

TDK DOLGOZAT

Molnár András



Nyugat-magyarországi Egyetem
Simonyi Károly Műszaki, Faanyagtudományi és
Művészeti Kar
Informatikai és Gazdasági Intézet

Általános eloszlású Petri hálók szimulációja

Szerző: Molnár András

első évfolyamos gazdaságinformatikus M.Sc. hallgató, molnaran@gain.nyme.hu

Konzulens: Dr. Horváth Ádám

adjunktus Nyugat-magyarországi Egyetem Simonyi Károly Műszaki, Faanyagtudományi
és Művészeti Kar, horvath@inf.nyme.hu

2014. november 3.

Tartalomjegyzék

1. Bevezetés.....	1
2. Elméleti áttekintő	3
2.1. Fontosabb eloszlások.....	3
2.2. Adott eloszlású véletlen szám generálása	5
2.3. Szimuláció.....	6
2.4. Szimulációs módszerek és az eredmények vizsgálata.....	7
2.5. Eseményvezérelt szimuláció (Event-driven simulation).....	12
2.6. Petri háló	13
2.7. Petri háló kibővítése	15
2.8. A tüzelők kiszolgálási típusa.....	16
3. Tervezés folyamata	19
3.1. A program célja és a követelmények ismertetése	19
3.2. Gépek szerinti modellezés – az első tervezési irányvonal	20
3.3. Petri háló szerinti modellezés – a második tervezési irányvonal	22
3.4. Konkurencia kezelése Petri hálóval	23
4. Program felépítése és működése	26
4.1. Szimuláció:.....	26
4.2. Eseménysor	29
4.3. Tüzelők (Transition) felépítése	31
5. Tesztesetek	34
5.1. M/D/1-es sorbanállási rendszer vizsgálata	34
5.2. M/D/1/2/2-es sorbanállási rendszer vizsgálata.....	35
5.3. A faablakok gyártási folyamatának modellezése	36
6. Összefoglalás.....	39
Irodalomjegyzék.....	40
Függelék	41

1. Bevezetés

Manapság a technológia rohamos fejlődésének köszönhetően a gyártórendszerek egyre hatékonyabbak, azonban a fejlődés gyakran a rendszer komplexitásának növekedésével jár, mely nem csupán a rendszer elemeinek számában, hanem azok sokféleségében és kapcsolataik összetettségében is megjelenik. Gyártórendszerek tervezése és vizsgálata ezért gyakran számítógépes támogatást igényel. A rendszerek elemzésénél általában a következő feladatokat kell elvégeznünk:

- a gyártórendszerben található szűk keresztmetszetet jelentő egység meghatározása;
- az egyes munkaállomásoknál található elemek (átlagos) számának mérése;
- a rendszerben alkalmazott strukturális módosítások hatásának mérése;
- a rendszer bizonyos elemeinek meghibásodásának hatása;
- a gyártást érintő időközök változásának hatása (pl.: igénynövekedés);
- a rendszer produktivitásának mérése.

A gyártórendszerek vizsgálatára különböző eszközök állnak rendelkezésünkre, ezek közül az egyik leggyakrabban használt módszer a szimuláció.

A feladatom hosszú távú célja, egy olyan szimulátor készítése, mely képes gyártórendszereket modellezni, annak fontosabb jellemzőit mérni, a tervezési folyamatot támogatni, valamint a gyártás maximalizálását elérni a gépek karbantartási gyakoriságának optimális értékének meghatározásával.

Jelenlegi munkám célja egy olyan szimulátor készítése, mely képes egy valós (vagy tervezett) gyártórendszer szerkezetét egy megfelelő számítógépes modellben reprezentálni, a modell működését szimulálni, és ezen a modellen méréseket végezni.

Munkám során megismerkedtem a gyártórendszerek alapfogalmaival, legfontosabb alkotórészeivel és azok funkcióival. A rendszerek felépítésének és feladatainak sokfélesége miatt egy olyan modellező eszközt kerestem, amely képes a gyártórendszer részei közti kapcsolatokat pontosan leírni, valamint grafikus reprezentációra is lehetőséget nyújt. Ez a feladat nagy kihívást jelentett, hiszen különböző termékeket gyártó rendszerek gyakran alapvetően más gyártógépeket, folyamatokat és erőforrásokat igényelnek. Két tervezési irányvonal közül a Petri háló egy általános eloszlásokkal kibővített változata mellett döntöttem, mivel a fenti feltételeknek eleget tesz.

A gyártórendszer modellezése mellett részletesen megismerkedtem a szimulációhoz kapcsolódó szakirodalommal. Megismerve a szimuláció különböző típusait a programban

lehetőséget nyújtok termináló és nem termináló szimulációk futtatására, melyek segítségével a megfigyelt rendszert különböző helyzetekben vizsgálhatjuk.

Külön hangsúlyt fektettem a szimuláció által szolgáltatott adatok statisztikai mérésére is, az itt használt számításokhoz egy saját készítésű statisztikai modult használtam, melyben szimulációra optimalizált metódusokat alkalmaztam.

2. Elméleti áttekintő

Ez a fejezet munkám elméleti háttérét foglalja össze. Felsorolom a programban használt fontosabb eloszlásokat, ismertetem a szimuláció fogalmát, annak szerepét és alkalmazási lehetőségeit, rátérek a szimuláció eredményeinek vizsgálatánál alkalmazott módszerekre, majd bemutatom a program által is használt eseményvezérelt szimulációt (event-driven simulation). Ezután bemutatom a modellezés eszközeként választott Petri hálót, majd ismertetem annak kibővítését is. Végül részletezem az újonnan bevezetett eloszlások kapcsán a memory policy fogalmát, és ennek szerepét.

2.1. Fontosabb eloszlások

Ebben az alfejezetben röviden ismertetem a munkám során használt eloszlásokat, azoknak fontosabb tulajdonságait és felhasználásait.

Determinisztikus eloszlás

A determinisztikus eloszlás egy olyan konstans diszkrét eloszlás, mely csupán egyetlen valószínűségi pontot vesz fel pontosan egy valószínűséggel.

A vizsgált eloszlások közül ez az egyik legegyszerűbb, láthatjuk, hogy ez az eloszlás csupán egy értéket vehet fel. Modellezés során tipikusan akkor használjuk, ha olyan folyamatot jellemezünk, amely a környezetétől függetlenül mindig azonos ideig tart. Például néhány gyártási folyamat hosszát ezzel jellemezhetünk [2].

Egyenletes eloszlás

Az X véletlen változó egyenletes eloszlást követ, ha sűrűségfüggvénye:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{különben.} \end{cases}$$

, hogy $a < b$.

Az egyenletes eloszlás a folytonos eloszlások közül a legegyszerűbb szerepe azonban nem elhanyagolható. Különböző eloszlású véletlen számok generálásánál fontos szerepet játszik, egy későbbi fejezet során erre külön ki is térek.

Exponenciális eloszlás

Egy X valószínűségi változó exponenciális eloszlást követ pontosan akkor, ha létezik egy $\lambda > 0$ és sűrűségfüggvénye a következő:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Kumulatív eloszlásfüggvénye pedig a következő:

$$F(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

A λ paramétert ráta paraméternek nevezzük. A folytonos eloszlások között azért számít különlegesnek az exponenciális eloszlás, mivel az egyedüli, amely rendelkezik az örökifjú tulajdonsággal, ezt a következő képlet írja le:

$$P\{X > s + t \mid X > s\} = P\{X > t\}, \forall s, t \geq 0$$

Homogén Poisson folyamat esetén két egymást követő esemény közti távolság exponenciális eloszlást követ. Sorbanállási rendszerek vizsgálatakor az érkezési folyamatot gyakran Poisson folyamatnak tekintjük, így a szimuláció számára fontos érkezési időközök exponenciális eloszlást követnek.

Gyakran használják elektronikus rendszerek megbízhatóságának modellezésére [3-5].

Normál eloszlás

Egy X valószínűségi változó normál eloszlást követ, ha sűrűségfüggvénye a következő:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}, -\infty < x < \infty,$$

ahol a két paraméter μ és σ valós számok, valamint $\sigma > 0$. A képletben használt μ paramétert az átlagot, a σ paraméter pedig a szórást jelenti.

A normális eloszlást jellemző számok közül a legfontosabbak:

$$E[X] = \mu$$

$$V[X] = \sigma^2$$

Bár modellezés szempontjából nem a leggyakrabban használt eloszlás, mégis fontos megemlíteni, mivel statisztikában az egyik leggyakrabban használt eloszlás, szimuláció során a pontosság mérésénél elengedhetetlen használata. Továbbá fontos megemlíteni, hogy alkalmazhatjuk binomiális és Poisson eloszlás közelítésére is.

A normális eloszlást egyértelműen meghatározza átlaga és szórása. Külön ki kell emelnünk a normál eloszlás esetén a standard normális eloszlást, amely egy olyan normál eloszlás, ahol $\mu = 0$ és $\sigma^2 = 1$.

Gamma eloszlás

A gamma eloszlás egy olyan folytonos eloszlás, amely két paraméterrel rendelkezik. Az eloszlást többféleképpen is meg lehet adni, a számunkra előnyösebb formát ismertetem. Az X valószínűségi változót λ paraméterű, r -edrendű gamma eloszlásúnak nevezzük, ha sűrűségfüggvénye a következő:

$$f(x) = \begin{cases} 0, & x \leq 0 \\ \frac{\lambda^r x^{r-1}}{\Gamma(r)} e^{-\lambda x}, & x > 0 \end{cases}$$

, ahol r és λ pozitív valós számok.

Továbbá a $\Gamma(x)$ a gamma függvényt jelenti, amelyet a következőképpen írhatunk fel:

$$\Gamma(x) = \int_0^{\infty} s^{x-1} e^{-s} ds$$

A gamma eloszlás nagy előnye, hogy paraméterei variálásával sok folyamatot modellezhetünk. Sokoldalúságát bizonyítja, hogy használják sorbanállási rendszerek, klimatológiai vizsgálatok, pénzügyi szolgáltatások, ipari gépek meghibásodásának vizsgálatára [6-7].

2.2. Adott eloszlású véletlen szám generálása

Szimuláció során gyakran van szükségünk adott eloszlású véletlen szám generálására. Leggyakrabban erre az események bekövetkezési időpontjának meghatározásánál van szükség.

Egyenletes eloszlású véletlen számok generálása

Legegyszerűbben egyenletes eloszlású véletlen számot generálhatunk, a véletlen szám generátorok alapértelmezésben ilyen eloszlású számokat adnak vissza. Gyakran úgy minősítik a véletlen szám generátorokat, hogy az általuk generált számok mennyire jól közelítik meg az egyenletes eloszlást.

Adott folytonos eloszlású véletlen számok generálása

Más eloszlású véletlen számok generálásánál, már nincs ilyen könnyű dolgunk, gyakran nekünk kell ezt a problémát megoldanunk. Az egyik lehetőségünk az inverz transzformáció használata.

Legyen X egy olyan folytonos eloszlású valószínűségi változó, mely kumulatív eloszlásfüggvényét F -fel adjuk meg. Célunk X eloszlásának megfelelő számok generálása. Ezt a következőképpen tehetjük meg:

Vegyünk először egy egyenletes folytonos eloszlású U valószínűségi változót 0 és 1 között. Legyen F^{-1} F inverz függvénye. Amennyiben F^{-1} létezik:

$$X = F^{-1}(U)$$

Az algoritmus használata során először vegyük a generálni kívánt eloszlás kumulatív eloszlásfüggvényét ($F(X)$) (ha csak a sűrűségfüggvény áll rendelkezésünkre, akkor integráljuk azt, hogy megkapjuk a kumulatív eloszlásfüggvényt), majd írjuk fel rá:

$$F(X) = U$$

és rendezzük X -re.

Vegyünk példának exponenciális eloszlású számok generálását. Az exponenciális eloszlás kumulatív eloszlásfüggvénye (cdf):

$$F(x) = 1 - e^{-\lambda x}$$

Írjuk fel $F(X) = U$ -t az exponenciális eloszlás kumulatív eloszlásfüggvényét behelyettesítve, és rendezzük X -re.

$$1 - e^{-\lambda x} = U$$

$$e^{-\lambda x} = 1 - U$$

$$X = -\frac{1}{\lambda} \ln(1 - U)$$

Bár ez a megoldás pontos eredményeket szolgáltat, a függvények bonyolultsága és számítási költsége nagy lehet. Ez szimulációnál már nem elhanyagolható, hiszen működés során rendkívül sok véletlen szám generálás történik, ezért különböző közelítő megoldások alkalmazása gyakran indokolt lehet. A jövőben ezek vizsgálatára is kitérek, a pontosságot és költséget figyelembe véve választok majd megfelelő megoldást [8].

2.3. Szimuláció

Egyszerűbb gyártórendszereknél még viszonylag könnyen meghatározhatóak a rendszer teljesítményjellemzői, illetve a rendszert érintő módosítások hatásai is könnyen követhetőek külön eszköz nélkül is. Bonyolultabb gyártási folyamatok vizsgálatához azonban már szükségünk van egy olyan eszközre, amely képes összetett rendszerek esetén is eredményt szolgáltatni. Rendszerek mérhető tulajdonságainak vizsgálatánál az alábbiakat használhatjuk:

- létező rendszer mérése,
- matematikai analízis,
- szimuláció.

Bár a mérés pontos adatokat szolgáltat, a gyártórendszer nem áll rendelkezésünkre, vagy rendkívül nagy költséggel járna egy létező rendszer felépítése és vizsgálata. A matematikai analízis pontos adatokat szolgáltat, viszont a rendszer bonyolultsága korlátozza ennek használatát. A korlátozó tényezők többek között szerepel a komplexitásból adódó rengeteg állapot, valamint a hálóban megadható különböző típusú tüzelők.

Az általam is választott szimuláció az egyik legszélesebb körben használt modellezési eszköz az iparban, ugyanis alkalmazható létező rendszerek vizsgálatára, valamint hasznos rendszerek tervezése során is, megfigyelhetjük, hogy a rendszer miképp reagál az egyes módosításokra. Előnye, hogy bonyolult rendszerek is modellezhetőek, valamint a modellezés mélysége is szabályozható, hátránya, hogy a futási idő gyakran hosszú, valamint a szimuláció csupán egy konkrét esetet vizsgál.

2.4. Szimulációs módszerek és az eredmények vizsgálata

A szimuláció eszközének használata során nagy hangsúlyt kell fektetnünk a kapott eredmények elemzésére, megértésére. Ebben a fejezetben röviden összefoglalom azokat a szempontokat, melyeket a szimuláció által szolgáltatott eredmények vizsgálatakor szem előtt kell tartanunk. Továbbá áttekintem az adatok vizsgálatakor használt legegyszerűbb és leggyakrabban igénybe vett módszereket, azok előnyeit és hátrányait. Érdemes megjegyezni, hogy jelenleg nem minden adatvizsgálati problémára létezik általánosan elfogadott szimulációs eljárás.

Szimuláció használata során sokan abba a hibába esnek, hogy egy szimuláció eredményére hagyatkozva vonják le következtetéseiket a vizsgált rendszerről. Egy szimuláció futtatása során az időbeli léptetésre valószínűség-eloszlásokból vett véletlen mintákat használunk, az adott minták alapján hozott becslések egy valószínűségi változó csupán egy konkrét értékét adják meg, ez a valószínűségi változó pedig nagy szórással rendelkezhet.

A szimuláció eredményeinek véletlen voltát a következőkben egy kicsit részletesebben szemléltetem:

Legyen $Y_1, Y_2, \dots$ egy kimeneti sztochasztikus folyamat egyetlen szimuláció futtatása alapján. Az Y_i -k olyan valószínűségi változók, melyek általában nem függetlenek és nem is azonosan

eloszlásúak. Ennek következtében sok statisztikai számítás, mely függetlenséget és azonos eloszlást követelnek meg (például az általam is használt konfidencia intervallum becslés) közvetlenül nem használható.

Legyenek $y_{11}, y_{12}, \dots, y_{1m}$ az $Y_1, Y_2, \dots, Y_m$ valószínűségi változók olyan lehetséges értékei, melyeket egy m hosszúságú szimuláció során (m darab mérést végezve) az $u_{11}, u_{12}, \dots$ véletlen számokat felhasználva kaptunk. Ha a szimulációt véletlen számok egy más sorozatával ($u_{21}, u_{22}, \dots$) futtatjuk, akkor $Y_1, Y_2, \dots, Y_m$ valószínűségi változók más lehetséges értékeit kapjuk ($y_{21}, y_{22}, \dots, y_{2m}$). Tehát, ha n független m hosszú futtatást (replikációt) végzünk (különböző véletlen számokat, valamint azonos kiindulási állapotot használunk), akkor a következő értékeket kapjuk:

$$\begin{array}{c} y_{11}, \dots, y_{1i}, \dots, y_{1m} \\ y_{21}, \dots, y_{2i}, \dots, y_{2m} \\ \vdots \\ y_{n1}, \dots, y_{ni}, \dots, y_{nm} \end{array}$$

A fenti változóknál az egy sorban elhelyezkedő változók egy futáshoz tartozó, különböző időpontban mért értékeket jelentenek, az egy oszlopban található változók pedig különböző futások azonos időpontban mért értékét adják meg.

Az egy sorban található értékek nyilván nem tekinthetők függetlennek. Az egyes oszlopok értékeit azonban függetlennek és azonos eloszlást követőnek tekinthetjük (az i -edik oszlop értékei az Y_i valószínűségi változó értékei). Ezt nevezzük futások közti függetlenségnek, ezt a tulajdonságot használok fel a későbbiekben az analízis során.

A sztochasztikus folyamatok tranziens és egyensúlyi viselkedése

Vegyük a következő kimeneti folyamatot: $Y_1, Y_2, \dots$. Legyen $F_i(y|I) = P(Y_i \leq y|I)$ minden $i = 1, 2, \dots$ értékre, ahol y egy valós szám, és I jelenti a kezdeti feltételeket. Az $F_i(y|I)$ -t nevezzük az i időpontban vizsgált, I kezdeti állapottal rendelkező kimeneti folyamat tranziens eloszlásának. Ha $F_i(y|I) \rightarrow F(y)$ $i \rightarrow \infty$ minden y és minden I mellett, akkor $F(y)$ -t egyensúlyi eloszlásnak hívjuk. Szimuláció futtatása során nyilván nem használhatjuk a $i \rightarrow \infty$ feltételt, azonban találhatunk egy olyan $k + 1$ időindexet, amelytől fogva az eloszlások jól megközelítik egymást. Szimuláció során gyakran feladatunk, hogy megkeressük ezt a $k + 1$ indexet, amely időponttól az eloszlást egyensúlyi eloszlásnak tekinthetjük.

A kimenet vizsgálata szempontjából két fajta szimulációt különböztetünk meg:

- termináló szimuláció,

- nem-termináló szimuláció.

Termináló szimuláció

Ennél a típusnál az egyes futások hosszát egy „természetes” esemény határozza meg. Az általam megvalósított programban ez az esemény a paraméterként megadható futási hossz átlépése.

Az egyensúlyi vizsgálattal ellentétben a kezdeti feltételek általában hatással vannak az eredményre, ezért ezeket a feltételeket a vizsgált rendszer alapján kell megadni.

Szimuláció során n futást végzünk azonos kezdeti feltételekkel valamint futásonként különböző véletlen számokkal.

Tegyük fel, hogy a szimulált rendszer egy számunkra érdekes paraméterét szeretnénk vizsgálni. Legyen ez a paraméter X , meg szeretnénk becsülni a X várható értékét., és jegyezzük fel értékét az egyes futtatások során. Vizsgáljuk X lehetséges értékeit a j -edik iterációban X_j valószínűségi változóval! A különböző replikációk során vizsgált X_j -ket felhasználva megkaphatjuk a mintaátlagot:

$$\bar{X}(n) = \frac{\sum_{j=1}^n X_j}{n},$$

mely felhasználható a X várható értékének becslésére, mivel az X_j -k ($j = 1, 2, \dots, n$) függetlennek és azonos eloszlásúnak tekinthetők. A fenti mintaátlag felhasználásával kiszámíthatjuk a minta szórásnégyzetét is:

$$S^2 = \frac{\sum_{j=1}^n (X_j - \bar{X}(n))^2}{n-1},$$

mellyel X szórásnégyzetét becsülhetjük meg.

Ezek után felírhatjuk a X várható értékére megközelítőleg $100(1 - \alpha)$ százalékos ($0 < \alpha < 1$) konfidencia intervallumot:

$$\bar{X}(n) \pm t_{n-1, 1-\alpha/2} \sqrt{\frac{S^2(n)}{n}}$$

A fenti konfidencia intervallum használata esetén azzal a problémával szembesülünk, hogy nem tudjuk szabályozni a konfidencia intervallum nagyságát (pontosabban a $t_{n-1, 1-\alpha/2} \sqrt{\frac{S^2(n)}{n}}$ értékét) és az $\bar{X}(n)$ pontosságát, hiszen ezek értéke az előre megadott replikációk számától

függ (n). A pontosság kérdésére visszatérek a Program felépítése és működése c. fejezetben, ahol a konkrétan részletezem a megvalósítást és a használt metódusokat.

Nem-termináló szimuláció - egyensúlyi paraméterek statisztikai vizsgálata

Ennél a szimulációnál nincs olyan előre meghatározott „természetes” esemény, amely meghatározná a szimuláció végét. Általában egyensúlyi paraméterek vizsgálatára használjuk, ilyen paraméter lehet például a rendszer egy gyártóegysége előtt sorbanálló igények átlagos száma.

Tegyük fel, hogy meg akarjuk becsülni $Y_1, Y_2, \dots$ folyamat várható értékét ($E(Y)$). Erre több megközelítés is létezik, programom készítése során két megközelítést valósítottam meg:

- Replikáció/törlés megközelítés,
- Batch átlagok megközelítés.

Rendszerek indításakor gyakran szembesülhetünk azzal a ténnyel, hogy a rendszer nem kerül rögtön stabil állapotba. Ezért a stabil állapot előtti megfigyelések, mérések nem a rendszer állandó állapotáról tanúskodnak, így statisztikai becsléseink is pontatlanok lesznek. A rendszerek általában fokozatosan érik el a stabil állapotot, majd utána ebben az állapotban működnek tovább.

A probléma megoldására a legkézenfekvőbb megoldás, hogy a felfűtési időszak során mért adatokat nem használjuk fel becsléseinkhez. Nagyobb problémát jelent a felfűtési időszak végének meghatározása, hiszen ez rendszerről rendszerre más és más.

Bár léteznek megoldások a felfűtési idő hosszának becslésére, jelen pillanatban az általam készített program csupán fix hosszúságú, felhasználó által megadható felfűtési időt tud kezelni. A becsléshez használt adatok vizsgálata csupán a felfűtési időszak letelte után kezdődik meg.

A jövőben ezzel a problémával bővebben foglalkozok, és becslések segítségével lehetőséget nyújtok a felfűtési időszak végének meghatározására.

Replikáció/törlés megközelítés

Ez a megközelítés nagyban hasonlít a termináló szimulációnál tárgyaltakhoz azzal a különbséggel, hogy csupán azokat a méréseket használom a számításokhoz, melyek a felfűtési idő után következnek.

Legyen a felfűtési idő hossza minden replikációban l , ettől az időponttól vizsgáljuk a méréseket. Továbbá legyen a végzett replikációk száma n' , a replikációk hossza pedig legyen

m' , ahol m' legyen jóval nagyobb, mint l . Legyen Y_{ji} a j -edik replikáció i -edik időpontban vizsgált rendszerparaméter valószínűségi változója. Ekkor X_j a következő:

$$X_j = \frac{\sum_{i=l+1}^{m'} Y_{ji}}{(m' - l)}$$

$j = 1, 2, \dots, n'$. Ebben az esetben X_j -k független és azonos eloszlású valószínűségi változóknak tekinthetők, tehát használhatjuk a termináló szimulációnál használt képleteket $E(X)$ becsléséhez, felírhatunk rá egy megközelítőleg $100(1 - \alpha)$ százalékos ($0 < \alpha < 1$) konfidencia intervallumot:

$$\bar{X}(n') \pm t_{n-1, 1-\alpha/2} \sqrt{\frac{S^2(n')}{n'}}$$

Ennél a megközelítésnél a pontatlanság a becsült várható értéknél jelenik meg leginkább.

Bár programom fejlesztése közben ebből a megközelítésből indultam ki, a fent leírtaktól néhány pontban eltértem, ezeket a különbségeket a program működésénél részletezem.

Batch átlagok megközelítés

Ez a megközelítés csupán egy futással dolgozik, hogy használhassuk a statisztikai módszereket, itt is el kell érünk, hogy a méréseink független legyenek.

Tegyük fel, hogy $Y_1, Y_2, \dots$ egy olyan kovariancia-stacionárius folyamat, ahol $E(Y_i) = v$ minden i -re. (Amennyiben létezik v , elég nagy felfűtési idő (l) eltelte után $Y_{l+1}, Y_{l+2}, \dots$ közel kovariancia-stacionáriusnak tekinthető). Tegyük fel továbbá, hogy a szimulációt m hosszú ideig futtatjuk. Bontsuk fel m -et n darab egyenként k hosszú egységre (batchre). Legyen $\bar{Y}_j(k)$ a j -edik batch átlaga, amelyet a batchben található k mérésből számolunk. Legyen továbbá

$$\bar{\bar{Y}}(n, k) = \frac{\sum_{j=1}^n \bar{Y}_j(k)}{n}$$

$\bar{\bar{Y}}(n, k)$ -t használjuk majd a későbbiekben v becslésére.

Amennyiben a batchek méretét (k) elég nagyra állítjuk, megmutatható, hogy $\bar{Y}_j(k)$ -k megközelítőleg korrelálatlanok, továbbá megközelítőleg együttesen normális eloszlásúnak tekinthetők. Ha $\bar{Y}_j(k)$ -k megközelítőleg korrelálatlanok és megközelítőleg együttesen normális eloszlásúak, akkor megmutatható, hogy $\bar{Y}_j(k)$ -k függetlenek és normális eloszlásúak. Mivel $Y_1, Y_2, \dots$ folyamatot kovariancia-stacionáriusnak tekintettük $E(Y_i) = v$ tulajdonsággal, ezért $\bar{Y}_j(k)$ -k ugyanazzal av átlaggal és ugyanazzal a varianciával rendelkeznek.

A fentiekből következik, hogy $\bar{Y}_j(k)$ -k normál valószínűségi változók ugyanazzal az átlaggal és varianciával. Mivel a normál valószínűségi változót egyértelműen meghatározza az átlaga és varianciája, ezért $\bar{Y}_j(k)$ -k v átlagú azonos eloszlású valószínűségi változók.

Ezek után beláthatjuk, hogy ha k értékét elég nagyra állítjuk, $\bar{Y}_j(k)$ -kat függetlennek és (v átlagú) azonos eloszlásúnak tekinthetjük. Ezek után már használhatjuk a korábban ismertetett becsléseket, és számolhatunk konfidencia intervallumot is.

A replikáció/törlés megközelítéssel szemben itt a pontatlanság a szórásnégyzet számításánál jelentkezhet [9].

2.5. Eseményvezérelt szimuláció (Event-driven simulation)

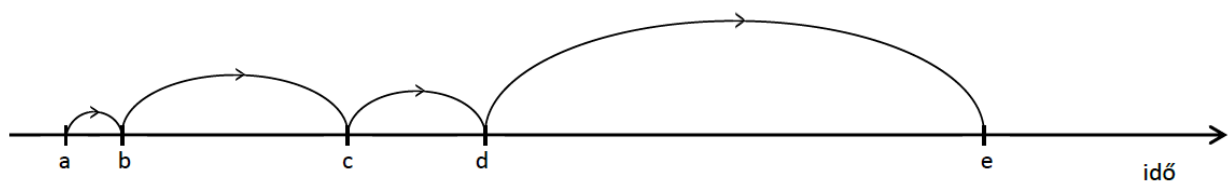
Szimuláció során a rendszer elemeinek időbeli működését figyeljük, ennek követésére szükségünk van egy eszközre, amely számon tartja, valamint dinamikusan növeli az időt. Az idő követésére, egy szimulációs időt tartunk számon, melyet a szimuláció futtatása során frissítünk. A szimulációs idő frissítésére a következő két módszer a leggyakoribb:

- a szimulációs időt akkor frissítjük, mikor a rendszer állapota megváltozott (1. ábra);
- a szimulációs időt mindig azonos mértékben léptetjük (2. ábra).

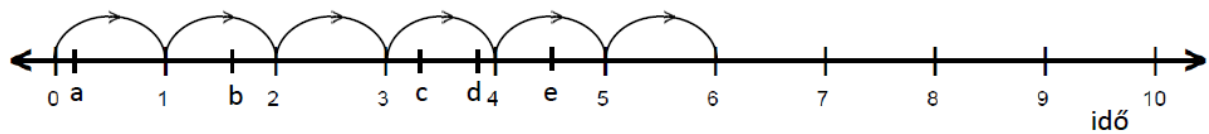
(Megj.: Az ábrákon a betűk események bekövetkezésének időpontját jelölik.)

A második megoldás előnye, hogy egyszerű, azonban a léptetés mértéke gondot okozhat, valamint a program számításidejét is jelentősen növelheti.

Az első megoldás mellett döntöttem, mivel ennél csupán, akkor végzünk méréseket, amikor a rendszer állapota ténylegesen megváltozik, így a program hatékonyabban dolgozik. Ebben az esetben tehát, csupán akkor frissül a szimulációs idő, ha egy olyan esemény következik be, amely megváltoztatja a rendszer állapotát. Ez az esemény mindig az időben leghamarabb bekövetkező esemény, a szimulációs idő, pedig az esemény bekövetkezésének időpontjához ugrik. Ennek az a következménye, hogy a rendszer állapota az esemény bekövetkezésének pillanatában megváltozik, és ebben az állapotban marad a soron következő esemény bekövetkezéséig. Ezt a módszert nevezzük eseményvezérelt szimulációnak [3].



1. ábra - Eseményvezérelt szimuláció



2. ábra - Idő vezérelt szimuláció

2.6. Petri háló

Bonyolult rendszerek létrehozása folyamán a könnyebb megértés és az átláthatóság érdekében szükségünk van olyan grafikus megjelenítésre, amely áttekinthetővé teszi munkánkat, valamint a rendszer formális leírását is lehetővé teszi. Kevés olyan eszköz van, amely mind a kettő tulajdonsággal bír, az egyik ilyen eszköz a Petri háló.

A Petri háló egy olyan eszköz, amellyel modelljeinket grafikusan leírhatjuk, valamint lehetőséget nyújt a szinkronizáció a fellépő konfliktushelyzetek, valamint kölcsönös kizárás jelölésére.

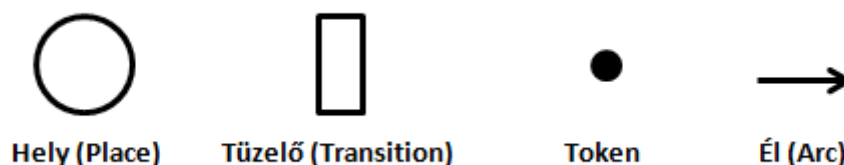
Sztochasztikus Petri hálók (SPN)

Ez a felírás a Petri hálók egy olyan bővítése, melynél a tüzelőknél számon tartjuk a tüzelési rátát (minden tüzelőnél külön-külön). A programban ettől annyiban térek el, hogy ráta helyett késleltetést használok, amely a ráta reciproka. Az SPN esetében az időközök exponenciálisan elosztottak, a Petri háló kibővítése c. fejezetben kitérek ennek fontosságára.

Egy Petri háló négy különböző összetevőből épülhet fel:

- hely (place): melyek a tokenekkel együtt a rendszer állapotát írja le,
- tüzelő (transition): eseményeket ír le, melyek megváltoztatják a rendszer állapotát,
- él (arc): definiálja a kapcsolatokat helyek és tüzelők között, súllyal jellemezhetjük,
- token: jelölők, melyek a helyekben tartózkodnak, jellemzik annak állapotát.

A következő ábrán (3. ábra) az alkotóelemek jelölését láthatjuk:



3. ábra - Az Petri háló alkotóelemei és jelölésük

A Petri hálót egy irányított gráfként képzeljük el, melynek csúcsait helyek és tüzelők képezik, ezeket a csúcsokat élek kötik össze. Fontos megjegyezni, hogy helyből csak tüzelőbe, tüzelőből pedig csak helybe futhat él. Egy tüzelőbe befutó éleket bemeneti éleknek, a belőle kifutó éleket pedig kimenő éleknek nevezzük. Egy tüzelőhöz rendelt esemény csak, akkor következhet be, ha az összes bemenő élnél szereplő helynél található legalább az él súlyának megfelelő számú token. Az esemény bekövetkezését nevezzük tüzelésnek, mely során az összes bemenő helyről az összekötő él súlyának megfelelő mennyiségű tokent távolítunk el, valamint az összes kimenő helyre az összekötő él súlyának megfelelő mennyiségű tokent helyezünk [10].

Általános sztochasztikus Petri hálók (GSPN):

Korábbi munkáim során, sorbanállási rendszerek vizsgálatánál a sztochasztikus Petri hálók egy bővített változatát, az általános sztochasztikus Petri hálót (GSPN) használtam.

A GSPN egy olyan Petri háló, amely az SPN szokványos elemein kívül a következő elemeket tartalmazza:

- rögtöntüzelő,
- tiltó él.

A rögtöntüzelő (Immediate transition) fontos tulajdonsága, hogy késleltetés nélkül tüzel, tehát amint a tüzelő engedélyezetté válik, rögtön tüzel, elsőbbséget élvezve a többi késleltetett tüzelővel szemben. Előfordulhat, hogy egyszerre több rögtöntüzelő is engedélyezett, ebben az esetben az a tüzelő tüzelhet, mely nagyobb prioritással rendelkezik. Amennyiben a konkurens rögtöntüzelők prioritása is megegyezik (vagy nincs külön megadva), akkor a tüzelőkhöz rendelt súlyokat vizsgáljuk. Ebben az esetben sorsolunk a tüzelők között a súlyokat is figyelembe véve, vagyis a nagyobb súllyal rendelkező nagyobb valószínűséggel tüzelhet. Példának vegyünk két konkurens tüzelőt, az elsőhöz rendelt súly kettő, a másodikhöz rendelt súly pedig nyolc. Ebben az esetben a kettő tüzelő közül az első 0,2-es valószínűséggel, míg a második 0,8-as valószínűséggel tüzelhet. Általában egy fekete vonallal vagy egy vékony téglalappal jelöljük.

Tiltó élet (inhibitor arc) abban az esetben használjuk, mikor a helyi állapot vagy egy feltétel letilt egy tüzelőt. Egy helytől a tüzelőhöz mutató tiltó él tehát azt jelenti, hogy amennyiben a helynél található token, a tüzelő nem engedélyezett. Az érintett tüzelő csupán abban az esetben válik engedélyezetté, ha a bemeneti éleknél minden tiltó élhez tartozó helynél egy

token sem szerepel, és minden hagyományos élhez tartozó helynél legalább az él súlyának megfelelő mennyiségű token található.

A következő ábrán (4. ábra) a tiltó él és a rögtöntüzelő jelölését láthatjuk [11].



4. ábra - A rögtöntüzelő és a tiltó él jelölése

2.7. Petri háló kibővítése

Gyártórendszerek modellezése során gyakran előfordul, hogy az exponenciális eloszlástól eltérő eloszlást használunk. Például mechanikai meghibásodásból adódó (például eszközök kopása) eseményeknél, inkább gamma eloszlást használunk. Ennek következtében már nem alkalmas sem az SPN, sem a GSPN ilyen rendszerek vizsgálatára.

Olyan modellezési eszközre van tehát szükségünk, amely képes más eloszlásokat is integrálni. Mint, ahogy egy korábbi fejezetben említettem, az exponenciális eloszlás különleges a folytonos eloszlások között, hiszen az egyetlen emlékezetmentes eloszlás. Ez a tulajdonság néhány olyan problémát hárított el, amellyel most szembe kell néznünk.

Egy ilyen felmerülő probléma, hogy mi történjen a tüzelésig hátramaradó idővel, abban az esetben, amikor a tüzelő már nem engedélyezett, illetve abban az esetben, ha megváltozik a tokeneloszlás.

Az időértéket tekintve két mechanizmust definiálhatunk:

- continue, mely esetén az időzítő értékét megtartjuk;
- restart, mely esetén az időzítőt újraindítjuk.

A felvetett probléma megoldására vezetjük be a memory policy tulajdonságot. Ez a tulajdonság a hátralévő időt tekintve a következő három irányelvet definiálja:

- resampling,
- enabling memory,
- age memory.

Resampling:

- minden egyes tüzelés után eldobjuk a tüzelőhöz tartozó időt, függetlenül attól, hogy a tüzelő engedélyezve van-e, vagy sem;
- amennyiben a tüzelő engedélyezve lett a tüzelés után, az időzítőt újraindítva generálunk számára egy új időpontot;

Enabling memory:

- a hátralévő időt megtartjuk, amennyiben a vizsgált tüzelő engedélyezve maradt;
- a hátralévő időt eldobjuk, amennyiben a vizsgált tüzelő nem engedélyezett;
- amennyiben a tüzelő újra engedélyezetté válik új időpontot generálunk az esemény számára;

Abban az esetben használjuk, ha a megszakítás hatására elveszik a végzett munka.

Age memory:

- a hátralévő időt mindig megtartjuk minden esetben;
- amennyiben a tüzelő nem engedélyezett, a hátra maradt időt eltároljuk, és a következő engedélyezéskor új időpont generálása helyett az eltárolt hátralévő időt használjuk;

Akkor használjuk az age memory opciót, ha megszakítás esetén a meglévő munka nem veszik el [12].

2.8. A tüzelők kiszolgálási típusa

Mielőtt rátérnék a kiszolgálás típusának ismertetésére, érdemes egy másik fogalmat tisztázni, ez a fogalom az engedélyezettség foka (enabling degree).

Engedélyezettség foka:

Tüzelők esetében beszélünk erről a fogalomról, mely azt adja meg, hogy a vizsgált tüzelő hányszor tüzelhetne egy adott tokeneloszlás esetén mielőtt letiltásra kerülne.

Tüzelők esetében a következő kiszolgálási típusokról beszélhetünk:

- egy szerveres (exkluzív);
- több szerveres (n-szerver);
- végtelen szerveres.

A továbbiakban a következő jelöléseket használom:

- $ED(t, m)$: A t tüzelő m tokeneloszlásban vett engedélyezési foka;

- $\theta(t)$: A t tüzelő intenzitása;
- $\theta(t, m)$: A t tüzelő m tokeneloszlásban vett tüzelési intenzitása.

A programban az egy szerveres kiszolgálást használom alapértelmezettként. Ezt az opciót használva a tüzelő csupán egy klienst képes egyszerre kiszolgálni. A tüzelés intenzitása, pedig független az engedélyezés fokától.

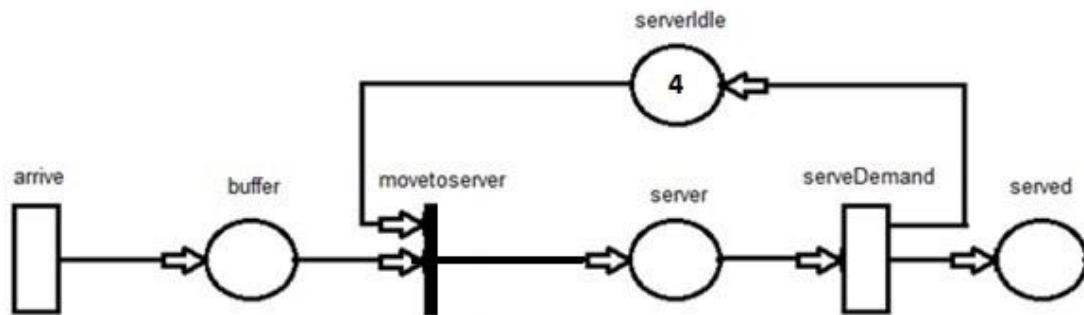
Több szerveres (n) kiszolgálás esetén a tüzelő legfeljebb n klienst képes párhuzamosan kiszolgálni. A tüzelés intenzitása, pedig a szerverszámtól (n) és az aktuális engedélyezés fokától függ:

$$\theta(t, m) = \min(n, ED(t, m)) \theta(t)$$

Végtesen szerveres kiszolgálás esetén a tüzelő végtesen számú klienst képes párhuzamosan kiszolgálni. Itt a tüzelés intenzitása egyszerűbben megadható, csupán az engedélyezés fokától függ [12]:

$$\theta(t, m) = ED(t, m) \theta(t)$$

A programban jelenleg az egy szerveres illetve a végtesen szerveres megoldást adhatjuk meg az egyes exponenciális késleltetésű tüzelőknél. Az n-szerveres megoldás azonban megvalósítható a Petri háló segítségével, erre láthatunk példát a következő ábrán (5. ábra). (Megj.: Az ábrán látható *serveDemand* tüzelő végtesen típusú kiszolgálójának tekintendő)



5. ábra - Az M/M/4 – es sorbanállási rendszer Petri hálójá

Az exponenciális késleltetésű tüzelőknél a végtesen típusú kiszolgálást egyszerűen meg tudtam valósítani, az eloszlás emlékezetmentessége miatt. Ennek a tulajdonságnak köszönhetően, elég csupán egy időpontot számon tartani, illetve frissíteni. Frissítés akkor, szükséges ha változott a tüzelő engedélyezési foka az adott tokeneloszlásban. Ebben az

esetben töröljük a tüzelőhöz tartozó korábbi időparamétert, és az új $\theta(t, m)$ paraméter felhasználásával generáljuk az új időpontot.

A többi eloszlás esetében azonban nem ilyen egyszerű a helyzet, hiszen már nem hanyagolhatjuk el az egyes klienseknél a kiszolgálásból hátramaradó időt. Ebben az esetben már nem hagyatkozhatunk egy időzítőre. Két esetet kell figyelembe venni:

- Az engedélyezési fok nő,
- Az engedélyezési fok csökken.

Amennyiben az engedélyezési fok nő egy új időzítőt inicializálunk az aktuális klienshez, a tüzelő $\theta(t)$ paraméterével. Így egy tüzelőhöz párhuzamosan több esemény és időzítőt kell számon tartanunk. Pontosabban minden klienshez egy eseményt és a hozzá tartozó időt kell eltárolnunk.

Az engedélyezési fok csökkenését már nehezebb kezelni, hiszen a csökkenés mértékének megfelelően kell eseményeket eltávolítanunk. Ilyenkor több kérdés is felmerülhet: Melyik kliens(eket) távolítsuk el? Mi történjen a törölt események hátralévő idejével?

A végtelen szerver lehetőségét a nem exponenciális eloszlással rendelkező tüzelőnél jelenleg nem adható meg a programban, azonban munkám folytatása során célom, hogy a végtelen szerver típusú kiszolgálás ezeknél a tüzelőknél is lehetőség legyen.

3. Tervezés folyamata

Ebben a fejezetben áttekintem a tervezés folyamatát, bemutatom a programmal szemben támasztott követelményeket, a különböző szempontokat, melyek a tervezési folyamatot alakították. Ezután ismertetem az első tervezési irányvonalat, bemutatom ennek előnyeit, hátrányait, a hozzá használt modell felépítését, majd részletezem, hogy miért vettem el a megközelítést. Majd áttérek a második irányvonalra, leírom az alkalmazott modell felépítését. Részletesen bemutatom a modell előnyeit és hátrányait, valamint az okot, amiért ezt a megoldást alkalmaztam a programban.

3.1. A program célja és a követelmények ismertetése

A feladatom hosszú távú célja, egy olyan szimulátor készítése, mely képes gyártórendszereket modellezni, annak fontosabb jellemzőit mérni, a tervezési folyamatot támogatni, valamint a gyártás maximalizálását elérni a gépek karbantartási gyakoriságának optimális értékének meghatározásával. Mielőtt rátérnék a program céljainak részletesebb ismertetésére érdemes a gyártórendszerek fogalmát megismernünk.

A gyártórendszert leírhatjuk

- emberek,
- folyamatok,
- struktúrák,
- szervezeti egységek,
- információs rendszerek,
- irányítórendszerek,
- karbantartó eszközök,
- gépek,
- szállítóeszközök,
- számítógépek,
- pufferek,
- és egyéb egymással kapcsolatban lévő eszközök

kombinációjaként, melynek feladata, hogy különböző anyagokból valami hasznos eszközt készítsen [13].

A fenti felsorolásból is láthatjuk, hogy összetett rendszerekről beszélünk, amelyek gyakran alapvetően különböző felépítésű és tulajdonságú eszközökből tevődnek össze. Külön kihívást

jelent egy olyan modell megalkotása, mely a fenti eszközök és személyeket megfelelően tudja modellezni.

A program célja, hogy egy valós (vagy tervezett) gyártórendszer szerkezetét egy megfelelő számítógépes modellben reprezentáljuk, és ezen a modellen méréseket végezzünk. A mérés során a rendszer legfontosabb jellemzőit ismerhetjük meg, ilyenek például a szűk keresztmetszet, a meghibásodások száma és azok gyakorisága, a rendszer (és annak részeinek) teljesítményjellemzői valamint a rendszerben alkalmazott módosítások hatásai.

3.2. Gépek szerinti modellezés – az első tervezési irányvonal

Az első modell a gyártórendszerek egyik kulcsfontosságú elemét a gyártógépeket helyezte a központba. Ez az ötlet elsőre nagyon jónak tűnik, hiszen a gyártórendszerek felépítésétől nem esik túl messze ez a megközelítés, hiszen az egyik alapvető egységről beszélünk, nem kell túlságosan elvonatkoztatnunk a valós rendszertől, ezért egy külső szemlélő számára is viszonylag egyszerű ismertetni a modell felépítését. Mivel egy jól definiált eszközt állítunk a modellezés középpontjába, ezért a felépített modell mérete sem lesz túl nagy és átláthatatlan.

A gyártóberendezések modellbeli felépítését a következőképpen képzelhetjük el:

A modellben definiált helyek szerepükben hasonlítanak a Petri hálók esetében definiált helyek feladatához. Ezek adják meg a rendszerben lévő erőforrások, eszközök és anyagok helyét. Tervezés során felmerült problémaként az erőforrások sokfélesége. Például néhány gép működtetéséhez szükség lehet bizonyos személyek jelenlétére, különböző típusú energiaforrásokra, a gép feladatától függően különböző anyagokra és további, a gyártási folyamat típusától függő elemekre. Láthatjuk, hogy rendkívül sokféle feltételt szükséges vizsgálni, ezek számára egy általánosan elfogadható felépítési szerkezetet találni rendkívül nehéz.

A gyártórendszerben a gyártáshoz szükséges elemek gépek sorozatán haladnak át, amelyek különböző erőforrásokat felhasználva átalakítják az elemeket. Két egymás után következő gépet helyek kötnek össze, egy gép rendelkezik bemenő és kimenő helyekkel.

A bemenő helyek listája definiálja a gép működéséhez szükséges erőforrásokat, eszközöket, anyagokat. Továbbá minden bemenő helyhez megadható a belőle szükséges mennyiség is. Fontos megjegyezni, hogy csupán, akkor gyárthatunk, ha az összes bemeneti helynél rendelkezésre áll a megfelelő mennyiség.

Kimenő helyek listája azokat az egységeket definiálja, ahova a gép az általa végzett munka eredményét továbbítja. Működése bemenő helyekhez hasonló, tehát a helyek mellett megadható a mennyiség is.

A helyek esetén felmerülhet a kérdés, hogy mekkora kapacitással bírjanak. Vegyünk példának egy olyan esetet, ahol két egymás után következő gép végzi feladatát, azonban a második meghibásodik, ezért nem tudja a bemeneti helyén felgyülemelő anyagokat feldolgozni, így az anyag egész addig gyűlik, amíg a gép meghibásodását elhárítják.

A gyártógépek által generált események:

A vizsgált modellben talán a legnehezebb modellezhető eszközöket a gyártógépek képezik. A szimuláció szemszögéből nézve ezek az egységek generálják a különböző eseményeket. Fel kell tennünk a kérdést, hogy a gépek milyen események generálásában vehetnek részt. A tervezés során a következő eseményeket határoztam meg: *production*, *failure*, *repair*, *maintenance begin*, *maintenance end*.

A felsorolt eseményeket minden gépre külön-külön kell vizsgálnunk, tehát a következőkben leírt események egy adott gépre vonatkoznak.

A *production* (gyártás) esemény felelős a tényleges gyártásért. Emlékezzünk, hogy csupán akkor generálhatunk ilyen eseményt, ha a bemeneti helyeknél rendelkezésre áll a megfelelő mennyiségű anyag, illetve erőforrás. Ha ez a feltétel teljesül, akkor a bemeneti helyeknél csökkentjük az anyagok és erőforrások mennyiségét a bemenő listában megadott mérték szerint, a kimenő helyeknél pedig az anyagok és erőforrások mennyiségét növeljük a kimenő listában szereplő mérték szerint. Fontos kérdés lehet, hogy mit történjen, abban az esetben mikor valamelyik kimenő helynél a mennyiség elérte a megadott limitet.

A *failure* esemény felelős a meghibásodások modellezéséért. Gyakran azért futtatunk szimulációkat, hogy a meghibásodások rendszerre kifejtett hatását figyeljük meg. Az esemény hatására a gép meghibásodik, ezért a géphez tartozó gyártás eseményt törölni kell az eseménysorból, valamint jelezni kell, hogy a gép hibás (ne generáljunk több *production* eseményt), továbbá generálni kell a gép számára egy *repair* (javítás) eseményt.

A *repair* a géphez rendelt javítás eseménye. Az esemény hatására a gép újra működőképes, tehát generálhatunk *production* eseményeket, amennyiben rendelkezésre állnak a szükséges erőforrások. Amennyiben a *maintenance begin* eseményig a megadott határértéknél kevesebb idő van hátra, akkor új *maintenance begin* eseményt és egy új *failure* eseményt kell generálnunk.

A következő két esemény a karbantartást modellezi. A *maintenance begin* a karbantartás kezdetét modellezi, ezt azért fontos számon tartanunk, mert karbantartás során a gépet legtöbbször le kell állítani, ami pedig hatással van a többi eseményre. Bekövetkezése során

törölnünk kell a *production* eseményeket (hiszen leállítjuk a gépet), a géphez tartozó *failure* eseményt, valamint generálnunk kell egy *maintenance end* eseményt.

Maintenance end a karbantartás végét modellezi, jelzi, hogy elindíthatjuk a gépet. Tehát jelezni kell, hogy a gépe újra működőképes, generálhatunk *production* eseményeket, amennyiben rendelkezésre állnak a szükséges erőforrások (bemeneti helyek), létre kell hoznunk egy új *maintenance begin* eseményt és egy új *failure* eseményt.

A modell megadásánál minden gépnél meghatározhatjuk, hogy melyik eseményt milyen eloszlással kívánjuk modellezni.

A következőkben összevetem a vizsgált modell előnyeit a hátrányaival.

Előnyök:

- a modell felépítése tükrözi a modellezni kívánt rendszert;
- a jól definiált egységek miatt a modell mérete aránylag kicsi;

Hátrányok:

- a modellt felépítő egységek számára nincs egy általánosan elfogadott szerkezet;
- a gépekhez rendelt események között bonyolult a kapcsolat;
- a modell struktúrája kötött, nehéz rendszer specifikus funkciók megadása;

Az előnyöket és a hátrányokat összevetve úgy döntöttem, hogy egy másik megközelítést használok, amely képes az imént felsorolt hiányosságokat pótolni. A későbbi munkáim során célom az itt felsorolt előnyöket a Petri háló szerinti modellezésbe átültetni.

3.3. Petri háló szerinti modellezés – a második tervezési irányvonal

Miután elvettem az előző fejezetben ismertetett irányvonalat, egy olyan modellezési eszközt kerestem, amely képes a korábbi megoldás hiányait pótolni, tehát egy olyan felírást kerestem, amely rugalmas, könnyen módosítható az adott szituációnak megfelelően, az események kapcsolata könnyen definiálható.

Megoldásként a Petri hálót választottam. Mint ahogy az elméleti részben is taglalom, a háló egy kibővített változatát alkalmaztam, Ez a megoldás alacsonyabb szinten közelíti meg a problémát. Ez a tulajdonság pedig lehetővé teszi, hogy a program ne csupán gyártórendszerek vizsgálatra legyen alkalmas, hanem más, Petri hálóval definiálható modellt is képes legyen szimulálni. Ilyen modellek lehetnek például a sorbanállási rendszerek.

A korábbi irányvonallal ellentétben az események kapcsolata nem a programkódban, hanem a Petri hálóban jelenik meg helyek és tüzelők formájában. Ez természetesen azzal jár, hogy a készített modellek nagyobbak és a háló felépítése nem érzékelteti olyan jól a valóságot

rendszer felépítését. A Petri háló precízen leírja a hálóban végbemenő folyamatokat, formális felírást biztosít.

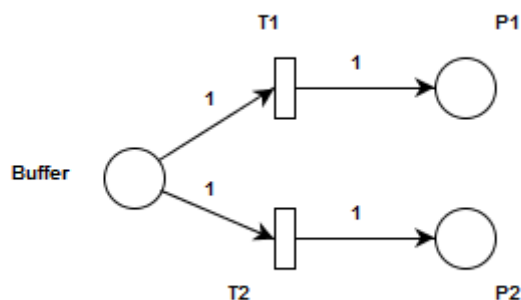
A jövőben a Petri háló struktúráját felhasználva, olyan hálórészleteket kívánok definiálni, amelyek a gyártórendszereknél gyakran előfordulnak, ezzel ellensúlyoznám a modellalkotás nehézségeit. A hálórészletek megalkotásánál cél, hogy könnyen módosítható és bővíthető legyen, hogy a vizsgált gyártórendszerek felépítését minél pontosabban megadható legyen.

3.4. Konkurencia kezelése Petri hálóval

A konkurencia kezelése fontos része különböző rendszerek modellezésének és tervezésének, ennek megvalósítása gyakran nem magától értetődő főleg bonyolultabb rendszerek esetén. Ebben a rövid fejezetben két esetet vizsgálok meg, mely során bemutatom a korábban bemutatott memory policy egy alkalmazási lehetőségét.

1. példa:

Vegyük a következő modellt: két nem exponenciális késleltetésű tüzelő azonos bemeneti hellyel rendelkezik, ezt mutatja a következő ábra (6. ábra).

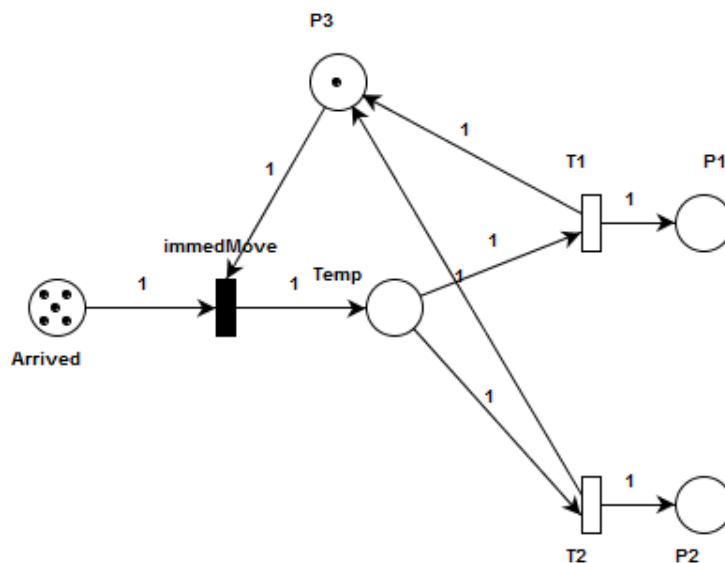


6. ábra - A példa alapfelépítése

A célunk egy olyan modell kialakítása, melynél az egyik tüzelő tüzelése esetén a másik elveszíti az addig végzett munkáját (restart mechanizmus). Korábbi munkáim során nem volt szükséges ezzel foglalkoznom, mivel az exponenciális késleltetésű tüzelők örökifjú tulajdonsága megoldotta a problémát, azonban ha nem exponenciális tüzelőket használunk, nem hagyhatjuk figyelmen kívül ezt a szituációt.

Elsőre talán azt gondolhatnánk, hogy a *resampling* memory policy megoldást jelentene, azonban ekkor minden tokeneloszlásban való változás esetén új időpontokat generálunk, vagyis minden tüzelés esetén elveszik a munka. Ez a fenti ábrán még elfogadható, de ha a hálónk több tüzelőből áll, melyek nem befolyásolják a fent ismertetett tüzelők munkáját, akkor láthatjuk, hogy ez a megoldás nem megfelelő.

Egy lehetséges megoldást mutat be a következő ábra (7. ábra). Az ábrán *immedMove* rögtöntüzelőként funkcionál, T1 és T2 esetén késleltetett tüzelőt használunk, *enabling memory* opcióval. Láthatjuk, hogy a Temp helyre csupán egy token léphet be egyszerre (a P3 hely miatt), tehát amikor *immedMove* tüzel, a Temp helyre egy token kerül. Ha T1 és T2 közül bármelyik tüzel, a másik tüzelő elől elmozgatja az egyetlen token, amely hatására továbbra már nem engedélyezett, így a hozzá tartozó munka elveszik (restart mechanizmus miatt). A „nyertes” tüzelése hatására visszakerül a token P3 helyre, ami engedélyezi *immedMove* rögtöntüzelőnk, amely a következő iterációban (késleltetés nélkül) egy token mozgat a Temp helyre.



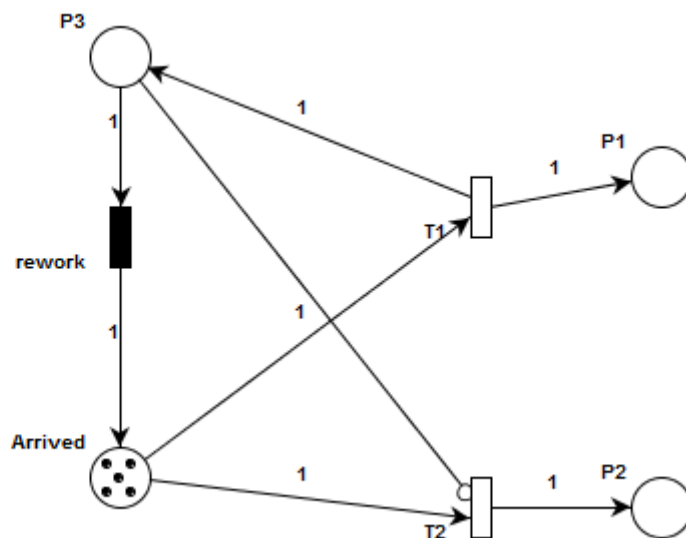
7. ábra - A javasolt struktúra az első példa esetén

2. példa:

A második példa nagyban hasonlít az elsőre, a különbség annyi, hogy csupán az egyik tüzelő veszti el munkáját a másik tüzelése esetén. Esetünkben legyen T1, amely nem veszíti el a munkáját (vagyis nem befolyásolja a T2 tüzelő tüzelése), T2 pedig legyen a tüzelő, amely elveszíti munkáját T1 tüzelése esetén. Vethetünk példaként egy gyártórendszert ahol, ha hibát észlel valaki a gyártás során (T1), akkor a hibás elemet újra kell gyártani (T2).

Az előző megoldásból kiindulva a legegyszerűbb megoldás a T1 tüzelő *memory policy* opcióját *age memory*-ra állítani. Bár ugyanúgy „kiüti” T2 tüzelése T1-et, az *age memory* miatt az elvégzett munka nem veszik el.

Amennyiben ragaszkodunk az *enabling memory* opcióhoz, a következő ábra (8. ábra) szemléltet egy alkalmas megoldást.



8. ábra - A javasolt struktúra a második példa esetén

Az ábrán látható tokeneloszlásban mindkét tüzelő engedélyezett. T2 tüzelése esetén T1 továbbra is engedélyezett marad (amennyiben persze van token az *Arrived* helyen). Amennyiben viszont T1 tüzel, P3 helyre egy token kerül, amelynek hatására T2 továbbra már nem engedélyezett, hiszen T2-höz P3 egy tiltóéllal kapcsolódik, ezért az enabling memory opció hatására a végzett munka elveszik. T1 tüzelését követő iterációban a *rework* rögtöntüzelő P3-ból elmozgatja a token, így újra engedélyezetté válik.

4. Program felépítése és működése

Ebben a fejezetben a program felépítését és működését ismertetem. Először a szimuláció folyamatáról és a használható metódusokról írok egy ismertetőt, ezután az eseménysort és az eseménysort érintő fontosabb függvényeket, az események kezelését, a memory policy opció hatását mutatom be. Majd felvázolom a tüzelőknél megadható paraméterek szerepét, valamint a tüzelők hierarchikus kapcsolatát. Végül kitérek a szimuláció szempontjából fontos metódusok bemutatására.

4.1. Szimuláció:

Ebben az alfejezetben összefoglalom a program szimulációt érintő legfontosabb részeit. A programban használt szimuláció, az elméleti részben áttekintett eseményvezérelt szimuláció technikáját alkalmazza. Alapvetően hat fő technika áll rendelkezésünkre [9], ebből munkámhoz kettő módszert vettem alapul:

- replikáció/törlés megközelítés;
- batch átlagok megközelítés.

Mindkettő metódus ugyanazt a szimulációs logikát alkalmazza (melyeket a fejezet későbbi részében fejtek ki) csupán a futási hossz, annak meghatározásának módja, a replikációk száma valamint a statisztikai mérések módja különbözik.

replicationDeletionSimulation:

Alapvetően az elméleti részben tárgyalt termináló szimuláció és replication/deletion megközelítést használtam kiindulásként. A felhasználó a paraméterfájlban megadhatja:

- a futások hosszát,
- a felfűtési idő hosszát,
- a relatív pontosság értékét,
- azon helyek nevét, ahol az egyes futások végén a konkrét tokenszámot vizsgáljuk,
- azon helyek nevét, ahol az egyes futások alatt mért átlagos tokenszámot mérjük.

A metódus először betölti a paraméterfájlt, majd ez alapján felépíti a Petri hálót, beállítja a kezdő feltételeket (vagyis a helyeknél beállítja a kezdő tokenszámot), rögzíti a futási paramétereket, a kezdő tokeneloszlás alapján meghatározza a lehetséges eseményeket, majd megkezdődik a tényleges szimuláció. Először a felfűtési idő hosszáig futtatjuk a szimulációt mérések nélkül, majd a futás végi rendszer állapotát eltároljuk (helyeknél lévő tokenszámot), ezt az állapotot használjuk kezdési feltételnek a replikációk vizsgálatánál.

Az egyes futások (replikációk) addig tartanak, amíg a szimulációs idő át nem lépi a paraméterként megadott futási időt. Az egyes replikációk végén az adott futás adatait felhasználva frissítjük a mintaátlag valamint a minta szórásnégyzetének értékét. Mivel minden iterációban elvégezzük ezeket a számításokat, ezért számítási költségüknek optimálisnak kell lennie. Ezt szem előtt tartva a következő képleteket használom a programban:

Mintaátlag:

$$\bar{x}_n = \frac{(\sum_{i=1}^{n-1} t_i) \cdot \bar{x}_{n-1} + t_n \cdot x_n}{\sum_{i=1}^n t_i}$$

ahol t_i az egyes replikációk hossza, $\bar{x}_{n-1}$ a megelőző iteráció átlaga, x_n pedig az aktuális futás értéke. Láthatjuk, hogy a megelőző iteráció átlagát az addig futtatott replikációk összhosszával súlyozzuk, az aktuális iteráció értékét pedig az aktuális futás hosszával.

Szórásnégyzet:

$$S_{n+1}^2 = \frac{\sum_{i=1}^n t_i}{\sum_{i=1}^{n+1} t_i} S_n^2 + \frac{t_{n+1}(x_{n+1} - \bar{x}_n)^2}{\sum_{i=1}^{n+1} t_i} + d^2 - 2d\bar{x}_{n+1} + 2d\bar{x}_n$$

a jelölések hasonlóak a mintaátlagnál használtakkal, azzal a különbséggel, hogy az aktuális iterációt $n+1$ indexszel jelöljük, továbbá $\bar{x}_{n+1} = \bar{x}_n + d$.

Miután megkaptuk a mintaátlag és a szórásnégyzet frissített értékét, ezeket felhasználva a becsült paraméterre számolhatunk konfidencia intervallumot, valamint vizsgálhatunk pontosságot is. Pontosság vizsgálatánál a következő feltétel teljesülését vizsgálom:

$$\frac{z_{\alpha}}{2} \cdot \frac{S_n}{\sqrt{n}} < \bar{x}_n \cdot e$$

ahol a bal oldalon a hibahatár, jobb oldalon pedig a mintaátlag ($\bar{x}$), és a felhasználó által megadott relatív pontosság (e) szerepel. A képletben szereplő n értékét a replikációk száma adja meg.

A pontosság vizsgálatát minden replikáció végén elvégzem (a frissített mintaátlag és szórás értékével), amint a feltétel teljesül, a becslést megközelítőleg pontosnak tekinthetjük, ezért a szimulációt leállíthatjuk.

A replicationDeletionSimulation metódust használhatjuk termináló szimuláció végzésére, ha a felfűtési időt nullára, a replikációk hosszát pedig az általunk vizsgálni kívánt időhosszra állítjuk. Ebben az esetben a helyeknél tipikusan a konkrét tokenszámra végzünk becslést.

Végezhetünk stady-state szimulációt is, ebben az esetben állítsunk be egy megfelelően hosszú felfűtési időt, valamint futási időként adjunk meg egy ennél jóval nagyobb értéket. Az egyensúlyi vizsgálatnál az egyes replikációknál a konkrét tokenszám helyett használjuk az átlagos tokenszám mérést.

batchMeansSimulation metódus:

A metódus alkotása során az elméleti áttekintőben részletezett batch átlagok megközelítést vettem alapul. A paraméterfájlban megadható:

- a batchek hossza,
- a felfűtési idő hossza,
- a relatív pontosság értéke,
- azon helyek neve, ahol az egyes futások alatt mért átlagos tokenszámot mérjük.

Szimuláció során az előbbiekhez hasonlóan járunk el: betöltjük a paraméterfájlt, majd ez alapján felépítjük a Petri hálót, beállítjuk a kezdő feltételeket (vagyis a helyeknél a kezdő tokenszámot), rögzítjük a futási paramétereket, a kezdő tokeneloszlás alapján meghatározzuk a lehetséges eseményeket, majd megkezdődjük a tényleges szimulációt. A felfűtési idő hosszágig futtatjuk a szimulációt mérések nélkül, majd futás végi rendszer állapotát eltároljuk és ez alapján kezdjük a tényleges szimulációt.

Az elméleti áttekintőben leírtakhoz hasonlóan egy futtatást végzünk, batchenként végezzük a mérést (az átlagszámítást), és ezeket a méréseket használjuk a vizsgált rendszerparaméter becslésére. A statisztikai számítások képletei megegyeznek a replicationDeletionSimulationnál leírtakkal, azzal a különbséggel, hogy ahol korábban a replikációk mért értékei szerepeltek ott a batchek mért értékeit használjuk. Tehát például a mintaátlagnál t_i az egyes batchek hossza, $\bar{x}_{n-1}$ a megelőző batchek átlaga, x_n az aktuális batch értéke, $\sum_{i=1}^{n-1} t_i$ pedig az aktuális batchet megelőző batchek hosszának összege (vagyis a batch előtti rendszeridő). Szórásnégyzet számításánál és a pontosság vizsgálatánál is hasonlóan járunk el. További különbség, hogy pontosságvizsgálatnál n értéke a batchek számától függ. A szimuláció a korábbiakhoz hasonlóan addig folytatódik, amíg a fenti feltétel nem teljesül. Mivel azonban csupán egy futtatást végzünk, ezért addig folytatjuk a futtatást (és generálunk batcheket), amíg a feltétel nem teljesül.

Ezzel a módszerrel csupán egyensúlyi paramétereket vizsgálhatunk, azonban mivel a programban a felfűtési időszak pontosabb meghatározására nincs jelenleg lehetőség, ezért a egyensúlyi paraméterek becslése várhatóan megbízhatóbb, hiszen a felfűtési időszak csupán a futtatás elején okozhat eltéréseket, ez pedig az egész futtatást tekintve gyakran nem jelentős.

4.2. Eseménysor

A program az elméleti összefoglalóban ismertetett eseményvezérelt szimuláció technikáját alkalmazza, tehát mindig a szimuláció legkorábban bekövetkező eseményhez „ugrik”. Az eseménysorban tárolom a rendszer eseményeit bekövetkezési idő szerint rendezve. Mivel a szimuláció működése az események és sorrendjük helyességétől függ, ezért elengedhetetlen az eseménysor folyamatos frissítése, hogy az aktuális rendszer helyes állapotát reprezentálja.

Eseménysor frissítése (updateQueue):

Az eseménysor frissítésére a háló inicializálása és minden egyes tüzelés után sor kerül. Inicializálás során szükség van frissítésre, hiszen meg kell határozzuk a kezdeti tokeneloszlásból, hogy mely tüzelők engedélyezettek. Tüzelés során pedig azért van szükségünk frissítésre, mert a megváltozott tokeneloszlás a tüzelők engedélyezettségét befolyásolhatja.

Először a rögtöntüzelők halmazát vizsgálom meg hiszen, ezek mindig elsőbbséget élveznek a késleltetett tüzelőkkel (timed transition) szemben. Frissítés során az engedélyezett rögtöntüzelők közül a legnagyobb prioritású számára generálok egy eseményt az aktuális szimulációs idő paraméterével. Amennyiben legnagyobb prioritással több tüzelő is rendelkezik, akkor ezek közül egyet sorsolok a tüzelők súlyát is figyelembe véve.

A rögtöntüzelők után a késleltetett tüzelők (timed transition) halmazát vizsgálom meg. Ezek közül az exponenciális tüzelőket külön kiemelem, mivel ezeknél nem kell a memory policyvel foglalkoznom, valamint figyelembe kell vennem a kiszolgálás típusát is.

Az exponenciális tüzelőknél először az engedélyezettséget állapítom meg, ha nem engedélyezett, akkor a tüzelőhöz tartozó eseményt eltávolítom az eseménysorból (ha van benne ilyen esemény). Ha azonban engedélyezett, akkor tovább vizsgálom a tüzelőt, megvizsgálom a kiszolgálás típusát is:

- exkluzív kiszolgálás esetén megvizsgáljuk, hogy van-e a sorban a tüzelőhöz tartozó esemény:
 - ha van, akkor nincs teendőnk,
 - ha nincs, akkor generálunk számára egy eseményt;
- végtelen kiszolgálás esetén meg kell vizsgálni, hogy van-e az eseménysorban a tüzelőhöz tartozó esemény:
 - ha van, akkor a tüzelő engedélyezési fokát kell mérnünk:
 - ha az engedélyezési fok megváltozott, akkor törölni kell az eseménysorban lévő eseményt, és az új engedélyezési fok szerint újat generálni.
 - ha az engedélyezési fok nem változott, akkor nincs teendőnk;
 - ha, akkor generálunk egyet az aktuális engedélyezési fok szerint

Az exponenciális késleltetésű tüzelők után a többi késleltetett tüzelőt tekintjük át. Ezeknél a tüzelőknél még nem valósítottam meg a különböző kiszolgálási típusokat, viszont a memory policy tulajdonság már jelen van, így ezeket számításba kell venni.

A memory policy szerinti három kategóriába kerülhetnek a tüzelők.

Resampling esetén függetlenül az engedélyezettségtől, rögtön töröljük a tüzelőhöz tartozó eseményeket. Amennyiben a tüzelő engedélyezett, generálunk számára egy új eseményt.

Enabling memory opció esetén először az engedélyezettségét vizsgáljuk:

- ha a tüzelő nem engedélyezett, akkor a hozzá tartozó eseményt töröljük (amennyiben van ilyen az eseménysorban);
- ha a tüzelő engedélyezett, megvizsgáljuk, hogy van-e hozzá tartozó esemény az eseménysorban
 - ha van, akkor nincs teendőnk;
 - ha pedig nincs, akkor generálunk számára egy eseményt.

Age memory opciónál is legelőször az engedélyezettséget nézzük:

- ha engedélyezett, akkor megvizsgáljuk, hogy van-e a tüzelőhöz tartozó esemény az eseménysorban.
 - ha nincs, akkor megvizsgáljuk, hogy a tüzelő munkáját korábban megszakítottuk-e (ezt a tulajdonságot a tüzelőnél tárolom).:
 - ha igen, akkor a korábbi munkáját folytatja, vagyis a hátralévő idő szerint hozunk létre eseményt;
 - ha nem, akkor egy új eseményt generálunk számára, melynek időpontját csupán a tüzelőnél megadott eloszlás befolyásolja;
 - ha van, akkor nincs teendők, (hiszen legkésőbb a szimuláció előző iterációjában elvégeztük a szükséges teendőket).
- ha nem engedélyezett, akkor megnézzük, van-e az eseménysorban a tüzelőhöz tartozó esemény:
 - ha van, akkor a tüzelőnél jelöljük, hogy megszakítottuk a munkáját, a munkából hátralévő időt pedig eltároljuk a tüzelőnél. Az eseménysorból pedig töröljük az eseményt.
 - ha nincs, akkor nincs teendők, (hiszen legkésőbb az előző iterációban elvégeztük a szükséges teendőket).

Ezzel a frissítés az aktuális iterációban lezárult.

4.3. Tüzelők (Transition) felépítése

Ebben a fejezetben röviden bemutatom a tüzelők általános felépítését, majd rátérek a különböző fajta tüzelők speciális tulajdonságaira.

Transition

A tüzelők őssztálya a Tranzition osztály, az itt definiált fontosabb metódusok a következők:

- `fire` metódus, amely a tüzelést írja le;
- `isEnabled` metódus meghívásával megtudhatjuk, hogy az adott tokeneloszlásban engedélyezett-e a tüzelő.

Ebben az osztályban tartom számon:

- a tüzelő nevét;
- a bemeni és kimenő helyek listáját, ezeknél a listáknál tartom számon a helyeket a tüzelővel összekötő élsúlyokat is.

ImmedTransition

A Transition osztály leszármazottja, a rögtöntüzelőt írja le. A Transition tulajdonságain kívül megadhatók a következő:

- `priority`, amely a tüzelőhöz rendelt prioritást adja meg. Ennek alapértelmezett értéke egy, nagyobb számok magasabb prioritási szintet jelölnek.
- `weight`, amely a tüzelőhöz rendelt súlyt adja meg. Ennek az alapértelmezett értéke ugyancsak egy.

TimedTransition

Ugyancsak a Transition osztály leszármazottja, minden késleltetett tüzelő ősosztálya. A következő fontosabb tulajdonságokat adja meg:

- `memoryType`, amely a tüzelőhöz rendelt memory policyt adja meg.
- `hasRemTime`, amely azt adja meg, hogy a tüzelő munkájából van-e hátralévő idő, vagyis megszakítottuk-e a tüzelő munkáját. Az `age` memory memory policynél használjuk.
- `remTime`, amely a munkából hátramaradó időt tárolja.

Továbbá egy új `getNextTimeDiff` metódust valósít meg, amely a leszármazott osztályokban játszik nagy szerepet, ugyanis ez a metódus generálja az adott eloszlás szerinti véletlen számokat (időközöket).

DelayedTransition

A TimedTransition leszármazottja, kifejezetten késleltetéssel megadható eloszlások számára. Csupán egy újabb tulajdonságot listáz: `delay`, amely a késleltetés paramétert adja meg.

DetTransition

A DelayedTransition leszármazottja, a determinisztikus tüzelőt írja le. A fix késleltetés mértékét a `delay` adja meg.

ExpTransition

Ugyancsak a DelayedTransition leszármazottja, azonban ez az osztály az exponenciális késleltetésű tüzelőt modellezi. Itt a késleltetésen kívül a kiszolgálás típusát adhatjuk meg, ezt a `serverType` változó tárolja.

NormalTransition

A TimedTransition leszármazottja, tulajdonságként a normáeloszlás két paraméterét adhatjuk meg:

- mean, amely az átlag értékét fejezi ki.
- variance, amely pedig a szórásnégyzet értékét adja meg.

GammaTransition

A TimedTransition leszármazottja, segítségével megadhatunk gamma eloszlást használó tüzelőket. Az őssosztály tulajdonságain kívül a gamma eloszlás két fontos paraméterét adhatjuk meg:

- shape, amely a gammaeloszlás alakparaméterét jelenti.
- rate, amely a gammaeloszlás ráta paraméterét (scale paraméter reciproka) adja meg.

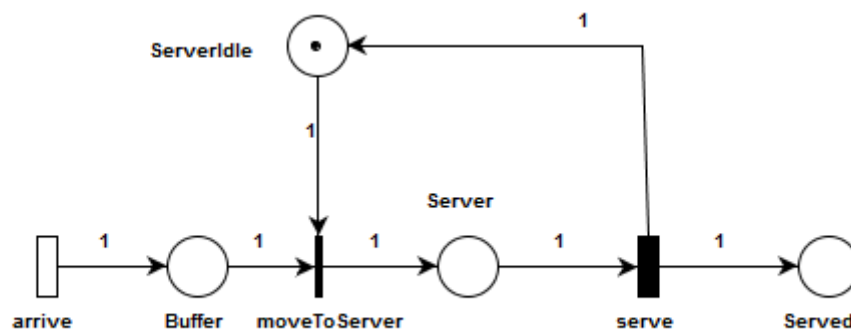
A fenti tulajdonságokon kívül a determinisztikus, exponenciális, normál illetve gamma eloszlást használó tüzelők esetében az időpontok generálására egy különálló Java könyvtárat használtam. Ez a könyvtár a *Stochastic Simulation in Java* (SSJ) nevet viseli, melyet az Université de Montréal fejlesztett [14].

5. Tesztesetek

Ebben a fejezetben néhány tesztesetet mutatok be. Minden tesztnél bemutatom a rendszert leíró Petri hálót, a rendszert és a szimulációt jellemző paramétereket, majd a kapott eredményeket összevetem az analitikus értékekkel, azok hiányában a TimeNET által szolgáltatott adatokat használom.

5.1. M/D/1-es sorbanállási rendszer vizsgálata

Először egy M/D/1-es sorbanállási rendszert vizsgálunk meg, a rendszer hálóját a következő ábrán (9. ábra) láthatjuk. A hálóban található *arrive* tüzelő exponenciális késleltetésű, a *moveToServer* rögtöntüzelő, a *serve* pedig determinisztikus késleltetésű tüzelő. Láthatjuk, hogy a rendszer egy szerverrel és végtelen pufferrel rendelkezik.



9. ábra - M/D/1-es rendszer hálója

A következő táblázat a tüzelők késleltetését adja meg.

1. táblázat - Az M/D/1-es sorbanállási rendszer esetén használt tüzelési paraméterek

Tüzelő	Késleltetés
arrive	10
serve	5

A szimulációt a batch átlagok módszert alkalmazva futtattam, a felfűtési idő hosszát 10000-re, a relatív pontosság értékét 0,001-re, a batchek hosszát 2000-re állítottam. A futtatás során a buffer és szerver helyeknél vizsgáltam az átlagos tokenszámot.

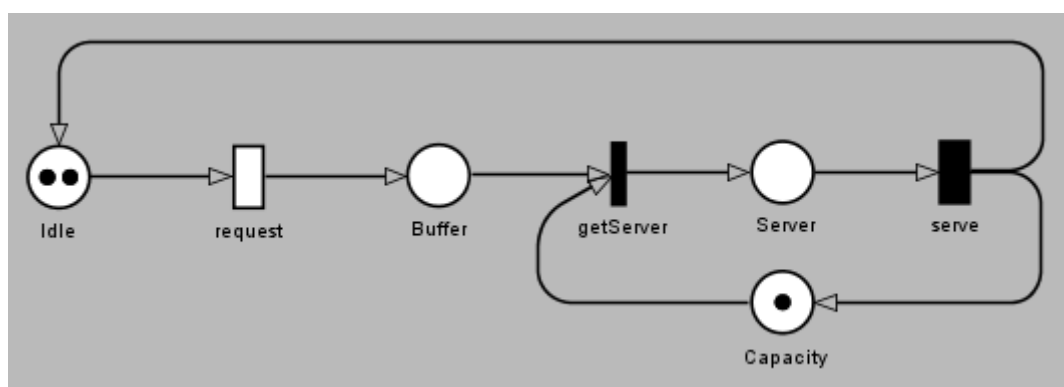
A következő táblázatban a szimulációs eredményeket összevetem az analitikus értékekkel.

2. táblázat - Az M/D/1-es sorbanállási rendszer szimulációs eredményei

Helyek	Szimuláció eredményei	Analitikus eredmények
buffer	0,25029	0,25
server	0,4999	0,5

5.2. M/D/1/2/2-es sorbanállási rendszer vizsgálata

A következő vizsgált rendszer az M/D/1/2/2-es rendszer, amely az előbbi rendszerhez hasonlít, tehát egy szerverrel rendelkezik, a különbség, hogy a rendszerben két igényforrás található, valamint a kapacitása is kettő, így a puffer egy igény tárolására képes. A rendszer hálóját a következő ábrán láthatjuk.



10. ábra - Az M/D/1/2/2-es rendszer Petri hálója

A hálón látható *request* tüzelő exponenciális késleltetésű, a *serve* tüzelő pedig determinisztikus késleltetésű. A következő táblázat a használt tüzelők késleltetéseit adja meg:

3. táblázat - Az M/D/1/2/2-es rendszer szimulációja során használt tüzelők késleltetési értékei

Tüzelő	Késleltetés
request	0,1
serve	0,06

A szimulációt a batch átlagok módszert alkalmazva futtattam, a felfűtési idő hosszát 30000-re, a relatív pontosság értékét 0.001-re, a batchek hosszát 3000-re állítottam. A futtatás során a buffer, szerver és idle helyeknél vizsgáltam az átlagos tokenszámot.

A következő táblázatban a szimuláció eredményeit vetem össze a TimeNET program eredményeivel.

4. táblázat - Az M/D/1/2/2-es rendszer szimulációs eredményei

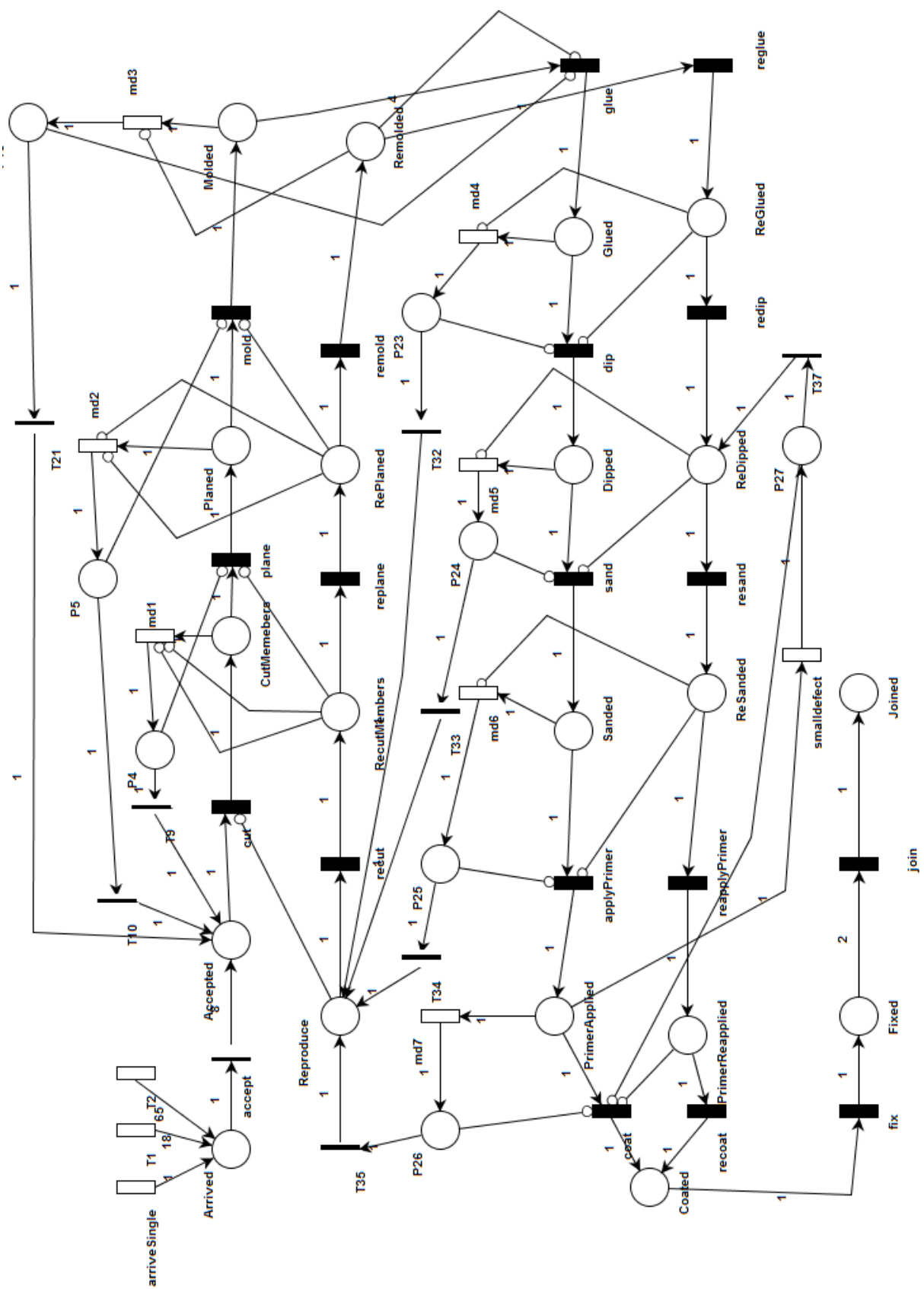
Helyek	Szimuláció eredményei	TimeNET eredményei
buffer	0,170198	0,170186
server	0,686292	0,68618
idle	1,143509	1,143634

5.3. A faablakok gyártási folyamatának modellezése

A következőkben egy bonyolultabb rendszer vizsgálunk, mely faablakok gyártásának folyamatát modellezi (a modell működésének és felépítésének részleteit ld. [15]).

5. táblázat - A DSPN modell tüzelői és azok késleltetései

Tüzelők	Késleltetés
arriveSingle	16,64
arriveAverage	12,07
arriveBig	1392
md1	3,125
md2	15
md3	50
md4	10
md5	60
md6	33,33
md7	22,22
smallDefect	33,33
cut/recut	0,0556
plane/replane	0,0104
mold/remold	0,05
glue/reglue	0,1667
dip/redip	0,0333
sand/resand	0,2
applyPrimer/reapplyPrimer	0,1111
coat/recoat	0,1111
fix	0,3333
join	0,01



11. ábra - Faablakok gyártásának DSPN modellje [15]

Itt a korábbiakkal ellentétben egy tranziens elemzést végzünk, a szimuláció egy naptári évet vizsgál. A vizsgálat során a rendszer év végi állapotra vagyunk kíváncsiak, ezért a szimulációhoz a replicationDeletionSimulation metódust alkalmaztam, a replikációk hosszát pedig 2088-ra állítottam. Az egyes tüzelőkhöz tartozó késleltetéseket a következő táblázat (5. táblázat) foglalja össze.

Az általam elért eredményeket a TimeNET szimulátorával számolt értékekkel vettem össze, mely 99%-os konfidenciaszinten, 1%-os maximális relatív hiba mellett futott (6. táblázat).

6. táblázat - Eredmények összevetése

Helyek	TimeNet	Szimuláció
Accepted	189,81933	195,84572
CutMembers + RecutMembers	0,13467	0,13404
Planed + RePlaned	0,63745	0,64565
Molded + ReMolded	3,25834	3,29334
Glued + ReGlued	0,10666	0,10846
Dipped + ReDipped	0,70611	0,71459
Sanded + ReSanded	0,35967	0,35886
PrimerApplied+ PrimerReapplied	0,35056	0,35666
Coated	584,55711	588,26728
Fixed	0,52422	0,58812
Joined	3021,378	3033,99968

6. Összefoglalás

A munkám hosszú távú célja egy olyan szimulátor kialakítása, amely képes gyártórendszerek modellezésére, annak fontosabb mérőszámait meghatározni. Legyen képes a tervezés folyamatát segíteni, különböző terveket összevetni egymással. Egy ilyen szempont a gyártás maximalizálása a gépek karbantartási gyakoriságának optimális értékének meghatározása. A program további feladata, hogy lehetőséget nyújtson alapvetően különböző gyártórendszereket vizsgálni.

Munkám célja a gyártórendszerek felépítésének és alkotóelemeinek, a rendszert jellemző legfontosabb tulajdonságok megismerése volt. A rendszerek felírásához kerestem egy megfelelő eszközt, megismerkedtem a gyártórendszereknél leggyakrabban előforduló eloszlásokkal, és az adott eloszlásnak megfelelő véletlen szám generálására. Megismerkedtem a különböző eloszlások kapcsán a memory policy fogalmával és a tüzelőknél betöltött szerepével. Kialakítottam egy programot, amely képes egy kibővített Petri hálóval megadott rendszer vizsgálatára, figyelembe véve a tüzelőknél megadott memory policyt is. Végül különböző tesztesetekkel ellenőriztem a program működését.

A következő félévek folyamán folytatom munkámat a félév során lefektetett alapokra építkezve. Fontos minden tüzelőtípusra megvalósítani a végtelen szerver lehetőséget, hiszen ez nagyban bővítheti a modellezhető szituációk körét. Továbbá célom a meglévő felírás bővítése, hogy a modellek megadása minél könnyebb és gyorsabb legyen. Ennek kapcsán mélyebben elmerülök a gyártórendszereknél alkalmazott gépek, eszközök, erőforrások és alkotórészek világában, és a megszerzett tudás alapján további funkciókkal bővítem a programot.

Irodalomjegyzék

- [1] Zimmermann, A., Knoke, M.: TimeNET 4.0: A software tool for the performability evaluation with stochastic and colored Petri nets; user manual. TU, Professoren der Fak. IV (2007)
- [2] <http://simul.iro.umontreal.ca/ssj/doc/pdf/guideprobdist.pdf> Utolsó megtekintés: 2014. 08.22.
- [3] Guy L. Curry Richard M. Feldman: Manufacturing Systems Modeling and Analysis Second Edition 2011
- [4] Richard M. Feldman Ciriaco Valdez-Flores: Applied Probability and Stochastic Processes Second Edition 2009
- [5] Exponential distribution. *Encyclopedia of Mathematics*. URL: http://www.encyclopediaofmath.org/index.php?title=Exponential_distribution&oldid=26442 Utolsó megtekintés: 2014. 08.22.
- [6] <http://www.inf.unideb.hu/valseg/JEGYZET/valseg/node159.htm> Utolsó megtekintés: 2014. 08.22.
- [7] F. Sogandi, A. Amiri : Estimating the Time of a Step Change in Gamma Regression Profiles Using MLE Approach 2014
- [8] <http://www.control.auc.dk/~henrik/undervisning/DES/lec03.pdf> Utolsó megtekintés: 2014. 08.22.
- [9] A. M. Law, W. D. Kelton; Simulation Modeling and Analysis. (Second Edition)McGraw Hill, New York 1991
- [10] Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley and Sons (1995)
- [11] <http://www.inf.ed.ac.uk/teaching/courses/pm/handouts/stochasticpetrinets.pdf> Utolsó megtekintés: 2014. 08.22.
- [12] Pierre Azema, Gianfranco Balbo: Application and Theory of Petri Nets 1997
- [13] Pierre E. Brunet: Modeling and Analysis Applications in Manufacturing System Design and Development (1995)
- [14] <http://simul.iro.umontreal.ca/ssj/indexe.html> Utolsó megtekintés: 2014. 08.22.
- [15] Horváth Ádám, Usability of Deterministic and Stochastic Petri Nets in the Wood Industry: a Case Study, In Proc. of the 2nd International Conference on Computer Science, Applied Mathematics and Applications, pp. 119-127, 2014.

Függelék

A paraméterfájl felépítése

A szimulálni kívánt rendszer Petri hálóval megadott struktúráját, valamint a szimuláció különböző futtatási paramétereit egy különálló XML paraméterfájlban kell megadni.

Az XML fájl gyökérelemeként a `petrinet` taget használjuk, ezen belül adjuk meg a többi elemet. A helyek megadására a `place` taget alkalmazzuk, melynek kötelező jelleggel egy `name`, opcionális jelleggel pedig egy `token` gyerekelemet kell megadnunk. A `name` a hely nevének megadására szolgál, a `token` pedig a kezdő tokenszám beállítására szolgál, melynek alapértelmezett értéke nulla.

A tüzelők megadása már bonyolultabb mivel több típusú tüzelőről van szó. Mielőtt azonban rátérnénk az egyes fajtájú tüzelők felírására, megadom azokat a tulajdonságokat, melyeket mindegyiknél alkalmazunk. Az adott tüzelő gyerekelemeként a helyekhez hasonlóan megadhatunk nevet, ezen kívül meghatározhatunk bemenő és kimenő helyeket az `inplace` illetve `outplace` tagek segítségével. Az `inplace` és `outplace` elemeknél további gyerekelemeként meg kell adnunk a kapcsolódó hely nevét, valamint a kapcsoló él súlyát, ezekre szolgálnak a `name` illetve `arc` tagek. A bemenő illetve kimenő hely neve nyilván csak létező hely neve lehet. Az élsúlyok alapértelmezett értéke egy, valamint nulla élsúllyal jelöljük a tiltó éleket.

Rögtöntüzelőt az `immedtransition` taggel definiálhatunk, gyerekelemként `priority` és `weight` elemeket adhatunk meg. A `priority` taggel a tüzelő prioritását, a `weight` elemmel pedig a tüzelőhöz rendelt súlyt határozhatjuk meg. Ez a két paraméter opcionális, alapértelmezett értékük egy.

Az exponenciális késleltetésű tüzelőnél a késleltetés a `delay` taggel, a kiszolgálás típusa pedig a `servertype` taggel adható meg. Az utóbbi elem `exclusive` vagy `infinite` lehet.

A következő tüzelőknél már előkerül az opcionális `memory policy` tulajdonság is, melyet a `memory` tag határoz meg. Ennek értéke `age`, `resampling` és `enabling` lehet, alapértelmezettként `enabling` opciót használom.

A determinisztikus késleltetésű tüzelőt a `dettransition` tag adja meg, késleltetését az exponenciális késleltetésű tüzelőnél leírtakhoz hasonlóan adjuk meg.

A normál késleltetésű tüzelőnél az átlag és variancia adható meg további paraméterként, ehhez a `mean` illetve a `variance` tag ad lehetőséget.

A gamma késleltetésű tüzelőnél a `shape` és `rate` paramétert adhatjuk meg hasonló nevű tagekkel.

A következőkben rátérek a szimuláció futtatási paramétereire, ezeket a system elem gyerekeiként kell definiálni. Először érdemes megadni a szimulációhoz használt metódust, mely a BatchMeansSimulation illetve a ReplicationDeletionSimulation lehet. A definiáló tag neve method, a megadható értékek pedig batch illetve repdel lehet.

BatchMeansSimulation esetén természetesen meg kell adnunk a batchek méretét, melyre a batch tag ad lehetőséget. Ennél a metódusnál megadhatjuk azokat a helyeket, amelyeknél az átlagos tokenszámot kívánjuk mérni (legalább egy ilyen helynek szerepelnie kell), erre az avgtoken elemet használhatjuk.

Amennyiben azonban a ReplicationDeletionSimulation-t választjuk, batch helyett az egyes replikációk hosszának adhatunk értéket a terminatingtime taget használva. Ezt a metódust használva az átlagos tokenszámon kívül (melynek megadása azonos a BatchMeansSimulationnál leírtakkal) megadhatunk olyan helyeket is ahol a tokenszámot kívánjuk mérni, erre pedig a token taget használhatjuk.

A fentiekén kívül megadhatjuk a felfűtési idő hosszát is a warmuplength elemmel, melynek alapértelmezett értéke nulla. Kötelező jelleggel meg kell adnunk a relatív pontosság értékét az accuracy tag használatával. Opcionális paraméterként megadhatjuk a konfidenciaszintet is, melyre a confidencetype tag ad lehetőséget, melynek 1, 2 és 3 értékeket adhatunk, melyek rendre a 0,95-ös, 0,98-as és a 0,99-es konfidenciaszintnek felelnek meg. Amennyiben nem adjuk meg a 0,95-ös értéket használjuk. Opcionális paraméterként megadhatjuk továbbá a minimális batchek/replikációk számát a minnumofn taggel, alapértelmezett értéként negyvenet definiáltam. A kimeneti fájl elérési útvonalát az outfilepath határozza meg, amennyiben ezt nem adjuk meg, akkor futtatás helyén készül egy a bemeneti fájl nevével azonos (de txt kiterjesztésű) fájl.