikerrel feltárjuk ezeket az ok-okozati viszonyokat, világossá tesszük a hibák következményeit, és a kritikus pontokra koncentrálna változtatásokat hajtunk végre, akkor működés terén egy megbízhatóbb, hatékonyabb rendszert kaphatunk [3].

A kérdések megválaszolásában segítenek a rendszer naplózása során keletkező adatsorok, valamint azon alap rendszerismeretek, amik a folyamatok viszonyát, az elvárt működés ismeretét feltételezik. Ez utóbbi ismeretek a hibamentes működést leíró modell megalkotására adnak lehetőséget. Amikor azonban modell alapú hibadetektálást és megkülönböztetést végzünk, akkor nemcsak a rendszer normál működésének modelljére, hanem olyan modellekre is szükségünk van, amik leírják a lehetséges hibás üzemmódokat.

Az egyes eseményfájlok azonban nagyon nagy mennyiségű adatot tartalmaznak, így szükség van egy olyan módszerre, amelynek segítségével a fontos információk kinyerhetők belőle. Ha már rendelkezésre állnak a szűrt adatok, akkor következő lépésként egy olyan modellt kell felállítani, aminek elemzésével elérhetjük a kitűzött célokat. Ezekből a log elemekből a működést helyesen leíró, elemezhető gráfmodell visszaállítása azonban nem triviális feladat, mivel az események egymáshoz rendelt ok-okozati viszonya általában nem kerül rögzítésre, így csak következtetésekkel lehet belőle valamilyen eseményeken alapuló leíró modellt, például Petri hálót [4] visszaállítani. Ez a háló azonban az említett ok miatt redundáns éleket és csúcsokat tartalmazhat, amik jelentősen lassíthatják az elemzést, mivel

egy adott induló állapotból elérhető állapotok halmaza a szükségesnél nagyobb lesz.

E dolgozat erre a problémakörre ajánl egy új diagnosztikai célú modell-konstrukciós eljárást.

II. Problémafelvetés

Legyen a célunk egy technológiai rendszer létrehozása és a rendszerben előforduló hibák hatásának felmérése. Ehhez kiindulásként alkossuk meg a rendszer modelljét. A modell létrehozásánál vegyük figyelembe, hogy a vizsgált rendszerben meghibásodások, zavarások is előfordulhatnak.

Végezzünk hatásvizsgálatot, ami lehetőséget ad arra, hogy szükség esetén még a megvalósítás előtt újratervezzük a rendszert, figyelve a kritikus részekre, valamint ennek segítségével megfelelően felkészüljünk ezek kezelésére.

Alkalmas modellezési eszközt választva lehetőségünk van az elkészített modell szimulációjára, ami során a valós rendszer működésénél kapott eseménynaplókhoz hasonló log fájlokhoz juthatunk, melyek segítségével elvégezhető a rendszer validálása. Mint látható, a tervezési feladat esetében is hasonló kiindulási adatokat kapunk és a megoldandó probléma is hasonló az I. problémafelvetésben leírt, az üzemeltetés során végzett diagnosztikai vizsgálatokhoz, így az ott ismertetett megoldás ebben az esetben is alkalmazható.

Dolgozat felépítése

A bevezetés és problémafelvetések után, a dolgozat második fejezete áttekintést ad munkák elméleti háttéréről. Bemutatásra kerülnek a Petri hálók legfontosabb tulajdonságai, a folyamatbányászat alapjai, a maximális közös részgráfokon alapuló gráftávolság számításhoz kapcsolódó alapdefiníciók és az általunk is használt modellezési eszközök, programok.

A harmadik főrész saját eredményeinket tartalmazza, lépésről lépésre ismertetve a színezett Petri hálók segítségével történő hibadetekciós és hatásvizsgáló eljárásunkat, valamint az elemzéseket támogató saját fejlesztésű szoftverünket. A kidolgozott eljárásunk legfontosabb lépéseit egy egyszerűbb technológia rendszer esettanulmányának segítségével mutatjuk be. Az esettanulmányhoz kapcsolódó nagyobb méretű ábrákat a jobb olvashatóság céljából a Melléklet tartalmazza. Az ezekre történő utalás *Melléklet x. ábra* formában történik.

A dolgozat negyedik fejezetben eddigi eredményeink és további céljaink összefoglalása olvasható, míg az utána lévő részben az irodalmi hivatkozások kaptak helyet.

2. Elméleti áttekintés

Egy vizsgált rendszerben végbemenő események és lehetséges állapotok leírására alkalmas forma a Petri hálóval történő modellezés. Elnevezését Carl Adam Petri (1926-2010) német matematikusról kapta, aki 1961-ben publikálta először módszerét. Édesapja szintén a matematika tudományterületének doktora volt. C. A. Petri mindössze 13 évesen alkotta meg a most is használatos jelölésrendszer alapjait, igaz ekkor még csak kémiai folyamatok leírására szánva. Módszerének matematikai úton történő alátámasztását doktori disszertációjában („Kommunikation mit Automaten”) mindössze két hét alatt dolgozta ki [5].

A Petri hálók lehetőséget biztosítanak dinamikus rendszerek leírására, továbbá kiválóan alkalmazhatók diszkrét eseményű és diszkrét idejű rendszerek gráfrepresentációval történő modellezésére is. Manapság megkülönböztetünk alacsony és magas szintű valamint hierarchikus hálókat.

A következő alfejezetekben a Petri hálók fontosabb típusait mutatjuk be [4], majd áttekintjük a folyamatbányászat alapjait, és ismertetjük a gráfok összehasonlításához szükséges definíciókat.

2.1. Alacsony szintű Petri hálók formális és gráfrepresentációs leírása

A Petri hálók irányított, súlyozott páros gráfok. Formális módon, egy alacsony szintű Petri hálót a (P, T, A, w) négyessel definiálhatunk, ahol:

- P : a helyek véges halmaza,
- T : az átmenetek véges halmaza,
- $A : A \subseteq (P \times T) \cup (T \times P)$ irányítások (élek) halmaza,
- $w : A \rightarrow \mathbb{Z}^+$ irányítások súlya.

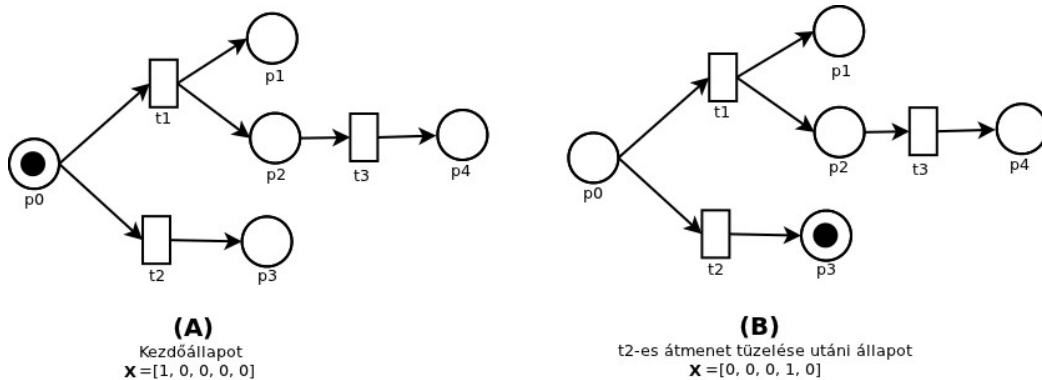
Továbbá feltesszük, hogy a hálóban nincsenek izolált helyek és átmenetek.

A gráfrepresentáció során a gráf csúcsait helyeknek és átmeneteknek feleltetjük meg úgy, hogy a helyeket körökkel az átmeneteket pedig téglalapokkal jelöljük. Rendszermodellezés oldaláról nézve a helyek egy esemény vagy művelet előfeltételeinek és következményeinek, az átmenetek pedig a lehetséges eseményeknek, technológiai beavatkozásoknak a leírására szolgálnak. A gráfban található irányított, súlyozott élek a helyek és az állapotok között lévő logikai relációt írják le. A helyek mindig átmenetekhez, az átmenetek mindig helyekhez kapcsolódnak élek segítségével. Két hely valamint két átmenet összekapcsolása nem megengedett.

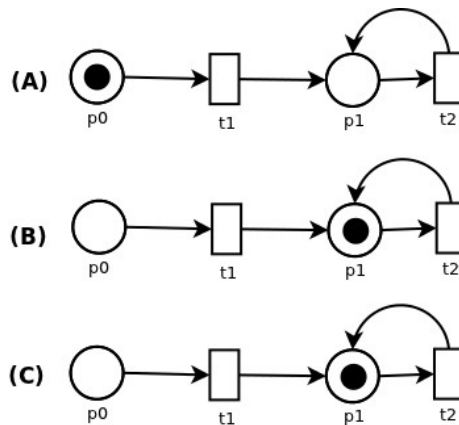
A helyekhez lehetőség van jelző pontokat, más néven tokeneket rendelni. Ezeknek a tokeneknek a segítségével valamilyen dolog (pl. termék, információ, állapot stb.) létét vagy hiányát, valamint számosságát jelölhetjük. Ennek megfelelően a jelzőpontok a rendszer állapotát írják le, és a rendszer viselkedésének meghatározásakor van szerepük. A rendszer állapotainak száma végtelen is lehet, mivel a tokenek száma az egyes helyeken felső korlát nélküli, nemnegatív egész. Ha a hálóban található token, akkor *marked* (jelölt) Petri hálóról

beszélünk, ami formálisan (P, T, A, w, x) ötössel adható meg, ahol P, T, A és w megfelel az alacsony szintű Petri háló definíciójában szereplő fogalmaknak,

- $x : P \rightarrow N$, a jelzőpontoknak a helyeken való eloszlását megadó függvény, valamint $\underline{x} = [x(p_1), x(p_2), \dots, x(p_n)] \in N^n$ állapotvektor, amely rendre az egyes helyekhez tartozó tokenek számát tartalmazza.



2.1. ábra. Holtponthelyzet



2.2. ábra. Livelock

A Petri hálók állapotváltozása a tokenek számának változásán keresztül valósul meg. A jelzőpontok számának az egyes helyeken való változtatásával lehet a háló működését vizsgálni, így azok pillanatnyi eloszlásai a modellezett rendszerbeli viszonyokat is jellemzik. Azt mondjuk, hogy a t_j átmenet képes *tokenek mozgatására*, vagyis *tüzelőképes*, ha

- $\underline{x}(p_i) \geq w(p_i, t_j)$, $(\forall p_i \in P \mid (p_i, t_j) \in A)$.

Ennek teljesülése esetén a t_j átmenetet *engedélyezett átmenetnek* nevezzük. Másként megfogalmazva, tüzelésre akkor van lehetőség, hogy ha az adott átmenethez kapcsolódó összes előfeltételi helyen legalább annyi token található, mint az adott hely és az átmenet között lévő irányított él súlya. Egy eseménysorozat (*tüzelési szekvencia*) lejátszódásakor az egyes helyeken a tokenek száma csökkenhet, növekedhet vagy akár változatlan is maradhat. Ez nem jelent mást, mint, a tokeneloszlás változásának lehetőségét tüzelés alkalmával. Az

egyes helyeken végbemenő tokenszám változás a következő módon definiálható:

$$f : N^n \times T \rightarrow N^n \text{ átviteli függvény,}$$

$$\underline{x}'(p) = \underline{x}(p) - w(p, t) + w(t, p),$$

$$f(x, t) = \begin{cases} \{(p, \underline{x}'(p))\} & | p \in P \text{ ha } t \text{ engedélyezett} \\ \text{nem definiált} & | t \text{ nem engedélyezett} \end{cases}$$

Ha egy háló adott állapotában van tüzelésre képes átmenet, akkor azt *aktív átmenetnek* is nevezik. Amennyiben nincs olyan átmenet, amely tüzelni tudna, akkor a rendszer *holtpontra* (deadlock) jutott. Fontos kiemelni, hogy egy háló holtpontmentességéből nem feltétlen következik a modellezett rendszer helyes működése, mivel végtelen ciklusba kerülés lehetőségét nem zárja ki. Ezen kívül holtpont kialakulása esetén érdemes megvizsgálni a tokeneloszlást olyan szempontból is, hogy egy elfogadóállapotnak (célállapotnak) megfelelő eloszlásról van-e szó. A holtpont kialakulását 2.1-es ábra szemlélteti, melyen látható, hogy a t_2 -es átmenet tüzelése után nem lesz engedélyezett átmenet.

A másik blokkolási lehetőség, ha van a jelöletlen állapotoknak egy olyan U halmaza, ahova ugyan eljuthatunk egy másik állapotból, de ezután nincs olyan tüzelési szekvencia, ami kivezetne U -ból. Az így kialakult helyzetet *livelock*nak nevezik. A 2.2-es ábra (A) része a kiindulóhelyzetet, (B) és (C) része rendre a t_1 -es és t_2 -es átmenetek tüzelése utáni állapotot mutatja. A (C) részt megvizsgálva látható, hogy a t_2 -es átmenet tüzelése után engedélyezett marad, viszont más átmenet nem válik engedélyezetté, így livelock alakul ki, ami a végtelen ciklusban történő működésnek felel meg.

Ha a hálóban egyszerre több engedélyezett átmenet is található, akkor a következő tüzelő átmenet kiválasztása véletlenszerűen történik. Ebben az esetben a Petri háló végrehajtása nem-determinisztikusnak tekinthető. Ilyenkor meg kell vizsgálni, hogy a tüzelőképes átmenetek egymástól függetlenek-e. Ha kettő vagy több átmenetnek ugyanaz a hely előfeltétele, és ezek az átmenetek egyszerre válnak engedélyezetté (vagyis az előfeltételeik nem függetlenek), akkor a kialakult helyzetet *konkurenciának* nevezzük. Erre példa az 2.1-es ábra t_1 -es és t_2 -es átmenetei esetében látható, ahol a tüzelő átmenet kiválasztása véletlenszerűen történik. A választott átmenet lejátszódása után a másik átmenet nem lesz engedélyezett.

A következőkben Petri hálók dinamikai vizsgálatai közül a leggyakrabban használtakat emeljük ki.

Elérhetőség és lefedhetőség

Az elsőként bemutatásra kerülő dinamikai vizsgálat, az elérhetőségi vizsgálat, melynek segítségével ellenőrizni tudjuk, hogy a kívánt vagy elkerülendő állapotok bekövetkezhetnek-e a vizsgált technológiai rendszer működése során.

- *Elérhetőség, elérhető állapotok halmaza:* Egy $H(P, T, A, w, x)$ Petri háló esetén, ha van olyan tüzelési szekvencia, azaz engedélyezett átmenetek olyan sorozata, amely az

$\underline{x}_0$ kezdőállapotból egy adott $\underline{y}$ állapotba viszi át a tokeneloszlás vektort, akkor azt mondjuk, hogy az $\underline{y}$ állapot $\underline{x}_0$ kezdőállapotból *elérhető*. A H Petri háló *elérhetőségi halmaza* alatt a következőt értjük:

$$R[H, \underline{x}_0] := \{ \underline{y} \in N^n : \exists s \in T^* (f(\underline{x}, s)) = \underline{y} \} \quad \diamond$$

- *Halott átmenetek*: Az elérhetőségi vizsgálat során felderíthetünk olyan átmeneteket is, amelyek egy adott $\underline{x}_0$ kezdőállapotból kiinduló tüzelési szekvenciák egyikében sem lesznek engedélyezettek. Ezeket az átmeneteket *halott átmeneteknek* nevezzük.
- *Lefedési vizsgálat*: Az elérhetőségi vizsgálat kiterjesztése a lefedési vizsgálat. Egy tetszőleges állapot lefedésén a definíció szerint a következőt értjük:
- Egy $\underline{x}_0$ kezdőállapottal adott H Petri háló $\underline{y}$ állapota lefedhető, ha $\exists \underline{x} \in R(H, \underline{x}_0)$, hogy $\underline{x}(p_i) \geq \underline{y}(p_i) \quad \forall i = 1 \dots n$ $\diamond$

Petri háló elérhetőségi, lefedhetőségi fája

A modell és a rendszer tulajdonságainak verifikálását az állapotátmeneti leírások támogatják. Az elérhetőségi fa felvételéhez határozzuk meg egy Petri háló adott $\underline{x}_0$ kezdőállapotból elérhető valamennyi állapotát, majd ábrázoljuk azt egy olyan gráfon, melyen a csúcsok az állapotoknak, az irányított élek pedig az engedélyezett átmeneteknek felelnek meg. Ennek a fajta ábrázolásnak az egyik előnye, hogy választ adhat az ismertetett viselkedési kérdésekre. Ilyen kérdések lehetnek például, hogy holpontra tud-e jutni a rendszer, mik a lehetséges elérhető állapotok, milyen úton juthatunk egy adott állapotba, milyen úton lehetséges az elfogadó állapotok elérése, a kívánt feladatot hajtja-e végre a rendszer, van-e kör? További előnye még, hogy segíthet a rendszerek hibamentesítésében is oly módon, hogy megválaszolja, hogy miért fordulhattak elő a fejlesztő/felhasználó által nem várt állapotok. A Petri háló modellből, a fentiekben leírt ábrázolási móddal létrehozott gráfot, *elérhetőségi fának* (reachability tree) szokás nevezni. Az elérhetőségi gráf létrehozása jól algoritmizálható folyamat. Fő hátránya azonban az „állapotok robbanása”. Speciális algoritmusok segítségével azonban a csúcspontok száma végezzé tehető. A véges reprezentáció megvalósítására a *lefedhetőségi fa* (coverability tree) vagy gráf megoldást jelent, de ez az információ egy részének elvesztésébe kerülhet. Az elérhetőségi fa véges változatát tehát lefedhetőségi fának nevezzük.

Korlátosság

A korlátosság vizsgálatát vezessük be egy példával. Tegyük fel, hogy a vizsgált rendszer egyik alrendszerének feladata a munkadarabok tömegének lemérése. A munkadarabok automatikusan és egyesével érkeznek a mérési helyre, majd mérés után egy robotkar teszi tovább őket egy szállítoszalagra. Ha a robotkar jól működik, akkor a mérleg mindig csak egy munkadarab tömegét határozza meg. Meghibásodása esetén a mérlegen több munkadarab mérése történik.

Petri hálóval történő modellezés során a mérleget egy hely fogja megjeleníteni, míg a munkadarabokat a tokenek szimbolizálják. Ennek megfelelően helyes működéskor a mérlegnek megfelelő helyen mindig csak egy jelzőpont található, míg meghibásodáskor a tokenek száma tetszőleges lehet. A korlátossági vizsgálat nem jelent mást, mint annak ellenőrzését, hogy az egyes helyeken található tokenek száma meghaladja-e az adott helyhez

rendelt maximális tokenszámot. Természetesen lehetnek olyan helyek is, amelyeken a tokenek száma nincs korlátozva.

Definíció szerint egy p_j helyet, akkor nevezünk *korlátosnak*, ha az összes elérhető állapotra igaz, hogy a p_j -n található tokenek száma a hozzá tartozó k korlát értékét nem lépi át.

Egy $\mathbf{x}_0$ kezdőállapotú H Petri háló $p_i \in P$ helyét k -szorosán korlátosnak nevezzük, ha $\mathbf{x}(p_i) \leq k, \forall \mathbf{x} \in R(H, \mathbf{x}_0)$ esetén. $\diamond$

Amennyiben ez a háló minden helyére érvényesül, akkor a hálót *korlátos hálónak* nevezzük. Abban az esetben, ha ez $k=1$ értékkel teljesül, akkor a hálót *biztonsági hálónak* nevezzük.

Konzervativitás

Egy másik érdekes dinamikai vizsgálat a konzervativitás vizsgálata, melynek során a modellben található tokenek számának változását követhetjük nyomon. Ha egy elosztott rendszer példáját tekintjük, ahol a tokenek például az egyes erőforrásoknak felelnek meg, akkor konzervativitás vizsgálat segítségével észrevehető ha egy erőforrás kiesik a rendszerből. A konzervativitást a következőképpen definiálhatjuk:

A H Petri hálót $\mathbf{x}_0$ kezdeti állapottal és $\underline{\beta}^T = [\beta_1, \dots, \beta_n]$ vektorral konzervatívnek

nevezzük, ha $\sum_{i=1}^n \beta_i \mathbf{x}(p_i) = \text{állandó}$, minden $\mathbf{x} \in R(H, \mathbf{x}_0)$ esetén, ahol

$\underline{\beta}^T = [\beta_1, \beta_2, \dots, \beta_n]$ súlyvektor, $\beta_i \geq 0, i=1, \dots, n, \forall p_1, \dots, p_n$ -hez. $\diamond$

Amennyiben a jelző pontok összege állandó, *szigorúan konzervatív hálóról* van szó. A konzervativitást vizsgálhatjuk részhalókra is.

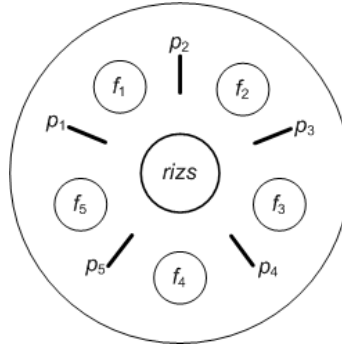
Perzisztencia

Tegyük fel, hogy a Petri háló egy adott $\mathbf{x}$ állapotában két vagy több engedélyezett átmenet van. Ekkor meg kell vizsgálni, hogy az egyik átmenet tüzelése után a másik állapot engedélyezett marad-e. Definíció szerint:

Egy Petri hálót perzisztensnek nevezünk ha igaz az, hogy bármely két engedélyezett átmenet közül az egyik tüzelése esetén, a másik átmenet engedélyezettsége nem változik. $\diamond$

Étkező filozófusok problémakörének ismertetése

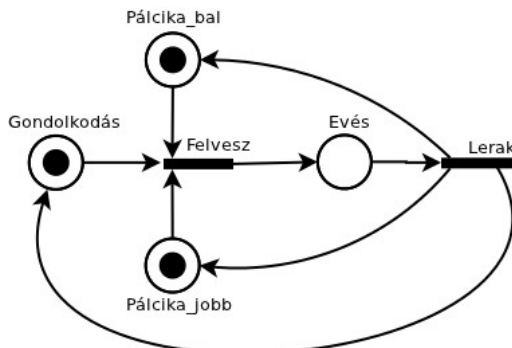
A fejezetben ismertetett Petri hálóval történő modellezés alapjait az irodalomból ismert, ún. „étkező filozófusok” példáján keresztül mutatjuk be. Célunk itt nem az eredeti probléma megoldása, hanem az alacsony szintű, valamint a kiterjesztett Petri hálók modellező erejének bemutatása.



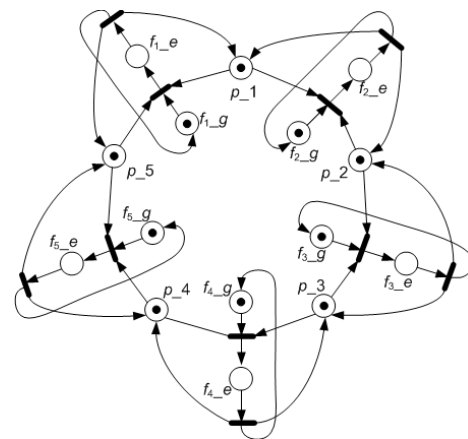
2.3. ábra. Filozófusok (f_1 - f_5) és az evőpálcikák elhelyezkedése (p_1 - p_5)

Az eredeti problémát E.Dijkstra vetette fel hallgatóinak feladatként, ami később számos változatban jelent meg a szakirodalomban az erőforrás elosztási feladatok alaptípusaként. Az alapfeladat szerint egy asztal körül öt filozófus ül a 2.3-as ábrán látható elrendezés szerint. Az asztal közepén egy tál rizs található. Feltételezzük, hogy a rizs sosem fogy el. A filozófusok között az asztalon egy-egy evőpálcika van az ábrának megfelelően. A filozófusok az asztalnál vagy gondolkoznak vagy esznek. Az evéshez azonban saját tányérjuk mindkét oldalán található evőpálcikára szükségük van, mely kizárja annak a lehetőségét, hogy az egymás melletti filozófusok egyszerre étkezzenek. Ha az összes filozófus egyszerre veszi fel például a jobb kézre eső evőpálcikáját, akkor holtponthelyzet alakul ki. A feladat olyan megoldás kidolgozása, melyben ez a holtpont nem alakulhat ki. Jól látható módon az ismertetett feladat a hatékony erőforrás-elosztás problémáját modellezi.

Egy filozófus állapotainak modellezése a 2.4-es ábrán látható. A helyek közül a *Pálcika_bal* és *Pálcika_jobb* a bal és jobb oldali evőpálcikák elérhetőségét, a *Gondolkodás* és *Evés* a filozófus állapotát, a *Felvesz* átmenet az evőpálcikák felvételét, míg a *Lerak* a letételeiket modellezi. A *Felvesz* átmenet csak akkor játszódhat le, ha a filozófus éppen gondolkodik és mind a két evőpálcika szabad. Az öt filozófus estének modellezése az *Hiba: A hivatkozás forrása nem található-ös ábrán* látható. Itt az f_{i_e} az i -dik filozófus evését, az f_{i_g} a gondolkodó állapotát modellezi, míg a p_i a pálcikák elérhetőségét.



2.4. ábra. Egy filozófus állapotait leíró Petri háló modell



2.5. ábra. Öt filozófus esetének Petri hálós modellezése

Az étkező filozófusok példája esetében az *Hiba: A hivatkozás forrása nem található-ös ábra* alapján belátható, hogy a megadott háló nem konzervatív, mivel étkezéskor az evőpálcikák helyein nincsenek tokenek. Ha azonban a pálcikák foglalt, illetve szabad állapotát külön hellyel modellezzük a háló átalakítható konzervatívvá.

Nyilvánvalóan látszik az is, hogy a példaháló nem perzisztens, hiszen a filozófusok *versenyhelyzetben* vannak az evőpálcikák használatáért, vagyis ha valamelyik filozófus el kezd enni, akkor a szomszédjának biztos várakoznia kell.

2.2. Kiterjesztett Petri hálók

A 2.1-es fejezetben bemutatott alacsony szintű Petri hálók bár jól alkalmazhatóak diszkrét eseményű, illetve diszkrét értékű rendszerek leírására, azonban nagyobb rendszerek modellezésekor vagy részletesebb modellek készítésekor a kapott háló és így az elemzése is túlságosan bonyolulttá válhat. Így a komplex rendszerek vagy folyamatok hatékony modellezéséhez szükségessé vált az alacsony szintű Petri hálók kibővítése. A következő részek ezen bővítésekről adnak áttekintést. A 2.1-es fejezetben az alacsony szintű Petri hálókra kimondott jelölési konvekciók, definíciók, elemzési lehetőségek analóg módon a kiterjesztett Petri hálókra is érvényesek. A bemutatásra kerülő hálótípusok gyakran együttesen fordulnak elő, vagyis például egy színezett Petri háló is lehet hierarchikus és időzített.

2.2.1 Színezett Petri hálók

A színezett Petri hálók az alacsony szintű Petri hálók kiterjesztései. A dániai Aarhus egyetemen már 1979 óta folyik a fejlesztésük. Céljuk egy elméletileg jól alátámasztott, a gyakorlatban sokoldalúan használható, a méretet és komplexitást is kezelni tudó modellező nyelv megalkotása volt. A színezett Petri hálókat a létező modellező nyelvek mellé kiegészítésnek vagy azokkal egymásba építve ajánlatos használni.

Színezett Petri hálóról beszélünk, ha az alacsony szintű Petri hálónál bevezetett tokenek a számosságukon kívül, valamilyen egyéb információt is hordoznak, azaz megkülönböztethetők. Ezen felül lehetőség van az átmenetekhez és élekhez feltételek (diszkrét függvények) hozzárendelésére, a helyeknél pedig definiált egy megengedett színhalmaz. A tokenekhez rendelt diszkrét értékű adatok típusát a tokenek színének nevezik, így a „színezett Petri háló” elnevezés innen származik.

A színezett Petri hálók és az alacsony szintű Petri hálók közti eltérésekből kiemelném, hogy színezett Petri hálók esetén:

- *A tokenekhez attribútumokat rendelhetünk,*
 - melyeket akkor használunk, ha valamiről (pl. egy termékről, eseményről) több információt szeretnénk megjegyezni. Például, ha a tokenünk egy könyvtárban megtalálható könyvet reprezentál amiről szeretnénk tárolni, hogy mi az azonosítószáma, címe, oldalszáma, kiadója stb., akkor a könyvről ezen információkat a tokenekhez csatoljuk. Ezen attribútum értékek az egyes tüzelések alkalmával változhatnak. Az előbbi példát folytatva, ha feltételezünk egy „*kölcsönözve*” mezőt, akkor értelemszerűen, kölcsönzés esetén a hozzá rendelt érték (bináris esetén igenről nemre/nemről igenre) változni fog.

- A helyekhez alaptípusok tartoznak,
 - (unit, int, bool és sztring, stb.), de lehetőség van összetett típusok létrehozására is.
- Minden helyhez egy egyéni azonosító, egy színhalmaz és kezdeti állapot is tartozik.
 - A helyekhez rendelt színhalmazok megadják, hogy adott helyeken milyen színű tokenek fordulhatnak elő.
- Az átmenetek is rendelkeznek egy névazonosítóval, valamint egy kifejezés résszel is.
- A helyeket és átmeneteket összekötő irányítások leírásokat, kifejezéseket tartalmazhatnak,
 - amik kiértékelésre kerülnek és az eredmény függvényében engedik tovább a folyamatot.

A formális definíció kimondása előtt vezessük be a következő jelöléseket:

Jelölje:

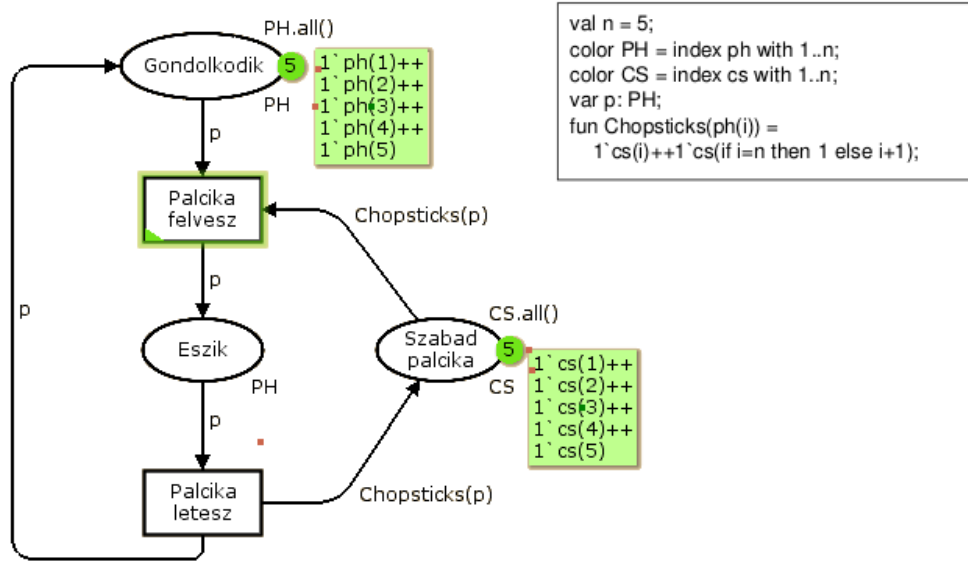
- $Type(k)$: egy k változó színosztályát (típusát)
- $Type(exp)$: egy exp kifejezés típusát
- $Var(exp)$: egy exp kifejezésben szereplő változók halmazát

A bevezetett jelöléseket alkalmazva a színezett Petri hálókat formálisan a következő kilencessel írhatjuk le [6]:

$CPN = (\Sigma, P, T, A, N, C, G, E, I)$, ahol

- Σ : színhalmaz; „nem üres” típusok véges halmaza,
- P : helyek véges halmaza,
- T : átmenetek véges halmaza,
- A : irányítások (élek) véges halmaza, ahol $P \cap T = P \cap A = T \cap A = \emptyset$;
- N : „csúcsfüggvény” : $A \rightarrow (P \times T) \cup (T \times P)$,
- C : színkészlet függvény: $C: P \rightarrow \Sigma$,
- G : őrfeltétel („guard függvény”) :
 $G: \forall t \in T: [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \Sigma]$,
- E : irányításokhoz tartozó kifejezésfüggvény (élkifejezés)
 $E: \forall a \in A: [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$,
 ahol $p(a)$ az $N(a)$ helye
- I : inicializálófüggvény (kezdőállapot):
 - $I: \forall p \in P: [Type(I(p)) = C(p)_{MS}]$.

Az alacsony szintű Petri hálónál már bemutatásra került étkező filozófusok problémájánál látható, hogy a filozófusok számának növekedésével a modellt leíró Petri háló mérete fokozatosan növekszik, s már öt filozófus esetén is a teljes gráf áttekintése nehézkes lehet. Azonban ha modellezést színezett Petri hálókkal végezzük el, akkor a háló szerkezeti komplexitása nem nő a filozófusok számával. A 2.6-os ábrán az öt filozófust tartalmazó színezett Petri háló modell látható.



2.6. ábra. Az étkező filozófusok probléma modellezése színezett Petri hálóval, öt filozófus esetén

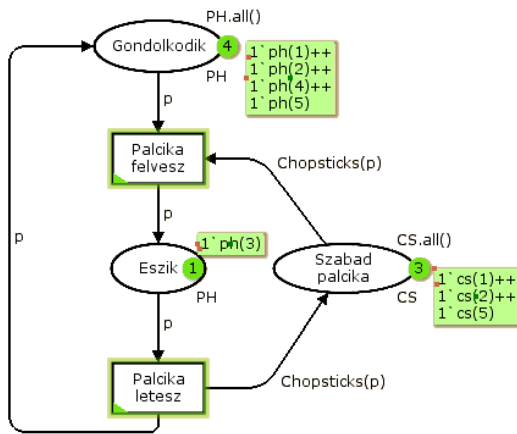
A 2.6-os ábrán látható módon a probléma modellezésénél :

- $P = \{\text{Gondolkodik}, \text{Eszik}, \text{Szabad palcika}\}$ az állapotok halmaza.
 - A *Gondolkodik* és a *Szabad palcika* helyeken kiinduláskor öt-öt token található a következő színhalmazokkal:
 - $\text{Gondolkodik} \rightarrow \{ph(1), ph(2), ph(3), ph(4), ph(5)\}$ színhalmaz
 - a színek a filozófusok megkülönböztetését szolgálják
 - $\text{Szabad palcika} \rightarrow \{cs(1), cs(2), cs(3), cs(4), cs(5)\}$
 - színek a pálcikák megkülönböztetését szolgálják
 - A *Gondolkodik* és *Eszik* hely típusa *PH*, ami a deklaráció szerint color (szín). A *Szabad palcika* hely típusa *CS* ami szintén color, de más színhalmaz tartozik hozzá.
- $T = \{\text{Palcika felvesz}, \text{Palcika letesz}\}$ az átmenetek halmaza.
- $A = \{(\text{Gondolkodik}, \text{Palcika felvesz}), (\text{Palcika felvesz}, \text{Eszik}), (\text{Eszik}, \text{Palcika letesz}), (\text{Palcika letesz}, \text{Gondolkodik}), (\text{Palcika letesz}, \text{Szabad palcika}), (\text{Szabad palcika}, \text{Palcika felvesz})\}$ az irányított élek halmaza.
 - Ezen irányításokhoz kifejezések és függvények rendelhetők. A példa esetén az éleken található „p” jelzés mondja meg, hogy milyen színű token átvitele történik. A *Chopsticks(p)* élhez rendelt függvény pedig azért felel, hogy minden filozófus csak a neki megfelelő evőpálcikát vehesse fel. Ha a filozófusok számozása a 2.3-as ábra szerint történik, akkor minden filozófus saját számával megegyező azonosítójú és az eggyel nagyobb azonosítószámú pálcikával tud enni.

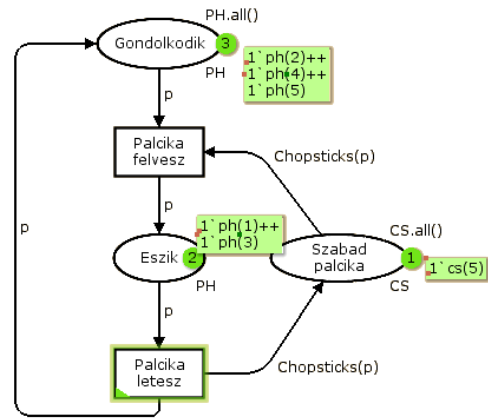
Tegyük fel, hogy először a hármas számú filozófus szeretne enni. Ekkor a *Palcika felvesz* átmenet tüzelése esetén, az *Eszik* helyre átkerül a *ph(3)*-as színű token, valamint a

Szabad pálcika hely színhalmazából kikerül a 3-as és 4-es azonosítójú pálcika, mivel a kiválasztott filozófus ezekkel tud étkezni. Az így kapott pillanatnyi tokeneloszlás a 2.7-es ábrán látható. A hármas azonosítójú filozófus étkezése, azonban nem zárja ki, az egyes azonosítójú filozófus étkezését. Étkezésük alatt, egyetlen egy pálcika (5-ös), marad használatlan. Az így kialakult állapotot a 2.8-as ábra szemlélteti.

Az így felépített színezett Petri háló modellek verifikálásában az állapottér modellek (2.1-es fejezet) nyújthatnak segítséget. Ezen modelleknél validálás alatt általában szimulációt értünk.



2.7. ábra. 3-as filozófus étkezése



2.8. ábra. 1-es és 3-as filozófusok étkezése

Szimuláció

A szimuláció során ellenőrizhetjük, hogy rendszerünk a megkívánt tulajdonságokkal rendelkezik-e. A létrehozott Petri hálónk szimulációja közvetlenül vagy automatikus módon is történhet. A lépésről-lépésre történő eseménylejátszásnál a modellező választja ki, hogy melyik aktív állapot tüzeljen. Ez a forma jól alkalmazható hibakeresés során, valamint az egyes lépések hatásának közvetlen megfigyelésére. Ezzel szemben az automatikus szimuláció esetén a gyors, hatékony, emberi beavatkozás nélküli végrehajtás a cél. Ezt a módszert tipikusan teljesítményanalízishez használják.

2.2.2 Hierarchikus Petri hálók

A hierarchikus Petri hálók főhálóból és részhálókból épülnek fel. A modulok összekapcsolása jól definiált interfészen keresztül történik. Alhálókat alkalmazva a modellezőnek lehetősége van lentről felfelé és fentről lefelé történő modellalkotásra is. Használatuk akkor ajánlott, ha a modellben ismétlődő részek fordulnak elő, valamint komplex háló esetén az átláthatóság könnyítésére. Minden hierarchikus Petri háló átalakítható nem-hierarchikus Petri hálóvá és fordítva. Számítási teljesítményt nézve nincs különbség a kétfajta háló között, azonban a hierarchikus hálók modellező ereje nagyobb. Segítségükkel pl. egy technológiai folyamat részfolyamatait különböző szinten tudjuk ábrázolni és az így kapott részhálókat szintenkénti elemzéseknek is alávetjük.

2.2.3 Időzített Petri hálók

Sok rendszer esetén az időnek meghatározó szerepe van, mivel működésük döntően függ az egyes tevékenységek idejétől. Ennek megfelelően a modell szimulációja során is szükség

lehet az idő figyelembe vételére. A Petri hálóban a helyek és átmenetek egymáshoz kapcsolása ugyan kifejez egyfajta időbeliséget, sorrendiséget, azonban az egyes műveletek hosszát, esetleges késleltetéseket nem tükrözik. Időzített Petri hálónál az átmenetekhez időzítőket lehet hozzárendelni, megadható a tüzelési idő, valamint az egyes helyeken a várakozás hossza, ami a valósághoz közelebbi modell megalkotását teszi lehetővé.

2.3. Adatbányászat

Egy rendelkezésre álló adathalmazból a számunkra fontos adatok kigyűjtése valamint rejtett kapcsolatok és összefüggések keresése az adatbányászat feladata. Az elnevezés ellenére az adatbányászat lényege, hogy egy adathalmazból a hasznos tudást (információt), összefüggéseket megkeressük. Az alkalmazások során nem ritkák a giga- sőt terrabájt nagyságú adathalmazok sem, így azok az algoritmusok célravezetők, amelyek hatékonyak, skálázhatók és amelyek a teljes adatbázist kevészer olvassák végig. További elvárás, hogy az adathalmaz fontosabb paramétereinek ismeretében az algoritmus futási ideje jósolható legyen [7].

Az adatbányászat feltételei a következők [7]:

- *nagy mennyiségű adat* álljon rendelkezésre, hogy az eseti összefüggések kizárhatóak legyenek;
- *sok attribútummal* rendelkezzen, hogy sokféle tudás kinyerésre történhessen meg;
- *tiszta* legyen az *adat*, vagyis jó minőségű, mert a hibás bejegyzések nehézséget jelentenek vagy rosszabb esetben hamis eredményre vezetnek;
- *torzíthatatlan* legyen az *adat*, vagyis figyeljünk a részsokaság kiválasztásának szempontjaira;
- *alkalmazási terület akcióképes* legyen, vagyis ne csak kinyerésre, hanem felhasználásra is kerüljön az adat;
- *a befektetés megtérülése mérhető* legyen.

A legjelentősebb adatbányászati feladatok a következők:

- gyakori minták kinyerése,
- attribútumok közötti kapcsolatok feltárása,
- klaszterezés,
- sorozatelemzés,
- eltéréselemzés,
- webes adatbányászat

Folyamatbányászat

A folyamatbányászat az adatbányászat egy speciális területének tekinthető. A folyamatbányászat célja a technológiai folyamatok adataiból a valódi, hasznosítható tudásanyag kinyerése. Lehetővé teszi a folyamatok különböző szempontok szerinti vizsgálatát, ami által számos kérdésre választ ad. Ilyen kérdések például, hogy melyik a leggyakrabban használt útvonal a folyamatban, milyen bejárható útvonalak, lehetőségek léteznek, mennyi az átfutási ideje az egyes eseményeknek? Ezen felül felderíthetők a gyártási folyamatban megjelenő hibák, amelyek a leállásokért, a hosszabb gyártási időért felelősek.

Egy intelligens irányítási rendszerrel szemben elvárt követelmény, hogy működése alatt eseménynaplót hozzon létre, amelyben rögzíti a végbemenő eseményeket, végrehajtókat, időbélyegeket, rendszermódokat, mérési adatokat és egyéb információkat. A folyamatbányászat során a folyamatkezelő információs rendszerek által létrehozott eseménynaplókból információ kinyerése történik [8].

A folyamatkezelő információs rendszerek és a folyamatbányászati alkalmazások között meg kell találni a megfelelő kapcsolatot, mivel a naplózás információs rendszerként máshogy történik. Ehhez iránymutatást a *meta modell* nyújt, amiben megadják a naplófájlok számára az alapvető követelményeket, vagyis azt, hogy milyen adatoknak kell feltétlenül rögzítve lenniük [3] [9].

2.4. Gráfok távolságának meghatározása

Számos alkalmazási területen szükséges a gráfok összehasonlítása és hasonlóságuk jellemzése valamilyen mérték szerint. Az irodalomban ismert lehetőségek közül ebben az alfejezetben a gráfok távolságának egy olyan értelmezését mutatjuk be, amely két gráf maximális közös részgráfján alapul [10]. Gráfok távolságának ilyen módon történő meghatározásakor csúcscímkezt gráfokat (Petri hálókat) alkalmazhatunk, így a távolság definiálásához szükséges alapfogalmakat is ilyen gráfokra értelmezzük. Megjegyezzük, hogy ezek a fogalmak analóg módon általánosíthatók csúcs- és élcímkézett gráfok esetén is.

Címkezt gráf

Egy csúcscímkezt G gráf a következő hármassal adható meg:

$$G = (V, E, \mu), \text{ ahol}$$

V : csúcsok véges halmaza

$E \subseteq V \times V$: élek halmaza

$\mu: V \rightarrow L_v$ csúcscímkezt függvény.

◇

A G gráfot $V = \emptyset$ esetén üres gráfnak nevezzük.

Részgráf

A $G_s = (V_s, E_s, \mu_s)$ gráfot a $G = (V, E, \mu)$ gráf részgráfjának nevezzük, ha $V_s \subseteq V$, $E_s = E \cap (V_s \times V_s)$ és $\forall v \in V_s$ esetén $\mu_s(v) = \mu(v)$.

Ebben az esetben a következő jelölést alkalmazzuk: $G_s \subseteq G$.

◇

Gráfizomorfizmus

Egy $f: V \rightarrow V'$ bijektív függvényt gráfizomorfizmusnak nevezünk

a $G=(V, E, \mu)$ gráfról a $G'=(V', E', \mu')$ gráfra, ha

- $(v_1, v_2) \in E \leftrightarrow (f(v_1), f(v_2)) \in E'$,
- $\mu(v) = \mu'(f(v)) \quad \forall v \in V$ -re.

Ha létezik gráfizomorfizmus G -ből G' -re, akkor a G és G' gráfokat izomorfoknak hívjuk. $\diamond$

Részgráf-izomorfizmus

Egy $f: V \rightarrow V'$ injektív függvényt részgráf-izomorfizmusnak hívunk G -ből G' -be,

ha létezik olyan $S \subseteq G'$ részgráf, hogy f gráfizomorfizmus G -ből S -be. $\diamond$

Két gráf közös részgráfja

A részgráf-izomorfizmus segítségével értelmezhetjük két gráf közös részgráfját az alábbi módon:

Legyenek G, G_1 és G_2 gráfok. A G gráf G_1 és G_2 gráfok közös részgráfja,

ha létezik részgráf-izomorfizmus G -ből G_1 -be és G -ből G_2 -be. $\diamond$

Maximális közös részgráf

A G gráfot maximális közös részgráfnak hívjuk, ha nincs G_1 -nek és G_2 -nek

olyan részgráfja, mely több csúcsot tartalmazna G -nél. G_1 és G_2 maximális közös

részgráfját $mcs(G_1, G_2)$ -vel jelöljük. $\diamond$

Megjegyezzük, hogy a maximális közös részgráf nem szükségképpen egyértelmű adott G_1 és G_2 esetén.

Két gráf közti távolság

Vezessük be a $|G|$ jelölést, amely a $G=(V, E)$ gráf csúcsainak számát jelöli.

Ekkor két nemüres gráf (G_1 és G_2) közti távolság az alábbi módon értelmezhető:

$$d(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{\max(|G_1|, |G_2|)}$$

$\diamond$

Bizonyítható, hogy a fenti d távolság kielégíti a mérték tulajdonságait [10].

Két gráf maximális közös részgráfjának klasszikus algoritmikus kiszámítási módja maximális klikk azonosításon [11] és visszalépésen [12] alapul.

Ezek az algoritmusok általában exponenciális idejűek. Az általunk összehasonlítani kívánt gráfok (Petri hálók) azonban címkézettek, a címkéző függvény injektív, továbbá az összehasonlítandó hálók speciális szerkezetűek, aminek következtében a maximális közös részgráf meghatározása jóval egyszerűbb probléma, mely polinomiális időben megoldható.

2.5. Modellezést, elemzést segítő szoftverek

A modellek megalkotásához számos, általában grafikus felületű szoftver áll rendelkezésre. Diagnosztikai vizsgálataink során a meglévő programokon kívül saját fejlesztésű szoftvert is használunk, melynek bemutatása a 3.1.5-ös fejezetben található. A következő rész a többi, általunk is használt modellezést és elemzést segítő programokról ad rövid áttekintést.

CPNTools

Munkánk során a technológiai rendszereket színezett Petri hálók segítségével modelleztük. A normál és a kiterjesztett referencia modellt leíró háló (3.1-es fejezet) megadását és működési módok szimulációját a CPNTools szoftver segítségével végeztük el. A CPNTools-t 2000-ben, a dániai Aarhus egyetemen fejlesztették ki, a Design/CPN elnevezésű szoftver továbbfejlesztett változataként. Széles körben elterjedt szoftverről van szó, ~ 10.000 kiadott licenccel a világ ~150 országában [13].

CPNTools legfontosabb, a modellezés és az elkészült modell elemzésére szolgáló jellemzői a következők:

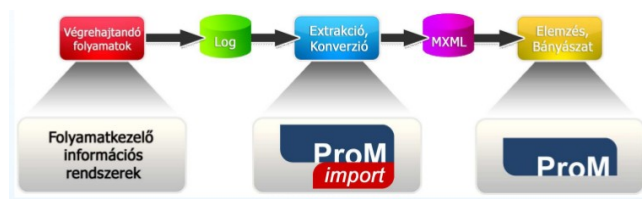
- egy és több szinten történő modellezés,
- időzített hálók létrehozása,
- felhasználó által irányított, lépésről-lépésre történő szimuláció,
- automatikus szimuláció,
- szimuláció közbeni eseménynaplózás lehetősége,
- a háló állapotterének elemzése,
- teljesítményanalízis.

A CPNTools-sal létrehozott modellekben található típusok, élkifejezések és őrfeltételek megadása egy erősen típusos, funkcionális programnyelven, a Standard ML-ben történhet. A Standard ML alapja a lambda-kalkulus, formális szintaktikával és szemantikával. E programnyelv további jellemzője, hogy tetszőlegesen bonyolult függvényeket és műveleteket lehet benne definiálni. Atomi (integer, string, boolean, felsorolás), strukturált (products, record, union, list és subsets) és összetett adattípusokat támogat. A standard ML összességében egy általános, tesztelt, számos irodalommal rendelkező modellező nyelv [6] [9].

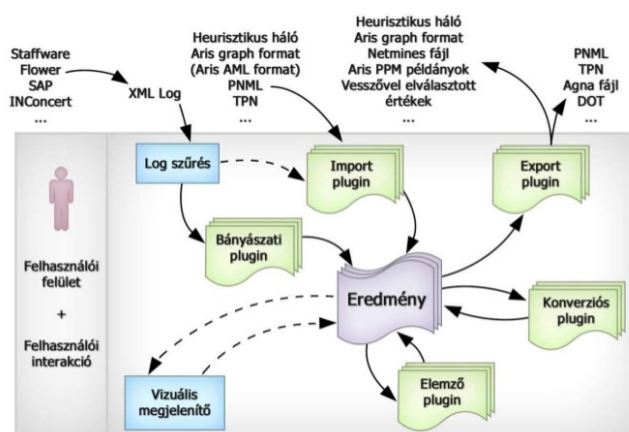
ProM

Munkánk során a rendelkezésre álló eseménynaplók elemzésére egy nyílt forráskódú folyamatbányászati eszközt, a ProM-et használtuk. Ez a szoftver közel 300 beépített pluginnal rendelkezik. Számos típuskonverzióra biztosít lehetőséget, támogatja legfontosabb folyamatmodellek importálását, valamint *.mxml formátumú eseménynaplók beolvasását. Képes a naplók irányított szűrésére, különféle elemzésekre, verifikálási feladatok megkönnyítésére. A folyamatbányászat fő lépései a 2.9-es ábrán láthatók, míg a ProM keretrendszer architektúráját a 2.10-es ábra szemlélteti.

A ProM lehetőséget ad arra is, hogy gráfokhoz log fájlokat rendeljünk, s számos szerkezeti, elérhetőségi, lefedési, statisztikai stb. elemzést végezzünk. A CPNToolsban létrehozott referencia modellből (3.1.1-es fejezet) megalkotott hibás működés lehetőségét is biztosító hálónk fájl típusa *.cpn kiterjesztésű, ami közvetlen beolvasása nem lehetséges a ProM adatbányászati szoftverbe. Ennek a problémának az áthidalására saját szoftvert fejlesztettünk, melyet a következő részben ismertetünk.



2.9. ábra. A folyamatbányászat fő lépései [9]



2.10. ábra. ProM architektúrája [8]

3. Saját eredmények

3.1. Hibadiagnosztika színezett Petri hálók segítségével

Egy megfelelő hibadiagnosztikai rendszer képes a vizsgált rendszer állapotainak minél szélesebb körű feltárására, ami által új lehetséges hibákra is rámutathat. A következőkben bemutatásra kerülő új eljárásunk diszkrét eseményű rendszerek modell-alapú diagnosztikájára alkalmazható [14] [15]. Ennek során a modellek megalkotása mindig egy hipotetikus, hibamentes működést leíró modellből indul ki, majd következő lépésben az ismert hibalehetőségek rendszerbe való integrálása történik meg. A diagnosztikához segítségünkre vannak a technológiai folyamatokból nyert információkat és a modell szimulációja során a különböző működési eseteket (normál, ismert és ismeretlen hibás) rögzítő ún. logfájlok. Az eseménynaplók felhasználásával újabb hálók felépítése, majd az összehasonlító elemzések matematikai úton történő elvégzése következik, melynek eredményeként következtetéseket tudunk levonni a hibák súlyosságára nézve. Ezen fájlok elemzésére adatbányászati, valamint saját fejlesztésű szoftvert használunk.

A következő alfejezetek bemutatják, hogy milyen lépéseken át jutottunk el végcélig, vagyis a hibák súlyosságának feltárásáig.

3.1.1 Normál referencia modell

A gyártórendszereknél ismert a műveletek helyes lejátszódási sorrendje és ezeknek a hibamentes műveleteknek a kapcsolata. Egy termelő rendszer esetén a hibamentes működés azt jelentené, hogy a gyártósor minden eleme kiszabott élettartama alatt hibamentes működésre lenne képes, vagyis sosem termelne selejtes terméket, soha nem állna le a működése, így összességében semmilyen veszteséget nem okozna. Nyilvánvaló, hogy ilyen rendszer nem létezik, így a cél az, hogy minél jobban megközelítsük a hibamentes állapotot, illetve a felmerülő hibákat kezelni tudjuk.

A hibadiagnosztikához vezető út első lépésként így létrehoztuk a gyártórendszer normál, hibamentes működését leíró színezett Petri háló modelljét, melyre normál referencia modellként hivatkozunk. A modellalkotás a 2.5-ös fejezetben ismertetett CPNTools szoftver segítségével történt.

A normál referencia modell megalkotásánál is érdemes számos modellezési elvet figyelembe venni. Ilyen például komplex működésű rendszerek esetén a modulokba való szervezés, azaz hierarchikus háló kialakítása, vagy az időzítés bevezetése. Az első lépés azonban a helyes absztrakciós szint meghatározása kell hogy legyen. Ennek során el kell dönteni, hogy mi a célja a modellnek, mely rendszertulajdonságokon van a hangsúly, és mik azok az elhagyható részek, amelyek nem befolyásolják a helyes következtetéseket. A színezett Petri hálókkal történő modellezés különböző absztrakciós szinteket támogat akár ugyanazon modellen belül is. A CPN magas absztrakciós szintje, tipikusan a tervezési fázis korai szakaszára jellemző, amikor lépésről-lépésre, a modellt fokozatosan finomítva készítünk el egy végrehajtható prototípust.

3.1.2 Hibás működés lehetőségét is biztosító Petri háló modellek

A rendszer normál referencia modelljének megalkotása után, az ismert hibák rendszerbe való integrálása következik, melynek eredményeképp különféle hibás működési helyzetek vizsgálata válik lehetségessé. Az így kapott új modellt kiterjesztett referencia modellnek nevezzük.

A rendszerben megjelenő hibák modellbe építése többféleképpen is történhet. A technológiai rendszer modelljének a megfigyelt adatokból történő visszaállítása esetén, a beépítendő ismert hibákat a log fájlok is tartalmazzák, így az egyik lehetőség, hogy az eseménynaplókból valamilyen algoritmus segítségével (pl. alfa algoritmus) állítjuk vissza a hálót [16]. Azonban az I. problémafelvetésben már említésre került, hogy a log fájlokból egy hálószerkezet visszaállítása közel sem triviális feladat. A visszaállítás során felesleges és hiányzó irányítások valamint redundanciák is megjelenhetnek, mivel a logokban általában nem kerül rögzítésre, hogy egy adott művelet mely események előzték meg, illetve követték. A ProM szoftver is támogatja a log fájlokból történő háló-visszaállítást, de gyakran az előbb említett problémákkal. Megoldásként viszont lehetőséget biztosít arra, hogy a felhasználó kézzel rendelje össze az egyes eseményeket megelőző és követő eseményeket, valamint a párhuzamosan lejátszódó részeket [16] [17]. A ProM az egyes hibákat külön eseményként építi be a modellbe.

Ha a technológiai rendszer vizsgálata a Petri háló alapú modell felépítésével történik, akkor a hibalehetőségek beépítése úgy is lehetséges, hogy a hibaeseményeknek megfelelő átmeneteket és hozzá tartozó helyeket vezetünk be a modellbe. Ekkor a modellezés a következőképpen történhet:

- normál működést leíró háló elkészítése,
- hibahelyek és/vagy hibaátmenetek beépítése.

A hibabeépítést célszerűen mindig megelőzi a normál referencia modell megalkotása, mivel az elemzések során szükség van a referencia- és a kiterjesztett hálóra is. Ha a hibákat helyként építjük be, akkor a szimuláció során a modellezőnek gondoskodnia kell arról, hogy ezek a hibahelyek a megfelelő tokent megkapják. Ez általában a szimulációba való kézi beavatkozást igényel. Ha a hibákat átmenetek segítségével modellezzük, akkor gyakran konkurencia helyzetet alakul ki a normál működést, illetve a hibás működést leíró átmenetek között, amit ugyancsak a modellezőnek kell kezelnie. Mindezek mellett a további helyeknek és átmeneteknek a beépítése növeli a háló bonyolultságát.

Munkánk során azonban ezektől a beépítési módoktól eltérően, a hibák modellbe való integrálását egy új, egyedi ötlet segítségével valósítottuk meg [18]. Ennek eredményeként sikerült további helyek és átmenetek beépítése nélkül, a normál referencia modell elemeihez tartozó inscription-ök („leírások”) módosításával beépíteni a hibákat. Ezek a leírások egyrészt az átmenetekhez rendelt ellenőrző függvények, melyek hibára vagy hibamentes működésre utaló értékkel térhetnek vissza, másrészt az átmenetek és helyek közt lévő irányított élek feltételes utasításai. Az, hogy melyik feltételes utasítás játszódik le, az a megelőző átmenethez rendelt ellenőrző függvény visszatérési értékétől (pl. token színétől) függ. A beépített függvények felelnek azért is, hogy egy rendszerleállást eredményező hiba esetén a háló tokeneloszlása a kezdeti tokeneloszlással egyezzen meg, vagyis inicializációs állapotba kerüljön a rendszer.

A hibák bekövetkezése véletlenszerű, előfordulásukat egy valószínűségi függvény irányítja. A hibák előfordulásának gyakoriságát a felhasználónak lehetősége van állítani. Ennek a gyakoriságnak a beállításánál pl. a meglévő log fájlokból adatbányászati elemzések log fájl segítségével kapott hibastatisztikák lehetnek segítségünkre.

Összegezve elmondható, hogy ezen új módszer részeként kifejezések segítségével modell kiegészítést alkalmaztunk anélkül, hogy valamilyen hálószerkezeti módosítás történt volna, így a hibák beépítése nem okozott szerkezeti komplexitás növekedést. A hibaszimuláció konkrét megvalósítását az esettanulmányt tartalmazó *3.2-es fejezetben* mutatjuk be.

3.1.3 Naplóállományok és felhasználásuk

Mind a normál, mind a kiterjesztett referencia modell létrehozásánál fontos szempont volt, hogy az elkészített hálók szimulációra alkalmasak legyenek, és a lehetséges működési módokat könnyen nyomon tudjuk követni. A CPNTool-ban készített modell szimulációs futtatása során, a lejátszódó eseményekről további információk (az eseményt végrehajtó személye, esemény típusa (normál/hibás), lejátszódás időpontja, stb.) folyamatonként külön trace fájlokba mentésre kerülnek, melynek eredményeképp „mesterséges” log fájlokhoz jutunk. A következő alfejezet ezen log és trace fájlokról ad áttekintést.

Logok és tracek

Az intelligens irányítási rendszerek működésük során monitorozó tevékenységet is végeznek, eseménynaplókba rögzítve az egyes beavatkozásokat illetve a megfigyelt mérési adatokat. Ezek az eseménynaplók más néven log fájlok tehát a technológiai rendszerbeli események véges sorozatát írják le, s az egyes eseményekhez rendelt információkat tartalmazzák. Hasonló log fájlokat kaphatunk, ha a modellező rendszerünkben végzett szimuláció során az egyes lépésekhez tartozó adatokat rögzítjük. Amennyiben a technológia rendszer működése ciklikus, azaz ugyanazt a műveletsort többször egymás után kerül végrehajtásra, akkor az egy ciklus adatait tartalmazó fájl trace-nek, míg az üzemeltetés során kapott tracek halmazát lognak nevezzük. A rendelkezésünkre álló trace illetve log fájloknak az elemzése adat- és folyamatbányászati (*2.5-ös fejezet*), valamint saját fejlesztésű szoftverrel (*3.1.5-ös fejezet*) történt.

Log fájlok értelmezése és felhasználása

A következőkben nézzük meg, hogy az CPNToolsban megalkotott modelljeink szimulálásakor kapott log fájlok és a valós technológiai folyamatok eseményfájljai milyen jelentőséggel rendelkeznek. Célunk, hogy ezen trace és log fájlok segítségével diagnosztikai vizsgálatokat végezzünk a rendszer működésének helyességét illetően, és meghatározzuk az előforduló hibák típusát és számát.

A CPNTools *.cpnxml kiterjesztésű trace fájlokat hoz létre. Egy trace-ben megtalálhatjuk, hogy egy adott gyártási számú termék milyen műveletsorokon haladt keresztül, kik vagy milyen gépek dolgoztak rajta stb. Azonban, ha a gyártórendszerünk teljes működéséről szeretnénk képet kapni, akkor az összes ilyen trace fájl együtt kell elemezni, mivel ezek együttesen tartalmazzák az összes legyártott termék információit. Az egyes trace fájlok egy *.mxml kiterjesztésű log fájlba való importálása a ProMimport [19], míg a végső kiértékelésük a ProM [20] adatbányászati szoftver segítségével történt.

3.1.4 Gráfszerkezeti vizsgálatok

Jelölje G_n a normál, G_f ($f = 1, 2, \dots$) pedig a lehetséges működési módokat leíró hálókat. A lehetséges működési mód egyaránt megfelelhet a normál működésnek, illetve az egy vagy több hibát tartalmazó hibás működési módnak. A gráfszerkezeti vizsgálatok során a normál referencia modell hálóját (G_n) hasonlítottuk össze mindig az adott tényleges működést leíró G_f hálóval. Ennek első lépéseként meg kellett határozni a G_n és G_f hálók maximális közös részgráfját.

A gráfizomorfizmus bonyolultságának eldöntése máig nyitott kérdés (sem polinomiális sem NP teljességét nem sikerült még bizonyítani), azonban a részgráf-izomorfizmus meghatározása bizonyítottan, általánosan NP teljes probléma. Az általunk összehasonlítani kívánt hálók címkézettek, címkéző függvényük injektív, így a maximális közös részgráf meghatározása algoritmikusan lényegesen egyszerűbb probléma. A gráftávolság számításához a 2.4-es fejezet két gráf közötti távolságot megadó definíciójában ismertetett képlet szerint, három gráf ($G_n, G_f, mcs(G_n, G_f)$) csúcsainak számára van szükség.

A különböző lehetséges működési módokat a normál működéssel úgy hasonlítjuk össze, hogy a megfelelő hálók távolságát a fentiek szerint meghatározzuk. A gráftávolság értéke alapján következtethetünk a normál működéstől való eltérés mértékére, azaz a hiba súlyosságára. Minél nagyobb egy számolt érték, annál súlyosabb hiba következett be. Mindezek gyakorlati példán történő bemutatása az esettanulmány 3.2.3-as részében olvasható.

3.1.5 Converter

Ahhoz, hogy a CPNToolsban létrehozott gráfokból kiindulva saját, és a ProM által is támogatott elemzéseket lehessen végezni szükségessé vált egy olyan szoftver kifejlesztése, amely az egyéni elemzési funkciókon kívül a két program közötti típuskonverziót is megoldja.

A színezett Petri háló alakú modelljeink megalkotásánál egy olyan technika kifejlesztése volt a cél, ami komplex rendszerek esetén is átlátható hálókat hoz létre, így a CPNTools környezetben elkészített modelljeink hierarchikus Petri hálók. A gráfok szerkezeti analízise során, azonban már egyszintű hálókkal célszerű dolgozni, hogy a teljes szerkezetet egyben kezelhessük. Ebből adódott programunk egyik fő feladata, a hierarchikus hálóként felépített modell egyszintű hálóvá történő alakítása. Az egy szintre hozás során vizsgálni kell az egyes átmenetek típusát, hogy kiszűrjük melyek azok amik alhálóba vezetnek, valamint szükség van az egyes modulokban található ki és bemeneti helyek összerendelésére, a modell legfelső szintjén található logikai kapcsolatok szerint.

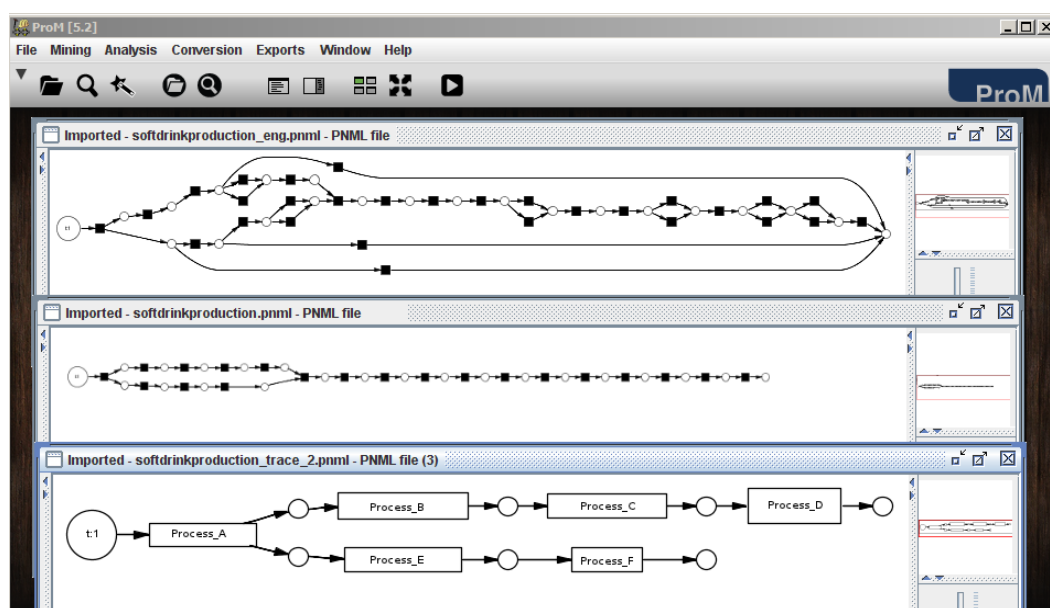
A normál referencia modellből (3.1.1-es fejezet) kiindulva az ismert hibalehetőségek integrálása az átmenetekhez és élekhez rendelt speciális kifejezésekkel történt. Ezen kifejezések értelmezésére azonban csak a színezett Petri hálónk saját környezetében lehetséges. Ennek megfelelően az elkészített szoftverünk képes arra, hogy ezeket a kifejezéseket a generált új hálóba átültesse oly módon, hogy szükség szerint egyértelmű hibaátmenetek kerülnek beépítésre a modellbe.

Programunk további fontos eleme a modellháló mellé társítható eseménynaplók feldolgozása. Az eseménynaplók az említett módon (3.1.3-as fejezet) egy vagy több lefutás eredményét tartalmazhatják. A ciklusok számától függetlenül a naplók elemzésének elvégzése után, szoftverünk által megjelenítésre kerülnek az egyes technológiai lépések során történt hibák a megfelelő műveletekhez társítva, azonosító nevük és súlyosságuk megjelölésével. A listázás után, a felhasználónak lehetősége van azon hibák kiválasztására, melyeknek együttes

hatását vizsgálni szeretné. Ennek eredményeként Converterünk lehetőséget biztosít a különféle hibamódokat leíró kiterjesztett referencia modellek (3.1.2-es fejezet) hálóinak létrehozására, melyek az általunk ajánlott hibadiagnosztikai eljárás fontos elemei. A kiválasztott elemek a normál referencia modell hálójába kerülnek beépítésre létrehozva ezáltal egy kiterjesztett modellt, melynek formátuma a ProM szoftverrel kompatibilis, így további elemzéseket tesz lehetővé (3.1-es ábra).

A felhasználó által kiválasztott hibák modellbe építésénél külön figyelmet kellett fordítani az egyes hibák hatásaira. Így egy, a technológiai rendszer leállását eredményező hiba esetén (ami után az egész műveletsort az elejéről kell újraindítani) a hálóban előreccatolás szükséges a leállási állapotot szimbolizáló helyre. Azonban csak a termék minőségét befolyásoló, de a további működést nem akadályozó hiba esetén egy, a hibás működésnek megfelelő átmenetet kell beépíteni.

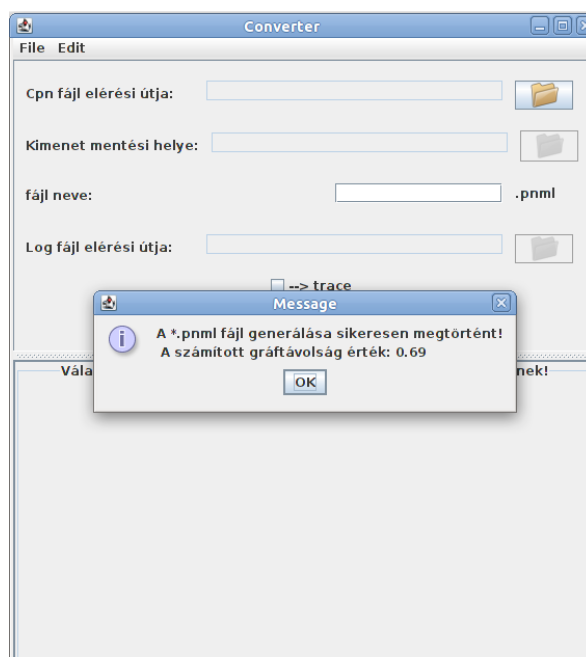
Ha azonban adott logfájlt szeretnénk elemezni és a benne rögzített hibák rendszerre gyakorolt hatásának súlyosságát vizsgálni, akkor a visszaállított gráf csak a log fájlban rögzített eseményekből épül fel, vagyis egy normál vagy hibás esemény bekövetkezése itt már nem opcionális mint a kiterjesztett modellnél. Eredményként így a tényleges működést leíró hálót kapjuk (3.1-es c ábra).



3.1. ábra. Visszaállított hálók;

a. Felhasználó által kiválasztott hibalehetőségeket is tartalmazó háló; b. Normál működést leíró háló; c. Trace-ből visszaállított tényleges működést leíró háló

Miután rendelkezésünkre állnak a Converter programunk segítségével átalakított és a felhasználó által kiválasztott hibamódokat is leíró Petri hálók, a gráf-összehasonlító vizsgálatok következnek. Ennek során gráftávolsági értékek számítása történik, a normál működést leíró referencia modell és valamelyik visszaállított háló maximális közös részgráfját használva (*matematikai ismertetését lásd 2.4-es és 3.1.4-es fejezetek*). A gráftávolság értékek számítása minden trace fájl konverzió során automatikusan is megtörténik (*3.2-es ábra*).



3.2. ábra. Converter programunk;
*.pnml fájl generálása és a gráfok távolságának
számítása

A logfájlok és a hálók szerkezetének leírására számos lehetőség van:

1. A normál referencia modell megjelenítésének lépései:
 - a) Converter programunkba olvassuk be a CP modellünkhöz tartozó *.cpn fájlt és végeztessük el a konverziót.
 - b) Importáljuk a kapott *.pnml fájlt a ProM-be. Ekkor a ProM megjeleníti a rendszer normál működésének egyszintű hálóját.
2. Adott hibás események bekövetkezésének lehetőségét biztosító Petri hálók megjelenítéskor a következő lépéseket kell elvégezni:
 - a) Integráljuk a korábban ismertetett új módszerünk segítségével a hibákat a referencia modellbe, majd végezzünk tetszőleges számú szimulációt.
 - b) A CPNTools által az egyes szimulációk során keletkezett trace fájlokat, a ProMImport [21] segítségével importáljuk egyetlen log fájlba.
 - c) A Converter programnak adjuk meg a CP modellünket és olvassuk be vele a modellhez tartozó log fájlt. A beolvasás után megjelenő listából válasszuk ki a lehetséges hibák közül azokat, amelyeket meg akarunk jeleníteni a modellünkben, majd végeztessük el a konverziót.
 - d) A kapott fájlt olvassuk be a ProM-be, így eredményeként a kiválasztott hibás működési módok lehetőségét is tartalmazó hálót kapunk.

3. Egy trace fájl megjelenítésének lehetséges módja:
- a) Integráljuk a korábban ismertetett új módszerünk segítségével a hibákat a referencia modellbe, majd egyszer futtassuk le a szimulációt.
 - b) Konvertáljuk a kapott trace fájlt a ProMImport segítségével.
 - c) A Converter programnak adjuk meg a CP modellünket és olvassuk be vele a modellhez tartozó log fájlt. A beolvasás után megjelenő listából válasszuk ki a lehetséges hibák közül azokat, amelyeket meg akarunk jeleníteni a modellünkben, majd végeztessük el a konverziót.
 - d) A kapott fájlt importáljuk a ProM-be, így megjelenik a gráfunk.

3.2. Esettanulmány

Ebben a fejezetben egy konkrét példán kerülnek bemutatásra a modellezés folyamatai, a megvalósított hibaszimuláció, a gráf-összehasonlító elemzések és az általunk használt gráftávolságot meghatározó módszer. A példaként választott üdítőt gyártó és palackozó technológiai rendszer modellezése során csak a főbb műveleteket integráltuk a modellbe, nem térünk ki minden lehetőséget lefedő részletekre. A kapott modell azonban a módszer szemléltetéséhez és eredményességének ellenőrzéséhez kellő részletességgel rendelkezik.

Mint korábban ismertetésre került, első lépésként el kell készíteni a rendszer normál, hibamentes működését leíró modelljét (célszerűen hierarchikus színezett Petri háló formában). Létrehozott modellünkben a gyártórendszer egyes részműveleteit külön alhálók tartalmazzák, amelyek a háló felső szintjén kerültek összekapcsolásra. Ennek eredményeként modellünk a hierarchikus Petri hálók csoportjába tartozik.

3.2.1 Üdítőgyártó modell normál működésének rövid áttekintése

Az általunk modellezett üdítőgyártó rendszer fő műveletei a következők:

- a beérkező üvegek mosása,
- az üdítő kikeverése szirupból és vízből,
- a kész keverék üvegekbe töltése és karbonizálása,
- az üvegek zárása és címkézése.

Ezek a főbb műveletek további lépésekre bonthatók, melyek részletezése és a helyszínekre történő besorolás az *1. táblázatban* leírtaknak megfelelően az alábbi:

Mosóterem

A gyárba beérkezett üres üvegpalackokat töltésük előtt tisztítani kell, így első lépésként a mosó terembe kerülnek, ahol az öblítő gépbe helyezik őket. A gép a palackokat fejre állítja, nagy nyomású vízzel átmossa, majd talpra állítja.

Keverő terem

A mosótermi műveletsorral párhuzamosan a keverő teremben megtörténik a szűrt víz, majd a szirup tartályba való betöltése, végezetül pedig ezek elegyítése.

Töltő- és karbonizáló sor

Az átmosott üvegek következő útja a töltősor karbonizáló részébe vezet, ahol befecskendezéssel szénsavat juttatnak a palackokba, majd a palackozó gép tárolójába továbbítódnak. Ezután a levegő eltávolításra kerül belőlük, majd betöltik az üdítőt. Az üdítővel teli, de még lezáratlan üvegek ezután átkerülnek a kupakozó és címkéző gépbe.

Kupakozó és címkéző gépsor

A kupakozó gép folyamatai a kupakok érkezésével kezdődnek. Itt történik a kupakok üvegekre helyezése, majd rácsavarása. A címkézési folyamat első lépéseként az üvegekre hideg ragasztót ken a gép, majd ráilleszti a címkéket, amit kefék segítségével az üveghez simít.

Start	Mosó terem
palackok érkezése	palackok öblítő gépbe helyezése
	palackok fejre állítása
	mosás
	palackok talpra állítása
Keverő terem	Kupakozó és címkéző terem
szűrt víz betöltése	kupakok érkezése
szirup betöltése	kupak illesztése
elegyítés	kupakok meghúzása
	hideg ragasztó felvitele
	címke illesztése
	címke simítása
Töltősor, karbonizáló terem	
szénsavasítás befecskendezéssel	
palackozó gép tárolójába továbbítás	
levegő eltávolítás	
üdítő betöltése	

1. táblázat. Műveletek lehetséges csoportosítási módja

Ahogy a normál működést leíró háló felső szintjén (Melléklet 5.1. ábra) látható, a mosó- és keverő terem műveletei egymással párhuzamosan játszódnak le, majd ezek befejeztével a töltő- és karbonizáló sor, majd a kupakozó és címkéző terem folyamatai kerülnek végrehajtásra.

Végrehajtók

Az egyes műveletek végrehajtója lehet ember és gép is. Modellezett rendszerünkben ilyen végrehajtó lesz pl. a beszállító, a minőségellenőr, vagy a gépek közül a mosást, keverést, elegyítést, karbonizálást, palackozást végző gép (Melléklet 5. táblázat). Az ismert végrehajtók műveletekhez rendelve az egyes log fájlokba mentésre kerülnek (3.3-as ábra).

3.2.2 Hibák modellbe való integrálása

A normál működés modellezése után a következő célunk a lehetséges hibák és azok bekövetkezési valószínűségének modellbe történő integrálása. Ennek során a háló felső szintjére a bekövetkezett hibákat kezelő átmenetek kerültek (Melléklet 5.2-es ábra), a részhálóknak pedig az átmenetekhez és az irányított élekhez bővített leírásokat (inscription) rendeltünk. Az átmenetek leírásai ellenőrző függvényeket tartalmaznak, amelyek hibának vagy normál végrehajtásnak megfelelő értékkel térhetnek vissza. Az irányítások feltételes utasítások (Melléklet 5.3-as ábra). Ezzel az általunk javasolt megoldással a hibás működés lehetőségét is tartalmazó rendszert úgy tudtuk reprezentálni, hogy a modell felső szintje kivételével nem kellett újabb hálóelemet beépítenünk. A modell felső szintje (így az ott megjelenő hibaátmenetek) a részhálók összekapcsolásakor, vagyis a teljes háló egy szerkezetként a ProM-ben való megjelenítésekor láthatatlan marad. A felső szinten lévő logikai kapcsolatok a részhálóknak kerülnek beépítésre.

Hibaosztályok

Korábbiakban említett módon a rendszerek működése során különféle hibák léphetnek fel. Ezen hibákat célszerű súlyosságuknak megfelelően osztályokba sorolni, ezzel is növelve a hatásvizsgálat hatékonyságát. Modellünkben a normál működés mellett három hibaosztályt különítettünk el. Ezek közül a legsúlyosabb az úgynevezett *leállást eredményező hibák* csoportja, amelyek fellépte esetén azonnal megszakad a gyártási folyamat, és az inicializációs állapotába áll vissza a rendszer. A következő hibaosztály neve *selejtet eredményező hiba*. Ide azok a hibák tartoznak, amelyek bekövetkezésükkor selejtes terméket gyártását okozzák. Ilyen eset áll fenn akkor, ha a mosás valamilyen ok miatt nem elég alaposan történik meg, így szennyeződés marad az üvegben, amibe az üdítő kerül. A legkisebb súlyú *kezelhető hiba* nevű hibaosztályba azok a folyamatok tartoznak, amelyek lejátszódása a termelési folyamatot nem befolyásolja, azonban valamilyen plusz munkát, kiadást okoz, de a termék kedvezményes áron még értékesíthető marad. Erre példa az üdítő alul ill. túltöltése, vagy a címke fordítva történő felragasztása. A modellezett folyamatunk egyes műveleteihez tartozó hibákat és azok hibaosztályát a 2. táblázat foglalja össze.

Látható, hogy ezek a hibaosztályok valós rendszerek esetében is jól kezelhetők, mivel egy, a normál működéstől eltérő esemény lejátszódása után ismert a következmény azaz tudjuk milyen beavatkozások szükségesek.

Ha azonban olyan hiba fordul elő a valós technológiai rendszerben, ami előre nem ismert, akkor a logfájlokból kell következtetni arra, hogy melyik hibaosztályba tartozik. Nyilvánvaló, hogy a modellező által készített modellben csak azok a hibák fordulhatnak elő, amelyeknek a beépítése a modellalkotás során megtörtént. Vizsgálataink során a valós működést a színezett Petri háló szimulációja jelentette, a működés elemzését pedig részben *Converter*ünk és a ProM szoftver, részben a maximális közös gráfon alapuló számítások segítségével végeztük el. A szimuláció és az elemzés összekapcsolását az általunk készített *Converter* program teszi lehetővé. Mint ismertettük, ez a program építi be a hibahelyeket is a hálóba. Ha ismeretlen hibát akarunk szimulálni, akkor ezt egyrészt a technológia rendszernek megfelelő Petri háló, másrészt a *Converter* program paramétereinek módosításával érhetjük el.

Művelet	Hiba	Hibaosztály
Mosás	Elégtelen víznyomás	Selejtet eredményező hiba
Mosás	Szennyeződés	Leállást eredményező hiba
Szűrt víz betöltése	Nincs víz	Leállást eredményező hiba
Szirup betöltése	Nincs szirup	Leállást eredményező hiba
Elegyítés	Nem megfelelő arány	Selejtet eredményező hiba
Üdítő betöltése	Nem megfelelő mennyiség	Kezelhető hiba
Kupak illesztése	Hibás kupakillesztés	Selejtet eredményező hiba
Hideg ragasztó felvitele	Ragasztási hiba	Kezelhető hiba
Címke illesztése	Hibás címke	Kezelhető hiba

2. táblázat. Üdítőgyártás modelljének hibaosztályai

3.2.3 Elemzések

A következő részben bemutatásra kerül, hogy a felépített modellek szimulációjától indulva mely lépések vezetnek a hibák súlyosságának meghatározásáig.

Szimuláció és konvertálás

Az előző fejezetekben bemutattuk a normál és a kiterjesztett referencia modellek létrehozásának folyamatát, melyek után a kapott modell szimulációja következik. A szimuláció során célunk egyrészt a háló és így a modellezett rendszer működésének dinamikai vizsgálata, másrészt log fájlok előállítás a ProM-ben végzendő elemzési feladatokhoz és a hibák gráftávolság alapú számításokkal történő diagnosztizálásához.

A vizsgálatok megkezdéséhez tehát szükségünk van az egyes működési ciklusokat leíró trace fájlokra (3.3-as ábra). Ezek a fájlok a modell szimulációjakor a modellező által megadott könyvtárba kerülnek elmentésre. A minél több adat elérése érdekében, automata szimulálást választva, 400 ciklust futtattam le, ami megfelel ugyanilyen számú palack üdítő ital elkészítésének. A hibák valószínűségét úgy állítottam be, hogy valamennyi hiba nagy valószínűséggel bekövetkezzon ilyen számú futtatás esetén. A szimuláció során így 400 trace fájl keletkezik, amit a PromImport segédprogram segítségével lehet egy log fájlba összegyűjteni.

```
<AuditTrailEntry>
<WorkflowModelElement>Palackok erkezes</WorkflowModelElement>
<EventType>normal</EventType>
<Timestamp>2011-01-01T01:06:00.000+01:00</Timestamp>
<Originator>Beszallito</Originator>
</AuditTrailEntry>
<AuditTrailEntry>
<WorkflowModelElement>Szurt viz betoltese</WorkflowModelElement>
<EventType>normal</EventType>
<Timestamp>2011-01-01T01:06:30.000+01:00</Timestamp>
<Originator>Keveres vezeto</Originator>
</AuditTrailEntry>
```

3.3. ábra. Részlet egy *.cpnxml trace fájlból

A korábban ismertetett módon, a színezett Petri háló modell struktúráját tartalmazó fájl, valamint a generált log fájl felhasználásával a modellező igényei szerinti a ProM adatbányász szoftvernek megfelelő formátumú háló generálása a Converter programunk segítségével történhet. A Converternek az üdítőgyártó példánk modelljének elérési helyét megadva, és a hozzá tartozó szimuláció során keletkező trace fájlokból felépülő log fájlt beolvasatva, listázásra kerülnek az előforduló hibák, műveletekhez kapcsolva hibaosztályuk megjelölésével. Ezen listából egy felhasználó által kiválasztott tetszőleges hibákat a 3.4-es ábra mutatja.

A felhasználónak a trace CheckBoxot kijelölésével van lehetősége egy trace fájlból visszaállított, valós működést leíró háló létrehozására (lásd 3.1-es ábra c része) és ezen háló valamint a normál referencia modell hálójának maximális közös részgráfok távolságán alapuló gráftávolságának kiszámoltatására (lásd 3.2-es ábra).

Cpn fájl elérési útja: C:\Uditogyartas\uditogyartas.cpn

Kimenet mentési helye: C:\Uditogyartas/

fájl neve: uditogyartas .pnml

Log fájl elérési útja: C:\Uditogyartas\cpnToolsSimulationLog.mxml

☐ --> trace

Konvertál

Válassza ki azokat a hibákat, amelyek a modelben szerepeljenek!

☐ Ragasztási hiba	Hideg ragasztó felvétele	hibas
☒ Szennyezodes	Mosas	selejtet eredményező hiba
☐ Hibas címke	Címke illesztése	hibas
☒ Nem megfelelő mennyiség	Udito betöltése	hibas
☐ Nincs szirup	Szirup betöltése	leallast eredményező hiba
☐ Nem megfelelő arány	Elegyítés	selejtet eredményező hiba
☐ Elegenden víznyomás	Mosas	leallast eredményező hiba
☒ Hibas kupakillesztes	Kupak illesztése	selejtet eredményező hiba
☐ Nincs víz	Szurt víz betöltése	leallast eredményező hiba

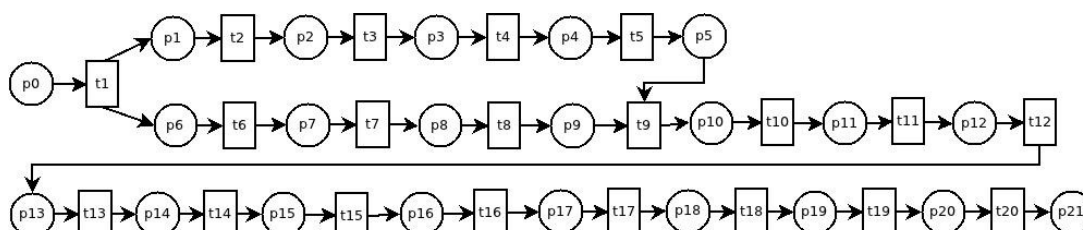
3.4. ábra. Converter program felhasználói felülete a kiválasztott log fájl beolvastatása után

Hibás működések elemzése

Az elkészített hálók megjelenítése (Melléklet 5.4.ábra) és az eseménynapló elemzése a ProM feladata volt. A ProM log fájlokból nyert összefoglaló és statisztikai elemzéseire példa a Melléklet 5.5-5.9.ábráin látható.

A lehetséges lefutási módok közül példaként a következő három esetet mutatjuk be:

- a műveleti folyamatok elején leállást eredményező hiba történt,
- a rendszer működése során egyetlen egy helyen kezelhető hiba következett be,
- a műveleti folyamatok során rendre egy kezelhető hiba, majd a műveleti folyamatok végéhez közeledve egy leállást eredményező hiba is történt.

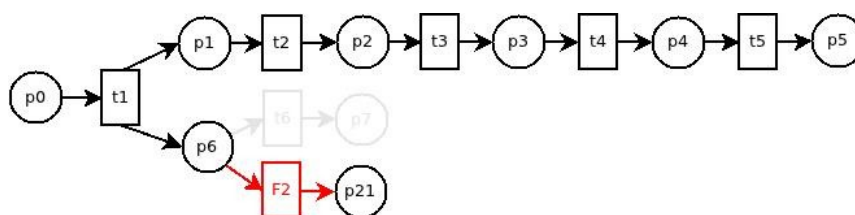


3.5. ábra. Normál referencia modell

A normál referencia modell (3.5-ös ábra) és a kiválasztott lefutások Petri háló modelljei a 3.5-3.8. ábrákon láthatók. A 3.5-3.8. ábrák t1-t20-as átmeneteihez tartozó műveletek az üdítőgyártás egyes műveleteinek felelnek meg a 3.táblázatban foglaltak szerint.

Átmenet címkéje	Átmenethez hozzá tartozó művelet megnevezése	Átmenet címkéje	Átmenethez hozzá tartozó művelet megnevezése
t1	Palackok érkezése	t11	Palackozó gép tárolójába továbbítás
t2	Palackok öblítő gépbe helyezése	t12	Levegő eltávolítása
t3	Palackok fejre állítása	t13	Üdítő betöltése
t4	Palackok mosása	t14	Kupakozó gépbe szállítás
t5	Palackok talpra állítása	t15	Kupakok érkezése
t6	Szűrt víz betöltése	t16	Kupakok illesztése
t7	Szirup betöltése	t17	Kupakok meghúzása
t8	Elegyítés	t18	Hideg ragasztó felvitele
t9	Töltősorra szállítás	t19	Címke illesztése
t10	Szénsavasítás befecskendezéssel	t20	Címke simítása

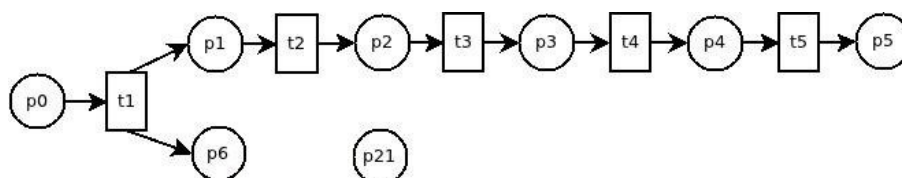
3. táblázat. Átmenet címkék és a hozzájuk tartozó műveletek megnevezései



3.6. ábra. A műveleti folyamatok elején leállást eredményező hiba

Az első esetben (3.6-os ábra) a t1-es esemény után a folyamat egyik párhuzamos ágában, a t6-os hibamentes működés helyett az F2-es hibás működés („nincs víz”) játszódik le. Mint ahogy a 3.4-es ábrán is látható, a „nincs víz” F2-es esemény egy leállást eredményező hibaosztály eleme, így ezen hiba bekövetkezésekor a rendszer működése leáll. Fontos megjegyezni, hogy ez a súlyos hiba a technológiai folyamat elején történt. E hibás működés gráfja (3.6.ábra) és a normál működést leíró gráf (3.5.ábra) maximális közös részgráfja a 3.7.ábrán látható. A maximális közös részgráfon alapuló gráftávolságot számolva a következőt kapjuk:

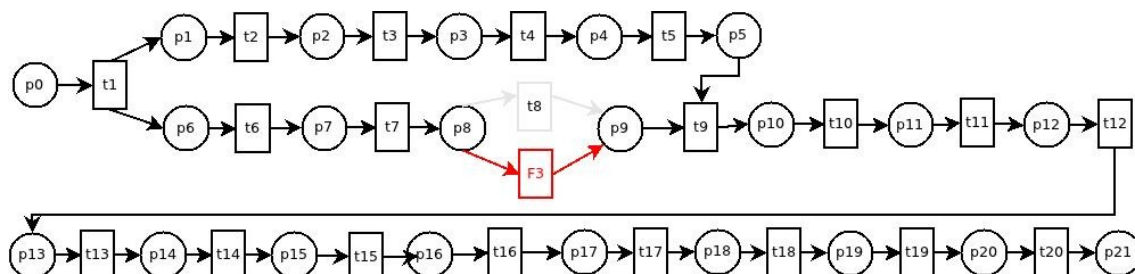
$$d(G_1, G_n) = 1 - \frac{13}{42} = 0.69$$



3.7. ábra. a 3.5-ös és 3.6-os ábrán látható gráfok maximális közös részgráfja

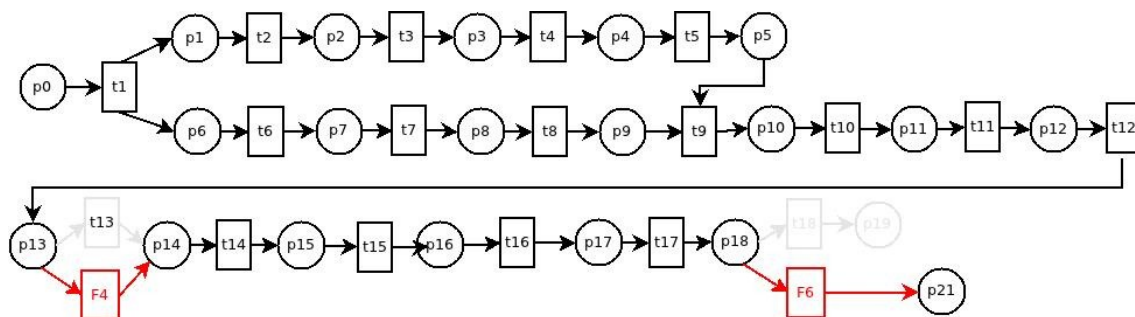
Második esetben (3.8-es ábra) egy kezelhető hiba következett be. Itt a t8-as normál működés helyett, az F3-as hibás működés („a betöltött szirup és víz aránya nem megfelelő”) játszódott le, azonban a technológiai folyamat nem került megszakításra, mivel a megfelelő arányok ellenőrzése csak a késztermékek minőségi ellenőrzésénél történik. Az ehhez az esethez tartozó gráfot a normál működés grádjával az előző esethez hasonló módon összevetve, a gráftávolságra 0.024-es értéket kapunk. Látható, hogy ez az érték jóval kisebb, mint az első esetben számolt gráftávolság.

Az általunk vizsgált hibalehetőségek körében a „kezelhető” és a „selejtet eredményező” hibaosztályba eső hibák mindegyike olyan, hogy a ProM-ben kapott Petri hálóját a normál működés hálójától csupán egy átmenetben tér el. Így a legnagyobb közös részgráf egyetlen csúccsal tartalmaz csak kevesebb pontot, mint a normál működés hálóját. Ezért minden olyan működésnél, ahol csak egy kezelhető vagy selejtet eredményező hiba játszódik le, az összehasonlítás során mindig ugyanazt a 0.024-es gráftávolságot kapjuk.



3.8. ábra. A rendszer működése során egyetlen egy helyen kezelhető hiba bekövetkezése

Utolsó esetként vizsgáljunk meg egy leállást nem eredményező, majd egy leállást eredményező hibát tartalmazó működést. A 3.9-es ábrán látható módon, a t13-as eset helyett az F4-es kezelhető hibát tartalmazó esemény (nem megfelelő mennyiségű üdítő került betöltésre) megy végbe, majd a t18-as normál működés helyett az F6-os leállást eredményező hiba (nincs ragasztó) következik be. Megfigyelhető, hogy az F6-os hiba után a rendszer azonnal végállapotába kerül, így az utolsó két esemény (t19, t20), már nem játszódik le. Ez esetben a számolt távolság 0.143, ami nem olyan magas mint az első vizsgált esetben, mivel a technológiai folyamatot megszakító hiba annak végén következett be.



3.9. ábra. A műveleti folyamatok során rendre egy kezelhető hiba, majd a műveleti folyamatok végéhez közeledve egy leállást eredményező hiba bekövetkezése

Az egyes esetekhez tartozó számolt értékeket a 4. táblázat foglalja össze.

Előforduló hibatípus	Számolt érték
Azonnali leállást eredményező hiba a folyamat elején	$d_1 = 0,69$
Leállást <u>nem</u> eredményező kis hiba	$d_2 = 0,024$
Leállást <u>nem</u> eredményező, majd egy leállást eredményező hiba a folyamat végén	$d_2 < 0,143 < d_1$

4. táblázat. Kiválasztott hibaesemények és a hozzájuk tartozó maximális közös részgráfok alapján számolt gráftávolság értékei

A számolt gráftávolságok alapján az alábbi következtetések vonhatóak le:

- Ha a technológiai folyamatban csak egy kezelhető hiba következik be, akkor a kapott gráftávolság kicsi, jelezve, hogy a működés csak kevéssé tért el a normálistól.
- Leállást eredményező hibák esetében nagyobb gráftávolság értékeket kapunk. Minél korábban következett be a leállást okozó hiba, annál közelebb lesz a kapott érték 1-hez. Ez megfelelhet a hiba gyakorlati szempontból minősíthető súlyosságának, ha egy, a folyamat elején leállást okozó hiba súlyosabbnak tekinthető, mint a technológiai folyamat vége felé bekövetkező leállás.

Mindezek alapján megállapítható, hogy a számított gráftávolságok alkalmasak a hiba súlyosságának jellemzésére.

4. Összegzés

TDK dolgozatomban a technológiai rendszerek diagnosztikai vizsgálatára egy új, általunk kialakított modell-alapú hibadetektáló eljárást mutattam be. Munkánk során a modellek felépítését színezett Petri hálókkal végeztük, mivel a technológiai rendszerek széles köre modellezhető ezzel a diszkrét idejű és eseményű rendszerek leírására alkalmas eszközzel.

A dolgozat első részében áttekintést adtam az alacsony és magas szintű Petri hálókról, a folyamatbányászat alapjairól, valamint két gráf maximális közös részgráfjának alapján értelmezhető gráftávolságról. Bemutattam az ajánlott modell-alapú hibadiagnosztika lépéseit, a korábban alkalmazott, és az újonnan javasolt hibamodellezési lehetőségeket.

A vizsgálatok során a modellbe a normál működést leíró Petri hálóból kiindulva beépítettük az ismert, lehetséges hibákat oly módon, hogy az alhálókból nem történt szerkezeti változás. A szimulációk során kapott vagy a technológiai rendszerek működésekor nyert log fájlokból és a referencia modellből, saját fejlesztésű szoftverünket felhasználva előállítottunk szerkezetileg is jól elemezhető Petri háló modelleket, melyek gráftávolságát is szoftverünkkel számoltattuk. A kapott modellek további értékelése a hozzárendelt log fájlokkal együtt egy adatbányászati szoftver (ProM) segítségével történt. A hibák mértékére a referencia modellt leíró háló és a log/trace fájlból előállított háló gráftávolsága alapján következtetéseket tudunk levonni.

Az eljárást, a szimuláció lépéseit, a logfájlok konverzióját, az adatok feldolgozását, a maximális közös részgráfon alapuló távolság meghatározását és a hibák súlyosságának értékelését egy technológiai példán mutattam be.

A kutatás lehetséges folytatása, a bemutatásra került modell-alapú hibadetektáló eljárás olyan irányú továbbfejlesztése, amely a rendszerben bekövetkező ismeretlen hibákat is hatékonyan képes felderíteni.

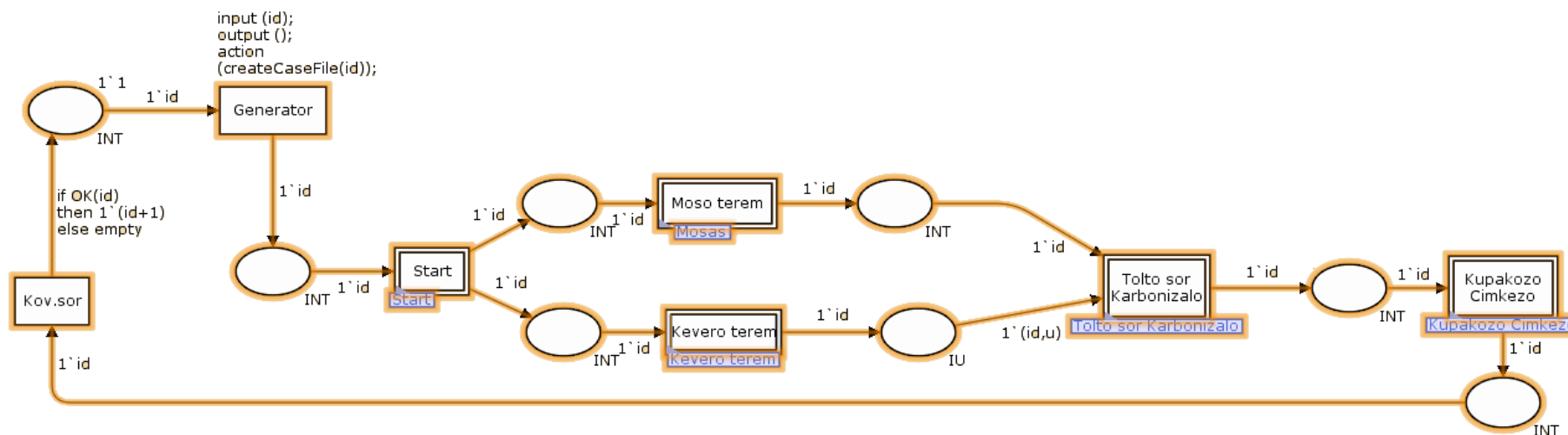
I. Irodalomjegyzék

- [1] Jackson, Michael A.: *Principles of Program Design*, Academic Press, London, 1975.
- [2] M.Blanke, M.Kinnaert, J.Lunze, M.Staroswiecki: *Diagnosis and Fault-tolerant Control*, Springer-Verlag, 2006.
- [3] <http://virt.uni-pannon.hu/index.php/oktatas/tantargyak/77-modell-alapu-diagnosztika-diszkret-modszerekkel>
- [4] Christos G. Cassandras, Stéphane Lafortune : *Introduction Discrete Event System*, Springer, 2008.
- [5] http://www.informatik.uni-hamburg.de/TGI/mitarbeiter/profs/petri_eng.html
- [6] Kurt Jensen, Lars Michael Kristensen: *Coloured Petri Nets*, Springer, 2009.
- [7] Dr. Bodon Ferenc, *Adatbányászati algoritmusok*, 2010.
- [8] Prof.dr.ir. Wil van der Aalst, *Process Mining from discovery to checking*, (<http://www.processmining.org/book/start>)
- [9] Werner Ágnes, *Folyamatbányászat; Fogalmak, eljárások és használatuk*, Modell alapú diagnosztika c. tárgy előadásvázlata
- [10] Horst Bunke, Kim Shearer: *A graph distance metric based on maximal common subgraph*, Pattern Recognition Letters 19, (255-259), 1998.
- [11] Levi, G: *A note on derivation of maximal common subgraphs of two directed or undirected graph*, Calcols, 9, 341-354, 1972.
- [12] McGergor, J.J. : *Backtrack search algorithms and maximal common subgraph problem*, Software Practice and Experience, 12 , 23-34, 1982.
- [13] <http://www.cpntools.org>
- [14] Gerzson, M., Leitold, A., Hangos, K. M., *Modell based process diagnosis using graph methods*, Factory Automation 2011 Conference, Győr, Hungary, pp.62-70
- [15] Márczi, B., Gerzson, M., Leitold, A., *Diagnostic investigations based on the Petri net model generated from the process information*, 2011. *Hungarian Journal of Industrial Chemistry Veszprém* (bírálat alatt)
- [16] Christian W. Günther and Wil M.P. van der Aalst, *A Generic Import Framework For Process Event Logs*
- [17] Wil van der Aalst , Christian Günther¹ , Jan Recker , and Manfred Reichert, *Using Process Mining to Analyze and Improve Process Flexibility*
- [18] Márczi B., *Diagnostic investigation based on Petri net model generated from the process information*, *Mobilitás és Környezet 2011 konferencia*, (2011. augusztus 29 - szeptember 1.)
- [19] A.K. Alves de Medeiros and C.W. Günther, *Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms*

[20] <http://www.promtools.org>

[21] <http://www.processmining.org/promimport>

5. Melléklet



5.1. ábra: Az üdítőgyártás hierarchikus Petri hálójának legfelső szintje

Start	
palackok érkezése	beszállító

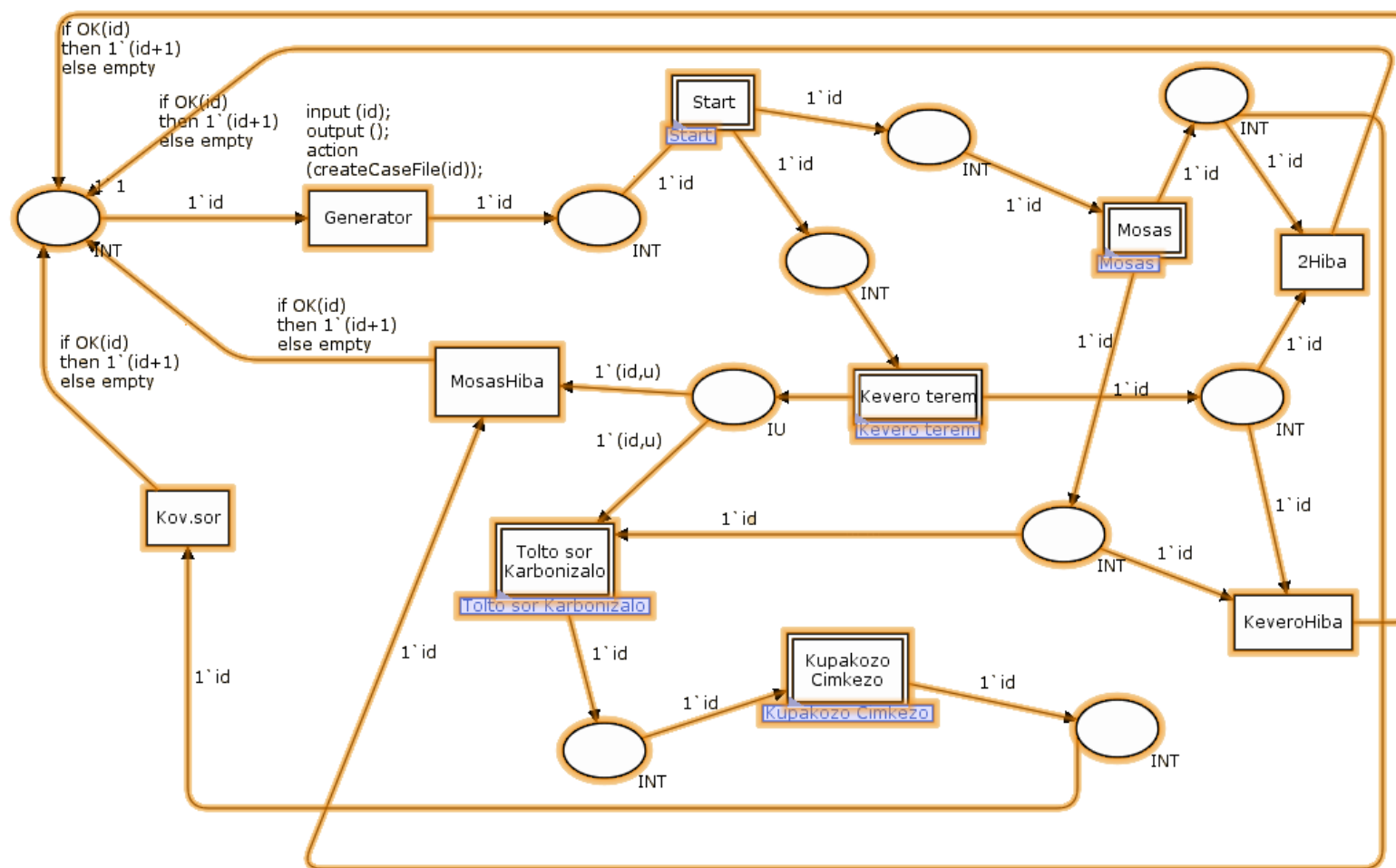
Keverő terem	
szűrt víz betöltése	keverés vezető
szirup betöltése	keverés vezető
elegyítés	elegyítő gép

Töltősor, karbonizáló terem	
szénsavasítás befecskendezéssel	karbonizáló gép
palackozó gép tárolójába továbbítás	szállító
levegő eltávolítás	palackozó gép
űdítő betöltése	palackozó gép

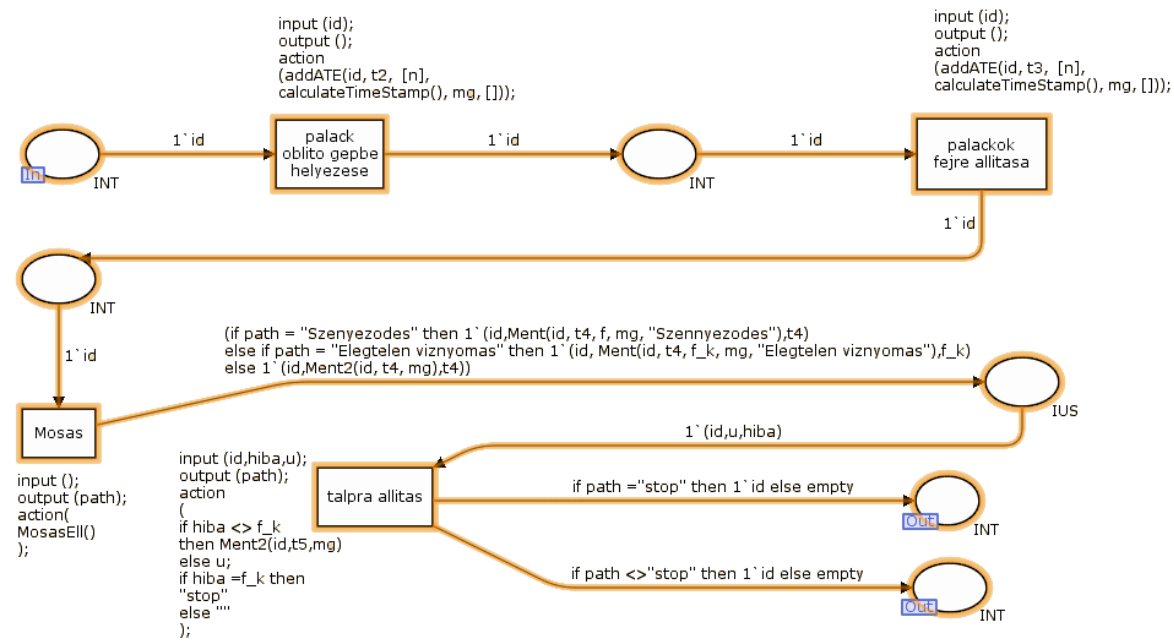
Mosó terem	
palackok öblítő gépbe helyezése	mosást végző gép
palackok feje állítása	mosást végző gép
mosás	mosást végző gép
palackok talpra állítása	mosást végző gép

Kupakozó és címkéző terem	
kupakok érkezése	szállító
kupak illesztése	kupakozó és címkéző gép
kupakok meghúzása	kupakozó és címkéző gép
hideg ragasztó felvitele	kupakozó és címkéző gép
címke illesztése	kupakozó és címkéző gép
címke simítása	kupakozó és címkéző gép

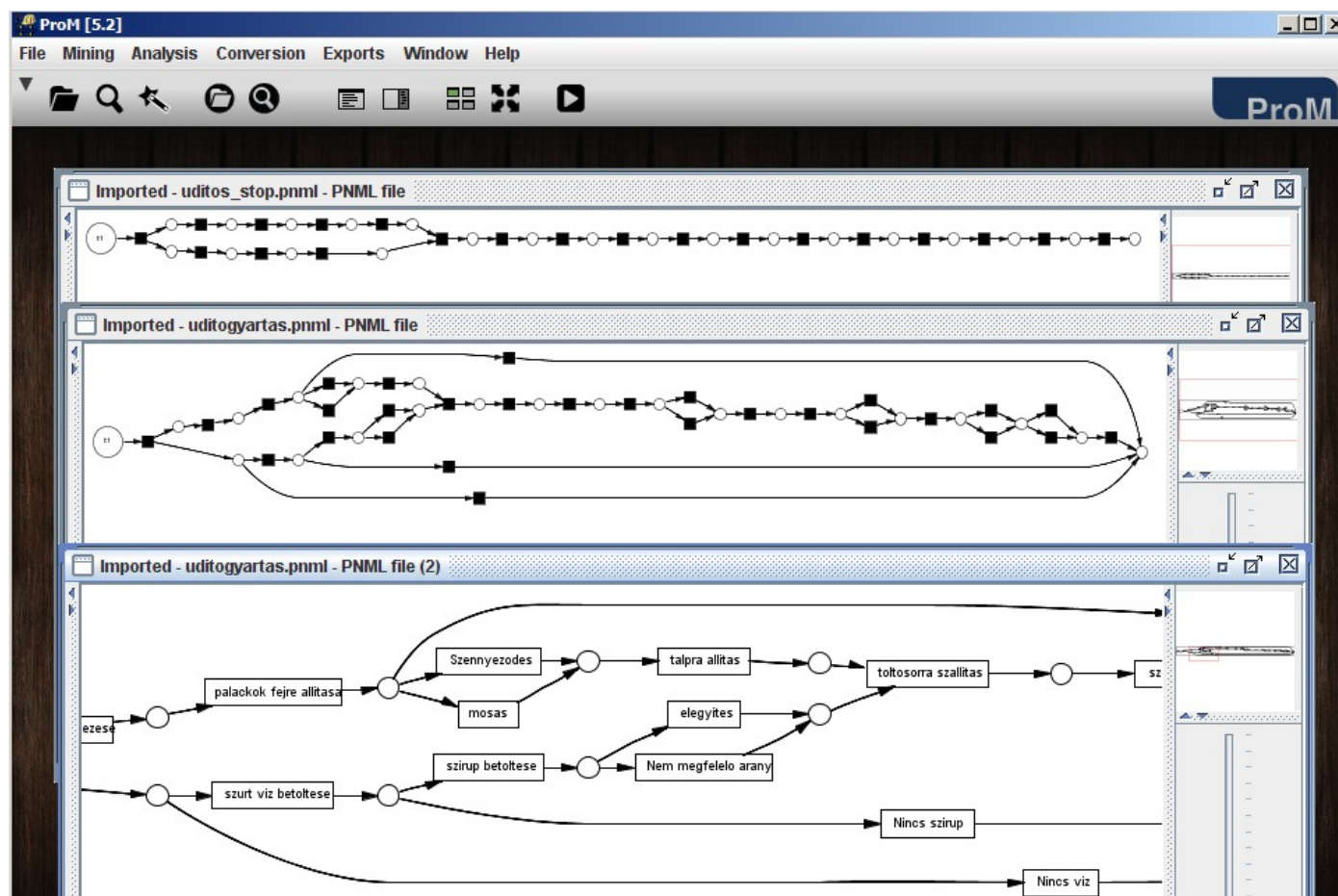
5. táblázat. Az üdítőgyártás folyamatai végrehajtókkal



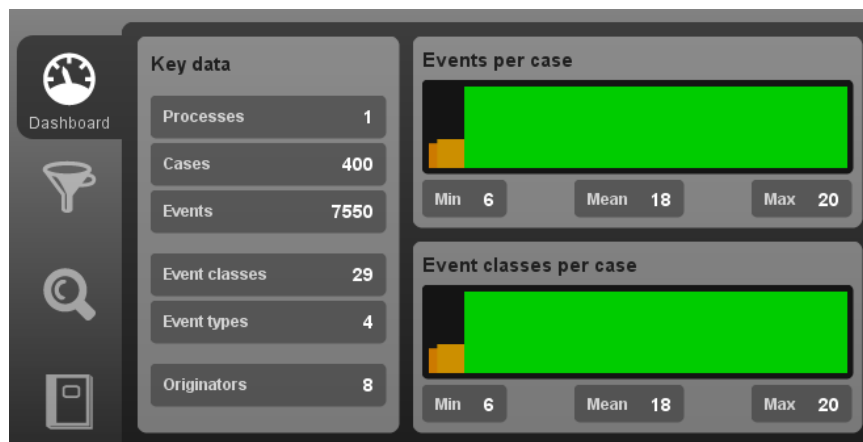
5.2. ábra. Az üdítőgyártó modell hierarchikus hálójának legfelső szintje



5.3. ábra. Üdítőgyártó modell mosóterméhez tartozó alháló



5.4. ábra. A Converter által létrehozott és a ProM által megjelenített, normál működés hálója, az ismert hibákat is tartalmazó háló, valamint ez utóbbiból egy részlet címkézett formában.



5.5. ábra. ProM által készített összegzés adott eseménynaplóból

originator	Cimke illesztése	Cimke simítása	Elegyítés	Hideg ragasztó felvitele	Kupak meghuzasa
Beszallito	0	0	0	0	0
Elegyito gep	0	0	375	0	0
Karbonizalo	0	0	0	0	0
Keveres vezeto	0	0	0	0	0
Kupakozo es Cimkezo gep	366	366	0	366	366
Mosast vegzo gep	0	0	0	0	0
Palackozo gep	0	0	0	0	0
Szallito	0	0	0	0	0

5.6. ábra. Végrehajtók tevékenységmátrixa (ProM)

Number of originators: 8		
Originator	Occurrences (absolute)	Occurrences (relative)
Kupakozo es Cimkezo gep	2196	29,086%
Mosast vegzo gep	1588	21,033%
Szallito	1098	14,543%
Keveres vezeto	795	10,53%
Palackozo gep	732	9,695%
Beszallito	400	5,298%
Elegyito gep	375	4,967%
Karbonizalo	366	4,848%

5.7. ábra. Végrehajtók szerinti értékelés (ProM)

Log events			
Number of audit trail entries: 29			
Model element	Event type	Occurrences (absolute)	Occurrences (relative)
Palackok érkezése	normal	400	5,298%
Palack oblitó gépbe helyezése	normal	400	5,298%
Palackok fejre állítása	normal	400	5,298%
Szűrt víz betöltése	normal	395	5,232%
Tálpra állítás	normal	388	5,139%
Szirup betöltése	normal	375	4,967%
Elegyítés	normal	366	4,848%
Töltőorra szállítás	normal	366	4,848%
Szenszavisítás befecskendezéssel	normal	366	4,848%

5.8. ábra. Logelemzés -részlet- (ProM): események, eseménytípusok, előfordulásuk

Udito betöltése	normal	310	4,106%
Udito betöltése	hibas	56	0,742%
Mosas	selejtet eredményező hiba	31	0,411%
Címke illesztése	hibas	22	0,291%
Szirup betöltése	leallast eredményező hiba	20	0,265%
Hideg ragasztó felvitele	hibas	17	0,225%
Mosas	leallast eredményező hiba	12	0,159%
Elegyítés	selejtet eredményező hiba	9	0,119%
Szűrt víz betöltése	leallast eredményező hiba	5	0,066%
Kupak illesztése	selejtet eredményező hiba	1	0,013%

5.9. ábra. Logelemzés -részlet- (ProM): események, eseménytípusok, előfordulásuk