2018. tavasz

Rendszermodellezés

kidolgozott jegyzet

Összeállította: Csia Kitti

Tartalomjegyzék

1. előadás: Modellezési alapismeretek 2
2. előadás: Strukturális modellezés 6
3. előadás: Állapot alapú modellezés 12
4. előadás: Folyamatmodellezés 20
5. előadás: Modellek ellenőrzése 28
6. előadás: Vizuális adatelemzés 38
7-8. előadás: Teljesítménymodellezés 48
9. előadás: Modellező eszközök, kódgenerálás 60
10. előadás: Modellek paraméterezése: regresszió, benchmarkok 67

Felhasznált irodalom:

A rendszermodellezés nevű tantárgy hivatalos oldalán (https://inf.mit.bme.hu/edu/courses/remo/materials) található aktuális előadások.

A tanszék által kiadott, felhasználható jegyzetek (http://docs.inf.mit.bme.hu/remo-jegyzet/).

Az előadásokon való részvételem alatt készült saját jegyzetek.

BACK

1. előadás:
Modellezési alapismeretek

1. **Modell**
   - Definíció
     - egy valós vagy *hipotetikus* világ egy részének **egyszerűsített** képe
       - bonyolult rendszer egyszerűsített, kisebb, áttekinthetőbb, véges képe
     - rendszert helyettesíti
     - modell egy kérdés megválaszolására épül
       - probléma szempontjából lényeges szempontok kiemelése
     - modell nem valóság!
     - modell és a modellezett rendszer viszonya
       - modell egyszerűsítheti, összemoshatja, elhagyhatja a modellezett rendszer bizonyos részleteit, részeit
   - Példa
     - modellvasút
     - matematika → gráfok (*úthálózat – leggyorsabb út keresés*)
2. **Diagram**
   - Definíció
     - modell egy nézete, amely annak bizonyos aspektusait grafikusan ábrázolja
       - tehát a diagram csak egy ábrázolási módja a modellnek, amivel olvashatóvá tesszük, szövegessé
3. **Modellezési nyelvek**
   - Célja: kommunikáció gép-gép, ember-gép, ember-ember között
   - Definíció
     - modellezési nyelv négy része:
       - ***absztrakt szintaxis/metamodell***
         - meghatározza nyelv elemeinek típusait, viszonyait
         - pl. gépek a modell belső tárolására

[K1] megjegyzést írt: A zölddel szedett, dőlt szövegek általában egy addig elő nem fordult fontos szó, definíció vagy tétel neve, melynek ismerete fontos.

A „rendszer”.

[K2] megjegyzést írt: Pl. ha egy modellnek végtelen sok állapota van, abból egyet kiemelni egy véges állapotot kapunk.

[K3] megjegyzést írt: Modellezési nyelv modellje. Metamodellnek is lehet modellje, és annak is lehet metamodellje, és így tovább. Ez egészen addig megy, amíg az egyik metamodellt már egy szabványos modellezési nyelv segítségével írunk fel.
UML-ben van önmagát leíró modell is, önmaga saját metamodellje.

- ***konkrét szintaxis***
  - szöveges/grafikus jelölésrendszert definiál elemtípusokhoz, kapcsolatokhoz
- ***jólformáltsági kényszer***
  - modellnek milyen követelményeknek kell megfelelnie
  - tovább szűri a lehetséges modellek körét
    - » pl. azonos nevű elemek tiltása
- ***szemantika***
  - absztrakt szintaxis által megadott nyelvi elemek jelentését definiáló szabályrendszer
    - » megadja, hogy pontosan hogy működik, mit jelent a modell
    - » jelentést ad a nyelvi elemeknek

- Példa
  - szöveges: Verilog, VHDL, Java…
    - előny: könnyebb modellt építeni
  - grafikus: 3D tervezők, UML diagramok…
    - előny: könnyebb olvasni

4. Nyílt és zárt világ feltételezése

- Definíció
  - ***zárt világ feltételezése***: minden állítás, amiről nem ismert, hogy igaz, hamis
  - ***nyílt világ feltételezése***: egy állítás annak ellenére is lehet igaz, hogy ez a tény nem ismert
  - különbség: nyílt világ elismeri, és használja az *ismeretlen* fogalmát, zárt világ minden tudást ismertnek tekint
- Példa
  - zárt világ
    - olyankor alkalmazzuk, mikor a rendszernek minden szükséges információ a rendelkezésére áll
    - metrójáratok - tudjuk, hogy az adott helyen van-e megálló vagy nincs
    - minden modell zárt világ!

- nyílt világ
    - amikor nem feltételezzük, hogy minden információ a rendelkezésünkre áll
    - orvosi nyilvántartás – beteg allergiás valamire, annak ellenére, hogy nincs a nyilvántartásában

5. **Rendszer és környezete**

- Definíció
    - ***rendszer***
        - egyértelműen definiálja a határait
        - határon belül eső dolgok
    - ***környezet/kontextus***
        - rendszerre ható tényezők összesége
        - határon kívül eső dolgok
        - *releváns környezeti elemek*
            - rendszerrel közvetve vagy közvetett módon kapcsolatban áll
        - *irreleváns környezeti elemek*
            - nincs kapcsolatban a rendszerrel
        - *fekete doboz*: teszteléskor a rendszer belső felépítését és viselkedését nem ismerjük
        - *fehér doboz*: ismerjük

[K4] megjegyzést írt: Szokás a környezetet is modellezni, mikor egy rendszert tesztelnek. Például autók tervezésénél szimulációkkal. Prototípus elhelyezése szélcsatornába stb.

6. **Absztrakció és finomítás**

- Definíció
    - ***finomítás***
        - modell részletezése/ pontosítása
        - környezet szempontjából akár helyettesíteni is tudja az eredeti modellt
        - több végkimenetelt eredményezhet

- ***absztrakció***
  - finomítás inverz művelete
  - modell részletezettségének csökkentése
  - modellezett ismeretek egyszerűsítése
  - finomítás után az eredeti modell absztrakcióval kapható vissza
  - egy végkimenetelt eredményez

- Példa
  - közlekedési lámpa
    - absztrakt modell: *tilos* és *szabad*
    - finomított modell: *szabad – zöld*, *tilos – piros, kettő közötti – sárga*

BACK

2. előadás:
Strukturális modellezés

1. **Strukturális modell**
   - Definíció
     - a rendszer felépítésére vonatkozó tudás
     - a rendszer felépítését reprezentálja
       - alkotórészei
       - azok tulajdonságai
       - egymással való viszonya alapján
     - *statikus*, tehát:
       - változhat az idő során – metróhálózat fejlődése
       - de időben nem ír le változásokat – miként mozognak a szerelvények
     - kiindulópontja egy (rész)rendszer, amelyet a része *reláció* mentén alkotórészekre bont, ezek lehetnek
       - további *részrendszer*ek
       - tovább nem bontott (*elemi*) *komponens*ek
   - Célja:
     - rendszer részekre bontása
       - kisebb egységeket könnyebb tervezni
       - részegységek újrahasználhatósága
       - általános célú komponensek használata
     - létező rendszer dokumentálása – „*rendszertérkép*”
     - adatszerkezet megalkotása – milyen információt kezel
2. **Tulajdonságmodell**
   - Definíció
     - ***jellemző***
       - modell által megadott *parciális függvény*
       - modellelemeken értelmezünk

[K5] megjegyzést írt: Strukturális modell statikus, viszont pl. a viselkedésmodell a rendszer működését, változását írja le.

[K6] megjegyzést írt: Változást nem ír le.

[K7] megjegyzést írt: Tulajdonságmodell mögötti matematikai struktúra.

- Példa
  - *kapacitás* jellemzőhöz tartozó függvény

    e: modellelem azonosítója

    n: nemnegatív egész szám

    $kapacitás(e) \rightarrow n$

    pl.: $kapacitás(T2) = 60$
  - *funkció* jellemzőhöz tartozó függvény

    e: modellelem azonosítója

    t: egy {*metró járműtelep, autóbuszgarázs*} halmaz eleme

    $funkció(e) \rightarrow t$

    pl.: $funkció(T4) = autóbuszgarázs$
  - fentiekhez hasonlóan bevezetünk egy *vágányhossz* jellemzőhöz tartozó parciális függvényt, csak bizonyos modellekre értelmezhetünk
  - csak akkor vesz fel értéket, ha a funkció attribútum értéke: *metró járműtelep*
  - tehát a *vágányhossz* jellemző csak a *metró járműtelep* típusú elemekre értelmezett
  - ***típus***
    - meghatározza, milyen más jellemzők értelmezettek az adott modellelemre
    - milyen más modellekkel lehet kapcsolatban
    - többi jellemző: *tulajdonság*
  - ***példány***
    - egy adott t típus példányainak nevezzük azon modellelemeket, amelyek típusa t

[K8] megjegyzést írt: Jelen anyagban az egyszerűség kedvéért feltételezzük, hogy a modellelemeknek pontosan egy típusa van.

3. Gráfmodell

- Definíció
  - ***gráf***
    - egy gráf rendezett pár, $G = (V, E)$
    - $\boldsymbol{V}$: nem üres halmaz
      - elemei: pontok/csúcsok
      - csúcsok száma: $v(G)$
      - csúcshalmaz jelölése: $V(G)$
    - $\boldsymbol{E}$: V-ből képezhető párok egy halmaza
      - elemei: élek
      - élek száma: $e(G)$
      - élhalmaz jelölése: $E(G)$
  - ***típusgráf***
    - egy olyan gráf, amelyben minden csomóponthoz egy típuscsomópont, minden éltípushoz egy típusél tartozik
  - ***példánygráf***
    - elemei a típusgráf csomópont – és éltípusainak példányai
    - minden él forrása és célja rendre az éltípus forrásának és céljának példánya
    - rendszert alkotó elemek egymáshoz való viszonyának leírása

[K9] megjegyzést írt: Egy rendszer metamodellje tartalmazza a típusgráfot, az egyes típusok közötti kapcsolatokat, illetve a további megkötéseket is. Pl. jóformáltsági vagy multiplicitási kényszerek.

4. Hierarchia

- Definíció
  - ***dekompozíció (faktoring)***
    - egy rendszer kisebb komponensekre bontása
    - azok könnyebben érthetők, fejleszthetők, karbantarthatók
    - *a rendszer hierarchiája a rendszer dekompozíciójával állítható elő*
    - részrendszer dekompozícióval további részekre bontható

- *helyes dekompozíció*
  - ha a dekompozícióval kapott rendszer minden elemének megfeleltethető az eredeti rendszer valamelyik eleme
  - az eredeti rendszer minden eleméhez hozzárendelhető a dekompozícióval kapott rendszer egy vagy több eleme

[K10] megjegyzést írt:

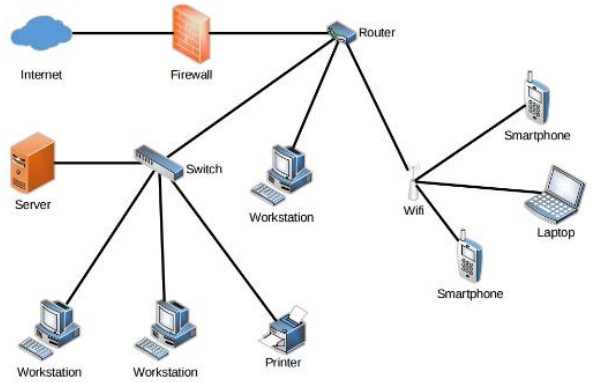
5. Nézetek

- strukturamodellekből különböző nézeteket állíthatunk elő
- tulajdonságmodelleken leggyakrabban használt műveletek:
- Definíció
  - ***szűrés***
    - a modell elemein kiértékelünk egy feltételt
    - azokat tartjuk meg, amelyek megfelelnek a feltételnek
      - tulajdonságmodell esetén a szűrés az elhagyott modellelemek, modell sorai
      - gráfnál a gráf csúcsai, élei
- Példa
  - *tulajdonságmodell szűrése*
    - szeretnénk megtudni, hogy mely telephely képes legalább 100 jármű befogadására
  - *gráfmodell szűrése*
    - soroljuk fel az M2-es metró megállóit
- Definíció
  - ***vetítés***
    - a modell egyes jellemzőit kiválasztjuk és a többit elhagyjuk
    - az is érvényes, ha az összes elemet megtartjuk
- Példa
  - olyan kimutatást adjunk, amely csak a budapesti autók színét tartalmazza

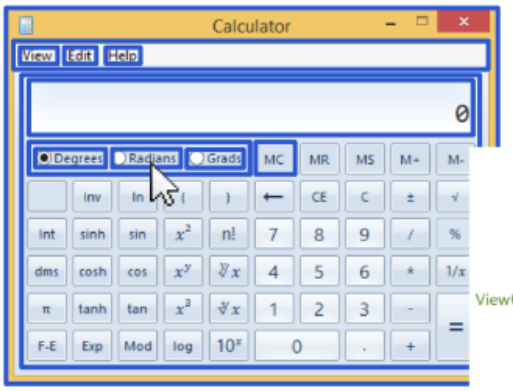
6. Strukturális modellezési technikák

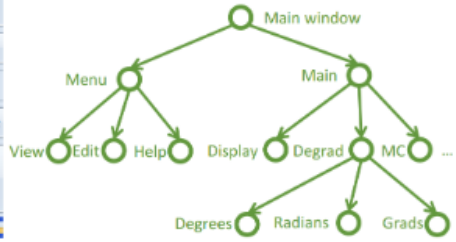
- ***hierarchia modellezés***
  - (*gráfoknál*) két részre tagolódik
    - modell *szerkezeti váz*át tartalmazó hierarchia (fa/erdő)
      - alkotóelemek rész-egész viszonyát mutatja
      - élcímekre szűrve is egy vagy több fa marad
    - *kereszthivatkozás* élek
      - *tartalmazási rend*től függetlenül, körmentesség korlátozása nélkül köthetnek össze elemeket
  - metamodell megmondhatja, mely éltípusok példányait fogjuk a szerkezeti vázat alkotó *tartalmazási él*eknek tekinteni
- elkészítési sorrend/megközelítés
- Definíció
  - ***top-down***
    - a modellezés során fentről lefelé (összetett rendszertől az alkotóelemig) haladva építjük
    - alaplépése a dekompozíció
  - fontosabb jellemzők:
    - részrendszer tervezésekor a szerepe már ismert
    - „félidőben” még nincsenek teljesen működő részek
    - részek problémái, igényei később derülnek ki
  - ***bottom-up***
    - alulról felfelé haladva (elszigetelt alkotóelemekből az összetett rendszer felé) építjük
    - alaplépése a kompozíció
    - egész rendszer összeszerkesztése külön modellezett vagy fejlesztett részrendszerekből
  - fontosabb jellemzők:
    - rendszer részei önmagukban kipróbálhatók, tesztelhetők
    - részleges készültségnél könnyebben előállítható a rendszer prototípusa
    - nem látszik előre a rész szerepe az egészben
- van „aranyközépút” is a kettő között

- Példa
  - számítógép hálózat



  - grafikus felhasználói felület (GUI) is hierarchikus modell





BACK

3. előadás:

Állapot alapú modellezés

[K11] **megjegyzést írt:** Közúti jelzőlámpa fokozatosan kidolgozott modelljén keresztül mutatjuk be ezt a modellezést.

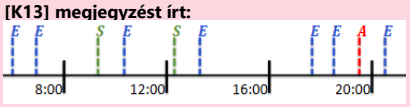
1. Egyszerű állapotgépek

[K12] **megjegyzést írt:** Digitális technikából ismert Mealy-automata formalizmus.

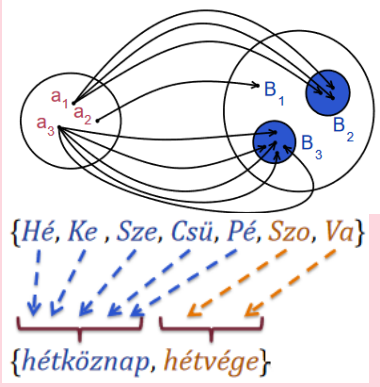
- Definíció
  - ***állapottér***
    - egymástól megkülönböztetett rendszerállapotok halmaza
      - nem minden állapothalmaz lehet állapottér
    - minden időpontban pontosan egy eleme jellemzi a rendszert (*pillanatnyi állapot)*
    - elemeit állapotoknak nevezzük, két kritériumnak kell megfelelniük – **teljesség, kizárólagosság**
    - példa: $\{hétfő, kedd, szerda, csüt. péntek, szom. vas.\}$
  - ***teljesség***
    - minden időpontban az állapottér legalább egy eleme jellemzi a rendszert
    - példa: $\{hétfő, kedd, péntek\}$ → nem teljes
  - ***kölcsönös kizárólagosság***
    - minden időpontban az állapottér legfeljebb egy eleme jellemzi a rendszert
    - ellenpéldák nem alkalmasak állapottérnek
    - példa: $\{hétköznap, hétvége\}$ → nem kizárólagos, holott teljes
  - *rendszer pillanatnyi állapota*
    - állapottér egyetlen eleme, amelyik abban az időpontban jellemző a rendszerre
    - példa: ma $szerda$ van
  - *rendszer kezdő állapota*: olyan állapot, amely a vizsgálatunk kezdetekor (pl. $t = 0$) pillanatnyi állapot lehet

- ***események***
  - *esemény*
    - pillanatszerű változás (a rendszerben vagy in/outputon)
    - példa: egyszerre ≤ 1
  - *eseményfolyam*
    - pl. in/output adatforrás
    - példa: telefon értesítés jelzése
  - *eseménytér*
    - megengedett események
    - beolvasható input értékek
    - kibocsátható output értékek
    - példa: $\{Email, SMS, Alacsony\ töltöttség\}$
- ***állapotfinomítás és -absztrakció***
  - az állapottéren mint halmazon végzett halmazfinomítás illetve halmazabsztrakció
  - eredménye egy újabb állapottér
  - *állapotfinomítás*
    - tervezés előrehaladása, több megvalósítási részlet
    - specializáció, kiegészítés
    - több rendszer együttes vizsgálata
    - több információ
  - *állapotabsztrakció*
    - hasznos, ha az absztrakt állapotok
      - » egységesek, majdnem ekvivalensek
      - » valamilyen szempontból egyformák az összevont állapotok
    - bizonyos feladatokra kevesebb információ is elég
      - » kisebb, egyszerűbb állapottérrel könnyebb tervezni
      - » tárolás, feldolgozás könnyebb
      - » elrejtett részletek szabadon változtathatóak
      - » szélsőséges eset: *állapotmentes modell*

[K13] megjegyzést írt:



[K14] megjegyzést írt:



- gyakori formája: dekompozíció
- kevesebb információ

▪ ***állapottér vetítése komponensre***

- állapotabsztrakciós művelet
- szorzat állapotteréből egy vagy több komponenst tart meg
- többit elhanyagolja

▪ ***diszkrét állapottér***

- nem létezik folytonos átmenet
- a rendszer pillanatnyi állapota mindaddig állandó, amíg egy pillanatszerű *esemény* hatására másik állapotba kerül
- *állapotátmenetekkel/tranzíciókkal* jól modellezhető
  - megengedi, hogy a rendszer állapotot váltson forrás – és célállapot között

▪ ***tüzelés***

- rendszer új állapota a célállapot lesz
- → egy adott *esemény* válthatja ki
  - speciális esete a *spontán állapotátmenet*, mikor a kiváltó esemény kívülről nem megfigyelhető
- vagy *akció*, amely maguk is válthatnak ki eseményeket

○ „állapot” két jelentéssel bír

▪ *szintaktikai*: állapotgráf egy csomópontja, melyet lekerekített téglalap jelöl (*állapotcsomópont*)

▪ *szemantikai*: állapottér egy eleme

○ Példa

▪ jelzőlámpa egyszerű állapottere:

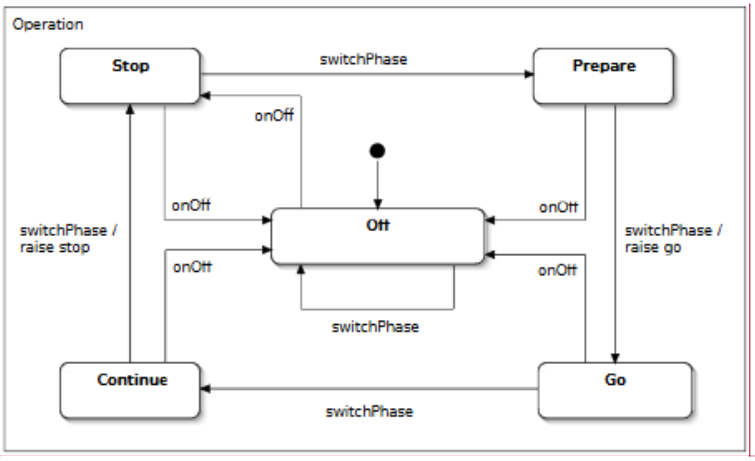
– Off: kikapcsolt állapot
– Stop: piros jelzés
– Prepare: piros-sárga jelzés
– Go: zöld jelzés
– Continue: sárga jelzés

[K15] megjegyzést írt:

$$\{AM,PM\} \times \{1h..12h\} \times \{0m..59m\}$$
$$\cup$$
$$\langle PM, 12h, 08m \rangle$$
$$\downarrow$$
$$\langle PM, 12h \rangle$$

- jelzőlámpa modellje:
- onOff: ki – és bekapcsolás kérése
- swithcPhase: jelzésváltás kérése
- stop: rendszer sárgából vörösbe váltott
- go: rendszer zöldbe váltott

```
interface:
  in event onOff
  in event switchPhase
  out event stop
  out event go
```



[K16] megjegyzést írt: Off, Stop…: rendszer állapotai
Off: kezdőállapot, fekete korongból húzott nyíl
Téglalapok közötti nyilak: állapotok közötti tranzakciók
Nyilakra írt címkék: tranzíciót kiváltó, illetve a kiváltott eseményekre hivatkoznak (Yakinduban esemény kiváltását „raise” jelöli).

- modell tulajdonságai
- Definíció
  - ***determinisztikus***:
    - állapotgépnek legfeljebb egy kezdőállapota van
    - bármely állapotban, bármely bemeneti esemény bekövetkezésekor legfeljebb egy tranzíció tüzelhet
    - egy állapotgép biztos determinisztikus, ha egy állapotátmenete van
  - ***teljesen specifikált***:
    - állapotgépnek legalább egy kezdőállapota van
    - bármely állapotban, bármely bemeneti esemény bekövetkezésekor legalább egy tranzíció tüzelhet
    - minden inputra van szabály

[K17] megjegyzést írt: Yakindu csak az ilyen modellek létrehozását támogatja.

[K18] megjegyzést írt: Rendszer holtpontmentes. Nem tartalmaz olyan állapotot, amelyből nem vezet ki tranzíció.

- **rendszer időbeli viselkedése**
- Definíció
  - ***végrehajtási szekvencia***:
    - állapotok és eseménynek egy (véges vagy végtelen) alternáló sorozat

      $$s_0 \xrightarrow{i_0/o_0} s_1 \xrightarrow{i_1/o_1} \dots$$

    - a rendszer kezdőállapota: s_0
    - a rendszer állapotmenete minden j-re: $s_j \xrightarrow{i_j/o_j} s_{j+1}$
    - egy állapot *elérhető*, ha a rendszernek létezik végrehajtási szekvenciája

2. Hierarchia

- Definíció
  - ***összetett (Harel) állapotgépek***
    - állapotok közös tulajdonságait és viselkedését általánosítja
    - *statechart*/ Harel-féle összetett állapotgép
    - Mealy-féle egyszerű állapotgép + állapothierarchia, ortogonalitás, változók, pszeudoállapotok
- Példa
  - pl. előző ábrába minden eseményhez csatolunk egy reset gombot, mely a stopra ugrik vissza
  - szoftverek: Yakindu, UML
- Definíció
  - ***régió***
    - összetett állapotnál
    - további állapotokat (kezdőállapotot is), tranzíciókat tartalmaz
    - *legfelső szintű régió*: magát az állapotgépet tartalmazza

- ***állapotkonfiguráció***
  - ha egy *tartalmazott állapot* (egyszerű vagy összetett) aktív, akkor az őt tartalmazó összetett állapot is akítv
  - állapotok egy olyan maximális, tovább nem bővíthető halmaza, melyek egyszerre lehetnek aktívak a rendszerben

3. Ortogonális dekompozíció

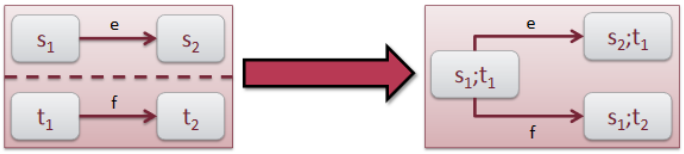
- Definíció
  - ***állapotterek direkt szorzata***
    - komponens állapottereken végzett kompozíciós művelet
    - eredménye egy újabb állapottér (*szorzat állapottér*)
    - szorzat állapotterében
      - komponens állapotterek minden állapot-kombinációjának egy-egy összetett állapot (*állapotvektornak*) felel meg
  - ***szorzatautomata***
    - aszinkron szorzás eredménye
    - ennek a vonatkozó régiójában az állapotok száma a két összeszorzott régió (egyszerű) állapotai számának szorzata
  - *állapottér-robbanás*
    - nagy modelleknél alkalmatlan használat → kezeletlenül nagy modelleket kapunk
  - ***ortogonális állapot***
    - összetett állapot
    - több régióval rendelkezik
    - régiói: ortogonális régiók
    - ezek az egyrégiós összetett állapottal megegyező módon akkor aktívak, ha a tartalmazó állapot aktív
    - aszinkron módon működik
      - tranzíciók külön – külön tüzelnek

[K19] megjegyzést írt: Egy adott esemény bekövetkezésekor az ortogonális régiók tranzíciói egyszerre tüzelnek.

- *versenyhelyzet*: ha a régiók azonos eseményeket dolgoznak fel
  - működés összehangolása megosztott változókon keresztül

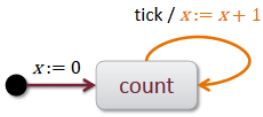
4. Aszinkron szorzat

- Definíció
  - a (Mealy-) állapotgépek aszinkron szorzata (régióknak is nevezett) komponens állapotgépeken végzett kompozíciós művelet
  - szorzat eredménye egy (Mealy) állapotgép, melynek
    - állapottere a régiók állapottereinek direkt szorzata
    - kezdőállapotban az összes régió kezdőállapotban van
    - átmeneti szabályait az összes olyan lépés alkotja, melyben
      - pontosan egy régió állapotátmenetet végez
      - többi régió állapota nem változik



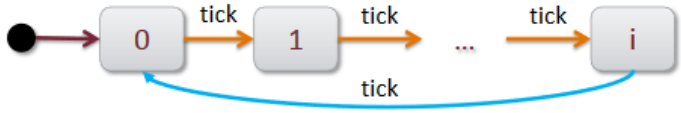
- *változók*
  - végtelen számláló:

  - változóval „x”:

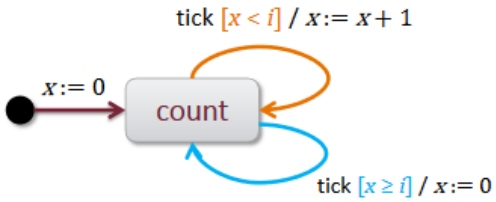


$$(count, \{x \mapsto 0\}) \rightarrow (count, \{x \mapsto 1\}) \rightarrow \ldots$$

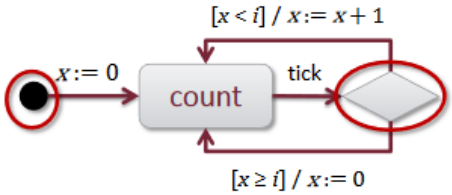
- *változók + őrfeltételek*
  - ciklus számláló:



  - őrfeltételekkel:



- *pszeudoállapot*
  - szemantikailag nem állapot:
    - nincs olyan időpillanat, amikor a rendszert jellemezné
  - szintaktikailag állapot:
    - lehet tranzíció kezdő- vagy célállapot



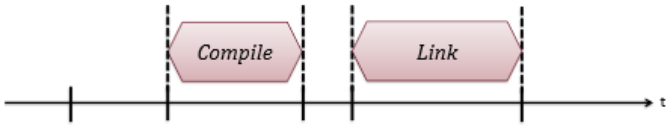
BACK

4. előadás:
Folyamatmodellezés

1. **Folyamatok**
    - folyamatmodellezés célja, rendszer folyamatának leírása
    - viselkedési modellek a rendszert többféle módon jellemzik
        - állapot alapú modellek
            - „miként változhat” a rendszer
            - pillanatszerű események
            - rendszereket az állapotukkal jellemezzük
            - milyen állapotokban lelhető fel a rendszer
                - és nevekkel látja el az állapotokat
            - milyen hatásokra mely állapotból mely állapotba lép át
        - folyamatmodellek
            - „mit csinál” a rendszer
            - tevékenységeknek időbeli kiterjedés tulajdonítása
            - megvizsgálva, hogy mely tevékenységek végezhetők el a másik előtt vagy után
            - rendszer állapotainak jellemzése
                - mely időpontban, mely tevékenység aktív, fejeződött már be
    - Definíció
        - ***folyamat***
            - tevékenységek összessége
            - adott rendben történő végrehajtás valamilyen célra vezet
2. **Folyamatmodellek alapjai**
    - Definíció
        - ***elemi tevékenység***
            - időbeli kiterjedéssel rendelkező tevékenység
            - kezdés és befejezésen túl további részleteket nem modellezünk

- Példa
  - elemi tevékenység
    - C forráskódból futtatható programot szeretnénk
    - forrásállományt le kell fordítanunk „*compile*”
    - fordítóprogramot nem mi készítjük el, csak indítjuk és leállítjuk, nem kell nekünk további részlet
    - időbeli kiterjedéssel bír:
      - háromelemű állapotteret határoz meg
        - » *kezdet* és *vég* időpontban pillanatszerű esemény
        - » kettő között *folyamatban* van

  - ***szekvencia***
    - tevékenységek szigorú végrehajtási sorrendje
    - folyamatmodellek tevékenységeiből folyamatot építenek fel
      - egyes tevékenységek egymáshoz képest mikor hajthatók végre
      - tevékenységek *vezérlési élek/vezérlésifolyamatélek* kötik össze
- Példa
  - C program elemi tevékenysége után a forrásfájlt össze kell linkelni más tárgykódokkal → *linkelés*
  - *jelölés szaggatott vonal az ábráknál*

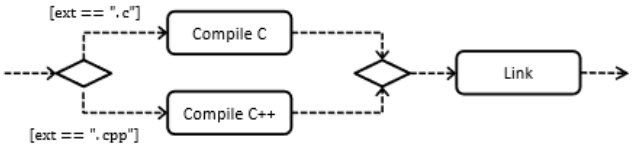


- Definíció
  - *folyamatpéldány*: folyamatmodellben a folyamat konkrét lefutásai
  - ***vezérlési elem/ vezérlési csomópont***:
    - csomópont a folyamatban
    - a folyamatmodell tevékenységei közül választ ki egyet vagy többet végrehajtásra
- Példa
  - bizonyos fájlokra C fordítót, másokra C++ kell hívni
  - elágazással lehet ábrázolni, lehet több *ága* is
  - ***nemdeterminizmus***: ez a modell nem adja meg, hogy melyik ágba kerülünk
    - hasznos, ha emberi döntéstől függ a választás
- Definíció
  - ***őrfeltétel***
    - előzővel ellentétben tehetjük determinisztikussá a választást
    - más modelleknél választási lehetőségek számát csökkentheti
    - folyamatmodellen kívüli tudás alapján kizárjuk a döntés után meglevő ágak némelyikét
    - (*állapotgépeknél is megtalálhatóak*)
- Példa
  - ugyanebben az elágazásban elhelyezünk egy őrfeltételt
  - eldönti helyettünk, hogyha .c végződik a fájl, akkor C-vel fordítja, ha .cpp, akkor C++-szal
- Definíció
  - ***döntési csomópont/decision***
    - csomópont a folyamatban
    - belé érkező egyetlen vezérlési él hatására a belőle kiinduló *ágak* (vezérlési élek) körül pontosan egyet választ ki végrehajtásra
    - őrfeltételekkel összhangban van

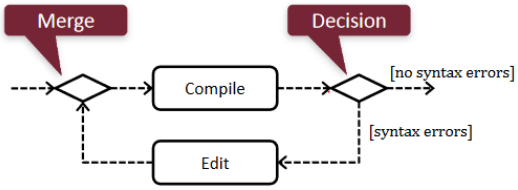
[K20] megjegyzést írt: Melyik ágba kerül a *token*, azaz, melyik lesz épp aktív.

[K21] megjegyzést írt:
Determinisztikusság bővítve:
akkor determinisztikus egy folyamatmodell, ha minden elágazásban végrehajtáskor őrfeltételek kizárják egymást.
Teljesen specifikált:
legalább egy őrfeltétel teljesül, ha nincs őrfeltétel, akkor az állandó „[igaz]” őrfeltételt alkalmaztuk.

- Példa
  - találkozási pont
  - őrfeltétel bármelyik ágra is fut, kivezetésnél ugyanoda érkeznek



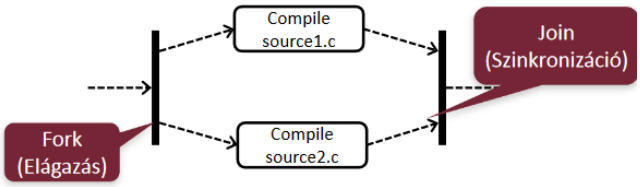
  - speciális eset, ha csak 1 feladat van megadva, másik úgynevezett *üres vezérlési él*
  - kiszedjük C++ fordítást
  - választás opcionális lesz, hogy egyáltalán akarjuk fordítani C-n vagy nem, például, ha módosult a fáj utolsó megnyitáskor vagy nem
- Definíció
  - ***ciklus***
    - folyamatmodell (részlet)
    - elágazás valamelyik ágán az elágazást megelőző merge csomópontba jutunk vissza
- Példa
  - program futásakor találunk fordítási hibát, így a programot ismét le akarjuk fordítani, amíg ki nem küszöböljük
  - ilyenkor ismét visszaugrik a fordítás előtti merge pontba



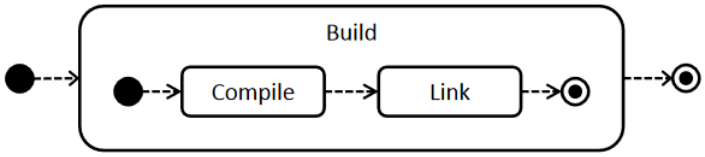
- Definíció
  - ***konkurens***
    - két bekövetkező tevékenység vagy esemény
    - nincs megkötve milyen sorrendben következnek be

- ***fork csomópont***
  - csomópont a folyamatban
  - belé érkező egyetlen vezérlési él hatására belőle kiinduló összes vezérlési él (*párhuzamos folyam*) kiválasztja végrehajtásra
- ***join csomópont (találkozási/szinkronizáló)***
  - csomópont a folyamatban
  - belé érkező összes párhuzamos folyam végrehajtása után kiválasztja a belőle induló egyetlen vezérlési élet

- Példa
  - nincs meghatározva két forrásfájlt milyen sorrendben fordítunk le
  - többmagos processzornál megpróbálható egyszerre is
  - fork csomópontból, vastag vonallal jelölt, indulnak
  - token bármelyikre mehet
  - joinban találkoznak a fordítás után, két tokent ekkor *összeolvasztjuk*
  - itt bevárják egymást a folyamatok



- Definíció
  - ***start (flow begin)***
    - csomópont, a folyamat indító eleme
    - pontosan 1 kimenete van
  - ***cél (flow end)***
    - csomópont, a folyamat befejező eleme
    - pontosan 1 bemenete van

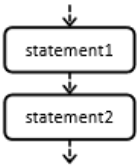


- ***folyamatpéldány***
  - folyamatmodellhez tartozó
  - diszkrét eseménysor
  - események alkotják, folyamatmodell által megszabott időrendben:
    - folyamat kezdete
    - folyamatot alkotó tevékenység kezdete
    - folyamatot alkotó tevékenység vége
    - folyamat vége
  - folyamatnak több folyamatpéldánya is lehet

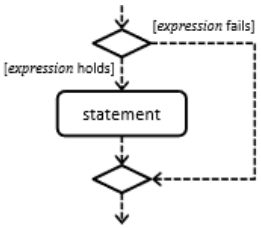
3. Folyamatmodellek felhasználása

- Definíció
  - ***vezérlési folyamat (control flow)***
    - program által meghatározott folyamatmodell
    - elvégzendő lépéseket, végrehajtásukra előírt sorrendet írja le
  - ***imperatív program***
    - programok, melyeket vezérlési folyamot határoznak meg
- C-szerű vezérlési struktúrák vezérlési folyamatai

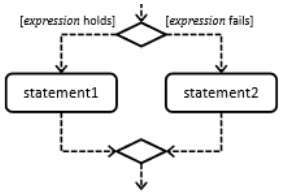
```
<statement1>
<statement2>
```

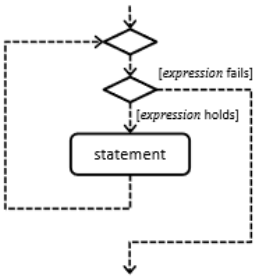



```
if (<expression>)
    <statement>
```



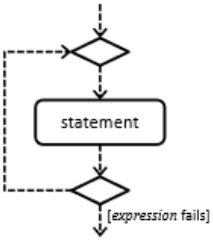
```
if (<expression>)
    <statement>
else
    <statement>
```





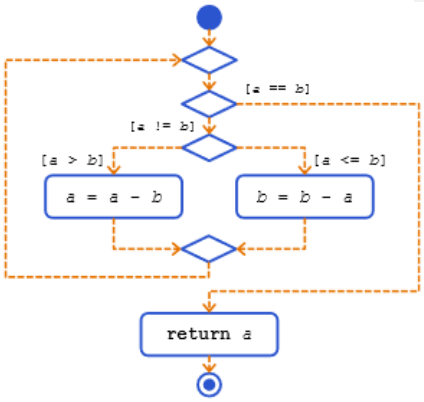

```
while (<expression>)
    <statement>
```

```
do
    <statement>
while (<expression>)
```



- Példa:

```
while (a!=b){
    if (a > b)}
        a = a - b;
    } else {
        b = b - a;
    }
}
return a;
```



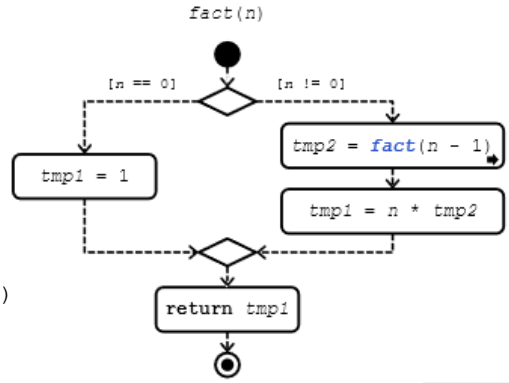
- Definíció
  - ***ciklomatikus komplexitás***
    - vezérlési folyamhoz tartozó:
      - $M = E - N + 2$
      - E: vezérlési élek száma
      - N: vezérlési csomópontok száma

[K22] megjegyzést írt: Képen élek (narancssárga).

[K23] megjegyzést írt: Képen csomópontok (kék).

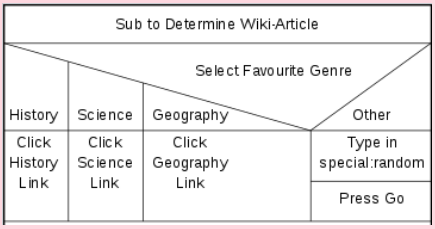
- Példa:
  - `n!` meghatározása

```
int fact (int n){
   int tmp1, tmp2;
   if (n == 0)}
      tmp1 = 1;
   } else {
      tmp2 = fact(n - 1)
      tmp1 = n * tmp2;
   }
}
return tmp1;
```



- Definíció
  - ***jólstrukturált blokk/részfolyamatok***
    - egyetlen elemi tevékenység önmagában
    - egyetlen folyamathivatkozás/hívás
    - üres vezérlési élszakasz
    - „*soros kapcsolás*”: $P_1, \dots, P_n$ jólstrukturált blokkok szekvenciája
    - „*fork-join kapcsolás*”: $P_1, \dots, P_n$ jólstrukturált blokkok egy n ágú fork és n ágú join közé zárva szekvenciája
    - „*decision-merge kapcsolás*”: $P_1, \dots, P_n$ jólstrukturált blokkok egy n ágú decision és n ágú merge közé zárva szekvenciája
    - „*ciklus*”:
      - kétágú merge csomóponttal kezdődik
      - majd jólstrukturált P_1 blokk jön
      - majd kétágú decision
        - » ennek egyik ága a részfolyamat vége
        - » másik jólstrukturált P_2 blokktól merge-be tér vissza
    - egyetlen start és egyetlen cél csomópontja van

[K24] megjegyzést írt: Pl. goto, break, return parancsok átláthatatlanná teszik a programok, túlmutatnak a jólstrukturáltságon.
Return kérdéses. Van, amikor az teszi a vezérlési strukturát egyszerűbbé.
Leírható Nassi-Shneiderman struktogrammal.



[K25] megjegyzést írt: Máshol definiált folyamatmodell újrafelhasználása.

[K26] megjegyzést írt: Egyszerű vezérlési éllel egymás után kötve őket.

BACK

5. előadás:
Modellek ellenőrzése

1. Követelmények a modellekkel szemben

- Definíció
  - ***helyesség***
    - modell vagy kód megfelelése a követelményeknek
      - megfelel a funkcionálisnak
      - nemfunkcionálisak ellenőrzése
    - szempontok
      - mindig képes teljesíteni a feladatot
      - hibamentes
      - nincs tiltott viselkedés
  - ***funkcionális követelmény***
    - követelmény, mely egy rendszerösszetevő által ellátandó funkciót definiálnak
  - **csoportosítás**
    - *megengedett viselkedés*
      - milyen állapotban lehet/nem lehet a rendszer
      - milyen viselkedés tilos
      - univerzális követelmények
        - » mindig igaznak kell lenniük
    - *elvárt viselkedés*
      - milyen állapotokba kell eljutniuk
      - milyen funkciókat kell tudnia
      - egzisztenciális követelmények
        - » lehessen lehetőség a teljesítésülésükre
  - ***nemfunkcionális követelmény/extrafunkcionális***
    - ezeken kívül eső követelmények
    - ezek a rendszer minőségére vonatkoznak, pl. megbízhatóság, teljesítmény kritériumai

[K27] megjegyzést írt: Funkcionális követelmények meghatározzák, hogy mit fog a rendszer csinálni a nemfunkcionális, meg hogy hogyan kell a rendszernek ezeket a funkciókat ellátnia.

[K28] megjegyzést írt: „Egy kereszteződés lámpái soha nem lehetnek egyszerre zöldek.”

[K29] megjegyzést írt: „A lámpa legyen képes zöldre váltani.”

- ***biztonsági követelmény***
  - definiálják mely viselkedés engedett, mely tiltott
  - univerzális követelmények, minden időpillanatban teljesülniük kell
- ***élőségi követelmény***
  - definiálják az elvárt viselkedést
  - egzisztenciális követelmények, megfelelő körülmények közt előbb-utóbb teljesíteni képes bizonyos elvárásokat
- ***holtpont (deadlock)***
  - állapot a rendszerben, amelyben a végrehajtás megáll
  - rendszer többé nem képes állapotot váltani
  - nem mutat semmilyen viselkedést
  - csak külső segítséggel képes kilépni
- ***livelock***
  - végrehajtás megáll
  - résztvevő komponensek egy végtelen ciklusban ragadnak → nem végez hasznos tevékenységet

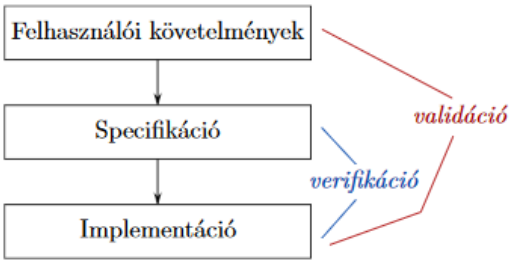
[K30] megjegyzést írt: Például: jelzőlámpán egyszerre sosem világíthat piros és zöld fény is.
Vannak azonban ezek keverékei is, különösen komplex esetekben, nem feltétlen lehetséges valamelyik kategóriába sorolni.

[K31] megjegyzést írt: Például: jelzőlámpa képes legyen zöldre váltani

[K32] megjegyzést írt: Gyakran előforduló oka, ha a rendszerben két vagy több folyamat várakozik egymásra. Nem modellezett beavatkozás nélkül nem lehet kilépni. Párhuzamos rendszereknél követelmény a holtpontmentesség.

- Példa
  - livelock
    - két ember szembe találkozik egymással, de udvariasan kitérnek egymás elől, de mindig azonos irányba
    - nem tudnak mozogni, képesek állapotot váltani, de nem haladnak előre
    - „végtelen ciklus”
  - deadlock
    - kölcsönösen nem tudnak megmozdulni a másiktól
    - „örök várakozás”

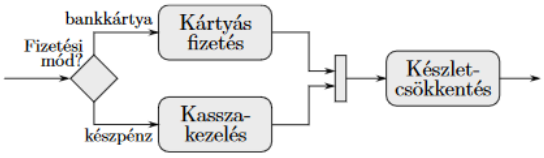
- **helyességvizsgálatok**
- Definíció
  - ***verifikáció***
    - azt vizsgáljuk, hogy az implementáció (az elkészített modell vagy rendszer) megfelel-e a specifikációnak
    - kérdés:
      - helyesen fejlesztjük-e a rendszert
      - megfelel-e az előírtaknak
  - ***validáció***
    - a rendszert a felhasználói elvárásokhoz hasonlítjuk
    - kérdés:
      - megfelelő rendszert fejlesztjük-e
- Példa
  - kereszteződést felszerelünk jelzőlámpákkal
  - minden tökéletesen működik, verifikáció helyes
  - megrendelő megkérdezi: „És hol lehet átkapcsolni villogó sárgára?”
  - ez nem volt a specifikáció része → nem tudjuk teljesíteni, de tudjuk, hogy amúgy a rendszer jól működik, csak a rendelői elvárásoknak nem felel meg



2. Statikus ellenőrzés

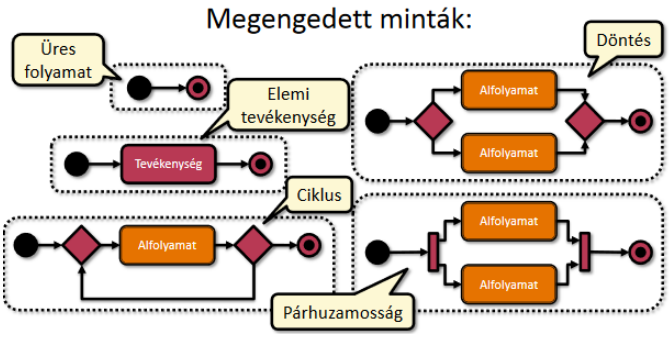
- Definíció
  - ***statikus ellenőrzés***
    - a vizsgált rendszert vagy modellt annak végrehajtása, szimulálása nélkül elemezzük

- Példa
  - „ránézésre látszik” hibákat érdemes ezzel javítani
  - szintaktikai hibák keresésénél is alkalmazzuk, amikor a rendszer tipikusan nem is futtatható
- ***szintaktikai hiba***
  - azok a hibák, amikor a modell nem felel meg a metamodelljének vagy a program nem felel meg a programozási nyelv formai megkötéseinek
  - szöveges leírások esetén könnyebben előforduló hiba
  - nagy biztonsággal kimutathatók (legkésőbb futtatás vagy végrehajtás során)
  - ritka, hogy egy statikus ellenőrző helyes kódot vagy modellt szintaktikailag hibásnak értékel
- Példa
  - állapotokhoz nem kötött állapotátmenet egy állapotgépben
  - hiányzó zárójel vagy elgépelés egy programkódban
- ***szemantikai hiba***
  - ha a rendszer szintaktikailag helyes, de logikailag nem
  - nem olyan egyértelmű hibákat „code smellnek” nevezünk, ezeket a statikus ellenőrző ki szokta szúrni
- Példa
  - szintaktikailag helyes, szemantikailag nem
  - join mindkét bemeneten tokent vár, de nem léphet tovább (holtpont)



[K33] megjegyzést írt:
False positive: ellenőrző eszköz helyes dolgot hibásnak jelez.
False negative: ellenőrző hibás dolgot helyesnek jelez.

- **védekezés szemantikai hiba ellen**
  - hibaminták megfogalmazása
    - statikus ellenőrző ezeket a hibamintákat keresi majd
  - megkötések alkalmazása
    - programozásnál kódolási szabályok
      - ne használjunk pointereket C-ben
    - jólstrukturált folyamatmodell kizárólag adott mintákból:



- ***szimbolikus végrehajtás***
  - bonyolultabb szemantikai hibák kiszűrésére
  - konkrét értékek helyesett szimbolikus értékekkel való „imitálás”
  - belső elágazások által támasztott feltételeket összegyűjtük → következtetünk az egyes változók értékeire

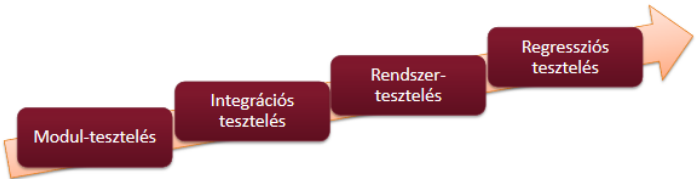
3. Tesztelés

- Definíció
  - ***tesztelés***
    - olyan tevékenység, amely a rendszert vagy egy futtatható modelljét bizonyos meghatározott körülmények közt futtatjuk/szimuláljuk
    - célja: vizsgált rendszer minőségének felmérése, javítása, hibák azonosítása
      - hibajelenségek meglétét vagy hiányát vizsgáljuk
  - **fajtái, szakaszai**
    - *modultesztelés*
      - egy komponens leválasztása és tesztelése

[K34] megjegyzést írt:
C nyelvehez leghíresebb szabálygyűjtemény. MISRA C. Bármilyen beágyazott rendszer esetén használható.

[K35] megjegyzést írt: Tehát már nem próbálgatással futtatás.

- *integrációs tesztelés*
  - több komponens együttes tesztelése
- *rendszertesztelés*
  - a teljes rendszer együttes tesztelése
- *regressziós tesztelés*
  - változtatások utáni (szelektív) újratesztelés

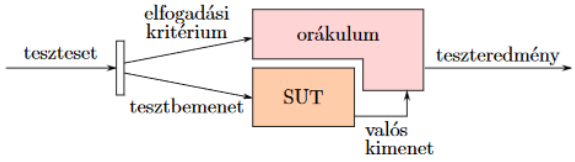


- ***debuggolás***
  - egy bizonyos hibajelenség lokalizálása, kiváltó okának megkeresése

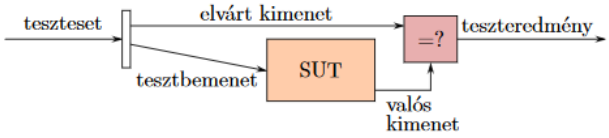
- **tesztelés komponensei**
- Definíció
  - ***tesztelendő rendszer (SUT, system under test)***
    - az a rendszer, amelyet a teszt során futtatni fogunk vizsgálat céljából
  - ***tesztbemenetek***
    - a tesztelendő rendszer számára biztosítandó bemeneti adatok
  - ***tesztorákulum***
    - olyan algoritmus és/vagy adat, amely alapján a végrehajtott tesztről eldönthető annak eredménye
  - ***teszteset***
    - adatok összessége, amelyek egy adott teszt futtatásához és annak értékeléséhez szükségesek
  - ***tesztkészlet***
    - tesztesetek adott halmaza
  - ***tesztfuttatás***
    - egy vagy több teszteset végrehajtása

[K36] megjegyzést írt:
„Bemenetei értékek, végrehajtási előfeltételek, elvárt eredmények (elfogadási kritérium) és végrehajtási utófeltételek halmaza, amelyeket egy konkrét célért vagy a tesztért fejlesztettek.”

- tesztfuttatás után orákulum segítségével megtudjuk a teszt eredményét (pass, fail, error)
- error: nem lehet eldönteni, hogy a teszt sikeres vagy nem
- teszteredmény megkapható
  - kapott, tesztesetben megfogalmazott elvárt kimenetek összehasonlításával
  - referenciaimplementációval összehasonlításból
  - ellenőrizhetjük implicit elvárásokat
    - pl. a kód nem dob-e kivételt



  - legegyszerűbb eset: teszteset közvetlenül tartalmazza az adott tesztbemenetekre elvárt kimeneteket



- **tesztelés metrikái**
  - tesztelés célja a minőség javítása, hibák megtalálása, javítása
  - egy idő után elfogynak a hibák, a rendszer „elég jónak” mondható → utána nem gazdaságos folytatni a tesztelést
  - honnan tudjuk mikor jutottunk el ide?
    - tesztkészlet fedésének mérése
    - tesztkészlet meglátogat minden állapotot, meghív minden metódust akkor sem vizsgáltunk meg mindent
    - ha mindent meghív, nem biztos, hogy minden utasítást érint (pl. elágazások)
  - ***állapotfedettség***

$$\frac{\text{érintett állapotok}}{\text{összes állapot}}$$

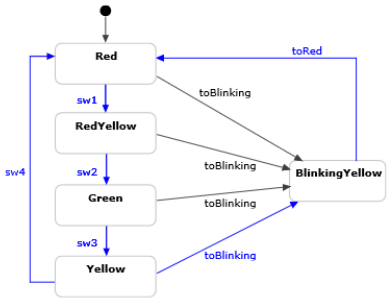
- ***átmenetfedettség***

$$\frac{érintett\ átmenetek}{összes\ átmenet}$$

- ***utasításfedettség***

$$\frac{érintett\ utasítások}{összes\ utasítás}$$

- Példa
  - közlekedési lámpa állapotgépe

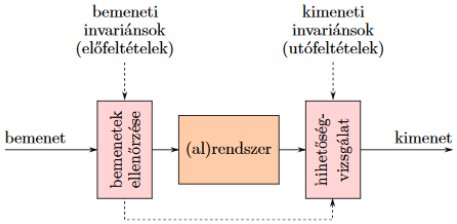


  - tesztkészlet

Teszteset	Bemenet (n)
1	`sw1, sw2, sw3, sw4`
2	`sw1, sw2, sw3, toBlinking, toRed`

  - ez a két teszt az összes állapotot be fogja járni, az állapotlefedettség 100%
  - átmenetfedettség nem teljes, 9-ből 6 tranzíciót fed le → átmenetefedettség 66,6%

- **Tesztelés futásidőben**
  - **„öntesztelés” vagy monitorozás**
    - magas minőségi elvárásoknál
      - pl. biztonságkritikus alkalmazási területek
    - külső komponensek használata, melyeknek minőségéről nem tudunk
    - ***invariáns***
      - követelmények, amelyek teljesülését folyamatosan, minden állapotban elvárjuk



- **monitorozás 2 fő lépése**
  - *bemenetek ellenőrzése*
    - bemeneti adatok megfelelőssége
    - definiált bemenetei invariánsok alapján
  - *hihetőségvizsgálat*
    - kimeneti adatok megfelelőssége
    - definiált kimenetei invariánsok alapján
- Példa
  - másodfokú egyenlet gyökeit kiszámoló

```
void Roots(float a, b, c, float &x1, &x2) {
    float d = sqrt(b*b-4*a*c);

    x1 = (-b+d)/(2*a);
    x2 = (-b-d)/(2*a);
}
```

  - nem helyes, hiányoznak elő – és utófeltételek (diszkrimináns nem lehet negatív stb.)

```
void RootsMonitor(float a, b, c, float &x1, &x2) {
    // előfeltétel
    float D = b*b-4*a*c;
    if (D < 0)
        throw "Invalid input!";

    // végrehajtás
    Roots(a, b, c, x1, x2);

    // utófeltétel
    assert(a*x1*x1+b*x1+c == 0 && a*x2*x2+b*x2+c == 0);
}
```

  - ***logelemzés***
    - monitor futtatása naplózott bemeneteken/kimeneten

[K37] megjegyzést írt: Előfeltétel.

[K38] megjegyzést írt: Utófeltétel.

[K39] megjegyzést írt:
Design by contract.
Ennek célja a rendszer minőségének javítása, hibák elkerülése.

4. Formális verifikáció

- Definíció
  - olyan módszerek, amelyek segítségével adott modellek vagy programok helyességét matematikailag precíz eszközökkel vizsgálhatjuk
  - fontosabb módszerei
    - **modellellenőrzés**
      - lehetséges viselkedések kimerítő vizsgálata

Tesztelés	Modellellenőrzés
Szúrópróbaszerű	Kimerítő/teljes
Elvárt kimeneteket ellenőriz	Állapotok sorozatát ellenőrzi
Kisebb számítási igényű	Nagy számítási igényű
Nem bizonyítóerejű	Formálisan bizonyít

    - **automatikus helyességbiztosítás**
      - axiómarendszerek alapján tételbizonyítás
    - **konformanciavizsgálat**
      - adott modellek közt bizonyos konformanciarelációk teljesülését vizsgáljuk
      - beláthatjuk, hogy különböző modellek viselkedése azonos vagy nem
      - megfelelősség ellenőrzése

BACK

6. előadás:
Vizuális adatelemzés

1. Felderítés, megerősítés, szemléltetés

- **vizualizáció eszköze**

felderítő adatelemzés (*EDA*)

  - adatok megértése, hipotézisek megsejtése
  - hipotézisek és modellek megerősítése
  - eredmények szemléletesebb prezentálása → megerősítő adatelemzés (*CDA*)

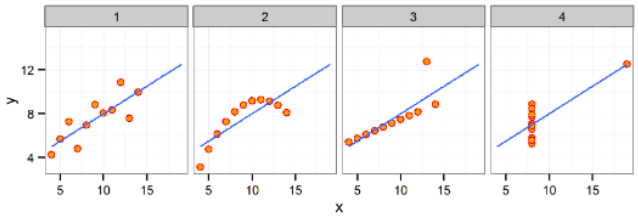
- ***EDA***

[K40] megjegyzést írt: John W. Turkey amerikiai matematikus és statisztikus munkássága.

  - az adatelemzési folyamat nagy hangsúlyt fektet az adatok „megértésére”
    - az adatok által leírt rendszer
    - adatokon belüli és az adatok és általuk leírt rendszer közötti összefüggések
  - adatok grafikus reprezentációja, mint ennek eszköze, kiemelt szerep
  - modelljelölt-építés, hipotézisek felállítása iteratív módon
    - modellspecifikáció – analízis – modell-újraspecifikáció lépéssorozat ismétlésével
  - CDA-val összevetve
    - CDA: pontos statisztikai mértékeket számol
    - EDA: sejtéseket fogalmaz meg, amelyeket a statisztikai/elemzés matematikai eszközeivel igazolhatunk (vagy cáfolhatunk, módosíthatunk)

- Példa
  - Dr. John Snow 1854-ben londoni kolerajárvány alatt bepontozta a térképen a halálesetek számát
  - kirajzolódott, hogy a Broad Streeten levő kút lehet az oka

- **Anscombe négyese (*Anscombe's quartet)***
  - *példa arra, hogy a vizuális elemzés könnyen ad olyan plusz információt, melyet az alap statisztikai módszerek nem feltétlenül vesznek észre*
  - hibás feltételezések elkerülése és intuíció



- **folyamat alapja interakció**
  - adatvizualizáció
    - több ábra együttes vizsgálata
  - vizuális kiértékelés
    - emberi kognitív képességek használata
  - vizuális kiválasztás, manipuláció
  - interpretáció, korreláció más modellekkel, kiértékelés

2. **Numerikus és kategorikus változók**

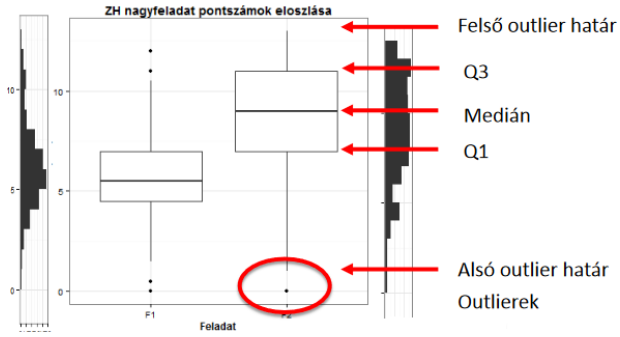
- ***numerikus (numerical)***
  - az alapvető aritmetikai műveletek értelmesek
  - **folytonos**
    - mért, tetszőleges értéket felvehet
      - adott tartományon belül
      - adott pontosság mellett
    - például: teremben ülők ZH pontszámának átlaga
  - **diszkrét**
    - számolt, véges sok értéket vehet fel adott tartományban
    - például: előadáson ülők száma

- ***kategorikus (categorical)***
  - matematikai műveletek nem értelmezhetőek rajtuk, max sorbarendezés
    - **ordinális**
      - teljes rendezés az értékeken
      - például: szállodai csillagok
    - **nominális**
      - például: férfi, nő
- ***terjedelem jellemzése: percentilis***
  - n-edik percentilisnél az értékek $n\%$-a kisebb
- Példa:
  - $\{3, 4, 4, 5, 5, 6, 10, 20\}$
    - $50.$ percentilis: 5
    - $25.$ percentilis: 4
    - $75.$ percentilis: 6

3. Egydimenziós diagramok

- reprezentálása pontdiagramon (*dot plot*)
- ***doboz-ábra (box plot)***
  - 5 jellemzőt szemléltet
    - minimum
    - maximum
    - medián
    - első kvartilis (Q_1)
    - harmadik kvartilis (Q_3)
  - kvartilisek távolságot egy doboz mutatja
  - **„bajusz" (*whisker*):**
    - első kvartilis alatti és harmadik feletti vonal
    - legfeljebb a kvartilisek közötti távolság 1,5x nyúlik
  - ezen kívül eső megfigyelések pontként ábrázolva
  - egy kategorikus változó mentén részhalmazokra bontva használjuk

[K41] megjegyzést írt: Inter-Quartile Range, $IQr = Q_3 - Q_1$



- több eloszlás esetén jellemzők gyors, vizuális összehasonlítására alkalmas
- **hátránya**
  - eloszlást erősen absztrahálja
  - → multimodalitás nem olvasható le róla

- *változó eloszlás becslése*
  - !!$x \in \mathbb{R}$, ahol $a, b \in \mathbb{R}$ intervallumon n megfigyeléssel rendelkezünk
  - képezzünk $[a, b)$ egy L nemátfedő intervallumokból (bin-ekből, cellákból) álló partícionálását:

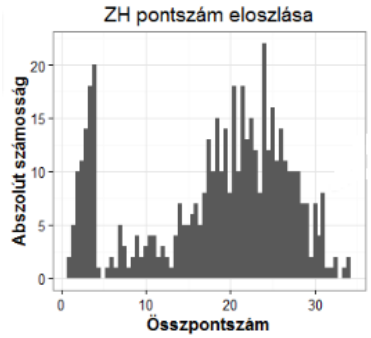
$$T_l = \left[t_{n,l}, t_{n,l+1}\right), l = 0, 1, \dots, L-1, \text{ ahol}$$
$$a = t_{n,0} < t_{n,1} < \dots < t_{n,L} = b$$

  - ! I_{T_l}, ahol l-ik cella indikátor-függvénye és ! $N_l = \sum_{i=1}^{n} I_{T_l}(x_i)$
  - T_l-be eső minták száma $(l = 0, 1, \dots L-1, \sum_{l=0}^{L-1} N_l = n)$
  - hisztogram, mint változó eloszlás becslője:

$$\hat{p}(x) = \sum_{l=0}^{L-1} \frac{\frac{N_l}{n}}{t_{n,l+1} - t_{n,l}} \cdot I_{T_l} \cdot (x)$$

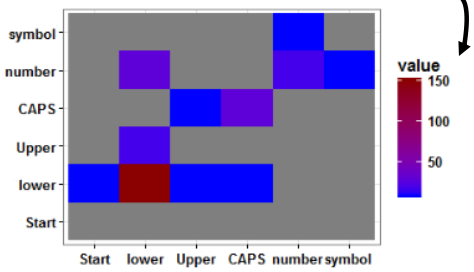
  - EDA szempontjából hátrány, hogy alakja érzékeny a(z)
    - első cella kezdőpontja,
    - cellaszélesség megváltasztására

[K42] megjegyzést írt: Legyen



4. Többdimenziós diagramok

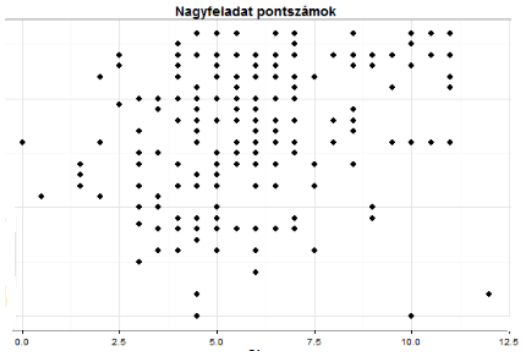
- ***kétdimenziós diagramok***
  - **folytonos-folytonos eset**
    - pl. szórásdiagramok, hőtérképek



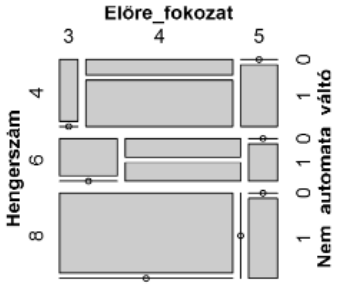
  - **kategorikus-folytonos eset**
    - kategória-kombinációk relatív számosságának felismerése → mozaik/fluktuációs diagram, de ezek már n-diagramtípusok, 2 változós eset speciális
    - pl. kategóriánkként alkalmasan színezett hisztogram, doboz-diagram
- ***n-dimenziós diagramok***
  - **tisztán kategorikus eset**
    - pl. mozaik, fluktuációs diagram
  - **tisztán folytonos eset**
    - pl. párhuzamos koordináta diagram
      - megjeleníthető kategorikus változó rajta

[K43] megjegyzést írt: *Heat map* – 2D hisztogramok

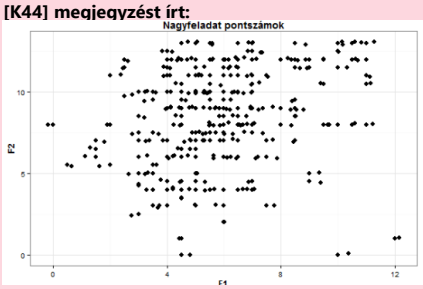
- **SPLOM – *scatterplot matrix* – szórásdiagram-mátrix**
  - sokváltozós adatkészlet változó-vektorát megfeleltetjük egy mátrix sorainak, oszlopaink
  - cellákban változópárok szórásdiagramja



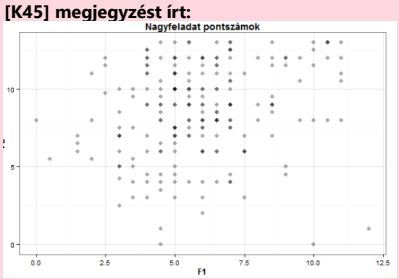
  - overplotting kezelése (*túl sok pont egy helyen*)
    - jitter
    - átlátszóság
    - méret növelése
- **mozaik és fluktuációs diagram**
  - *mozaik*:
    - kategorikus változók érték-kombinációi előfordulási gyakoriságainak területarányos vizualizációja
    - ábrát alkotó csempék/lapok (*tiles*) négyzet rekurzív vízszintes és függőleges darabolásával kapjuk
    - olvashatóság miatt 8 változónál többet ne használjunk (már 4-5-nél is gond)



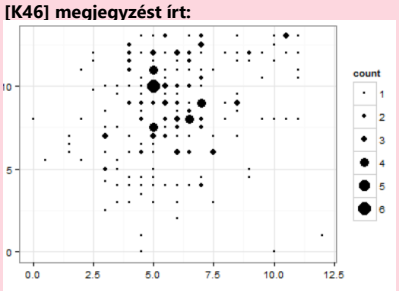
[K44] megjegyzést írt:



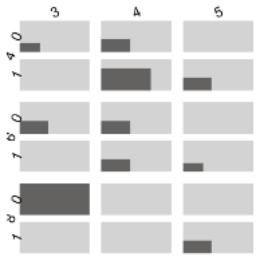
[K45] megjegyzést írt:



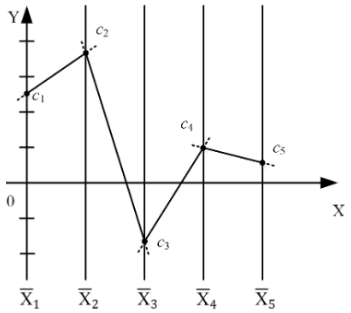
[K46] megjegyzést írt:



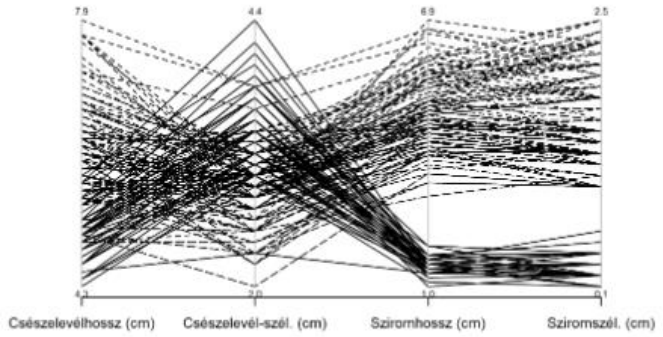
- *fluktuációs*:
  - mozaik nagyobb dimenziószám esetén → fluktuációs diagram szükséges (olvashatóbb)
  - érték-kombinációkhoz azonos méretű lapokat rendelünk → kombinációk könnyen beazonosíthatók
  - legnagyobb elemszámú lapot teljesen kitöltjük, többit az előfordulások relatív gyakorisága alapján
  - hátrány: kitöltött idomból nem tudjuk leolvasni a változókban értelmezett gyakoriságot



- ***párhuzamos koordináta diagram***
  - Descartes-féle koordinátákkal ellátott $\mathbb{R}^2$ euklideszi síkban egymástól egységnyi távolságra elhelyezi az $\mathbb{R}$ valós egyenes N másolatát
  - ezeket tengelyként használva ábrázolja $\mathbb{R}^N$ N-dimenziós euklideszi tér pontjait
  - $(c_1, \dots, c_n)$ koordinátákkal rendelkező $C \in \mathbb{R}^N$ pont képe teljes poligon-vonal
  - $N\ db$, szegmenseket meghatározó pontja rendre párhuzamos tengelyekre eső $(i-1, e_i)$ pontok $(i = 1, \dots, N)$



- magas dimenziószám esetén áttekintésre alkalmas
- hátrány: növekvő pontszám esetén egyre kevésbé átlátható
    - pl. Fisher-féle írisz-adatkészlet párhuzamos koordinátákra



    - DE, ha teszünk bele:
        - faktorváltozók szín szerinti megkülönböztetés
        - részhalmaz kiválasztás
        - tengelyen intervallum kiválasztás
    - → nagyon hatékony EDA eszközzé válik

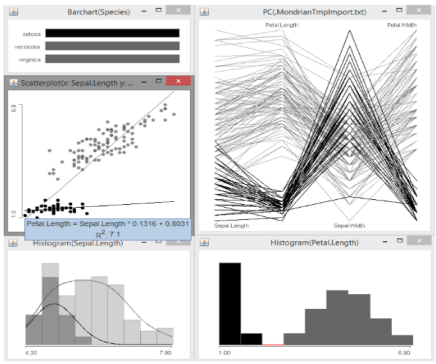
- **eszköztámogatás**
    - táblázat megadja a diagramtípusokat megvalósító R függvényeket
    - grafika területén különösen igaz, hogy ugyanannak a funkciónak több megvalósítása elérhető

Diagramtípus	függvény	csomag
oszlop, szintvonal, szórás, doboz, hisztogram, mozaik	`barplot, contour, plot, boxplot, hist, mosaicplot`	`graphics`
idősor, hőtérkép	`plot.ts, heatmap`	`stats`
fluktuáció	`fluctile`	`extracat`
párhuzamos koordináták	`parcoord`	`MASS`
szórás-mátrix	`splom`	`lattice`

5. Interaktív statisztikai grafika

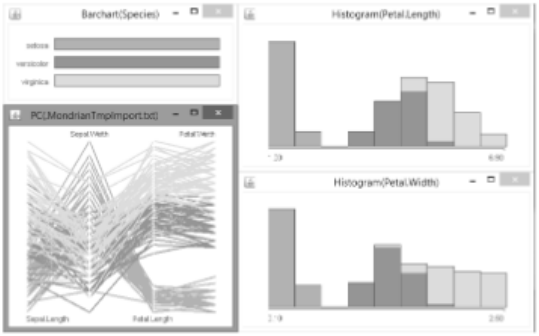
- **eszköz által támogatott interakciók:**
    - lekérdezések (*queries*) és helyi interakciók
    - kiválasztás és csatolt kijelölés (*selection and linked highlighting*)
    - csatolt elemzések/analízis (*linked analyses*)

- ***lekérdezések***
  - Definíció
    - interaktív diagramon megjelenített elemekkel kapcsolatos statisztikai információk megjelenítése (egérrel)
  - **megjeleníthető adatok függenek:**
    - ábra által alkalmazott statisztikai transzformáció
    - aktív kijelöléstől
  - **szórásdiagramon**
    - pont koordinátáit
    - egy kijelölés koordináta-intervallumait kérdezhetjük le
  - **oszlopdiagramon**
    - kategória elemszámát
    - aktív kijelölésbe tartozó elemek számát
- ***helyi interakciók***
  - diagramokkal mellékhatásmentesen interakcióba léphetünk
  - egyes operációk általánosan megjelennek, többi ábra-specifikus
- ***kiválasztás és csatolt kijelölés***
  - ez adja a minőségi különbséget a statikus és az interaktív statisztikai vizualizáció között
  - adatkészleten több diagram olyan, mint egy reláció különböző projekcióii (diagram változóira), ezek statisztikai transzformációi
  - diagramon elemeket egérrel kiválasztva az inverz transzformáció egy „sorhalmazt” határoz meg az eredeti relációban, melynek képe többi diagramon „kiemelve” vizualizálható



[K47] megjegyzést írt: Objektumok sorrendjének módosítása, skálamódosítás, nagyítás…

- *kategória-vezérelt színezés (colour brush)*
  - kategóriákat megjelenítő diagramoknál
    - pl. oszlopdiagram, mozaik-diagram, hisztogram
  - színezés a kezdeti diagram minden eleméhez egy színt rendel → többit ezzel színezi



- ***csatolt analízis***
  - a kiválasztások változásával reaktívan analízisek, illetve statisztikai modellek felállításai újrafuthatnak
  - megvalósítás: Mondrian szórásdiagramra illeszthető regressziós egyenes
    - teljes adatkészlet azonosítása, feltüntetése
    - paraméterek lekérdezéssel megtekinthetőek

[K48] megjegyzést írt: Egyes intervallumok oszlopai kategóriaként értelmezhetőek.

[K49] megjegyzést írt: Pl. oszlopdiagram oszlopai különböző színekkel színezi.

BACK

7-8. előadás:
Teljesítménymodellezés

1. **Teljesítménymodellezés**
   - rendszer vizsgálata, mely felhasználókat kiszolgál véges erőforrásokat használva
     - vizsgálat: feldolgozási idő (*válaszidő*) fókuszál
   - *átbocsátás*: egységnyi idő alatt feldolgozott tranzakciók száma
     - erőforrások kihasználtsága a *rendszer egyensúlyi állapotá*ban
   - *tranzakció*
     - teljes rendszer felé érkező felhasználói kérések
     - jelölés: tr
   - *kérések*
     - alrendszernek továbbított feladatrészek
     - jelölés: k
   - *átlapolódás* van, ha
     - van várakozási sor a rendszerben
     - van több feldolgozóegység
     - ha nincs átlapolódás → minden pillanatban legfeljebb egy tranzakció van a rendszerben

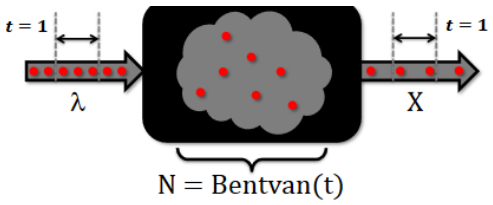
[K50] megjegyzést írt: Ugyanaz, mint a tranzakció, csak más rendszerekere nézve.

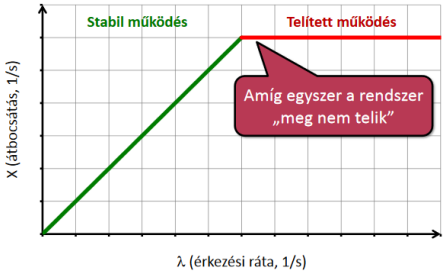
2. **Rendszerszintű tulajdonságok**
   - Definíció
     - ***érkezési ráta***
       - egységnyi idő alatt érkező kérések
       - jelölés: λ
       - mértékegysége: db/s

$$\lambda = \frac{be(t)}{t} \qquad [\lambda] = \frac{1}{s}$$

- ***átbocsátás***
  - egységnyi idő alatt feldolgozott kérések
  - jelölés: X, mint „throughput”
  - mértékegysége: db/s

$$X = \frac{ki(t)}{t} \qquad [X] = \frac{1}{s}$$



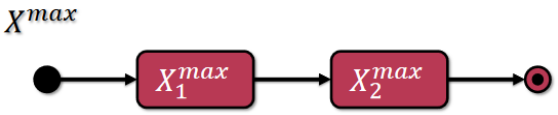


- ***válaszidő***
  - felhasználói kérések által a rendszer határain belül töltött átlagos idő
  - jelölés: R, mint „round-trip time”
  - mértékegysége: s
- ***rendszerben lévő kérések átlagos száma***
  - átlapolódás mértéke
  - jelölés: N
  - mértékegysége: db

- ***rendszerben egyensúlyi állapot***
  - ha $\lambda = X$
  - vagyis: egységnyi idő alatt ugyanannyi új felhasználói kérés érkezik a rendszerben, mint amennyit ezalatt feldolgozott
  - átlagos értékek **CSAK EKKOR** használhatóak

3. Folyamatmodellek elemzése

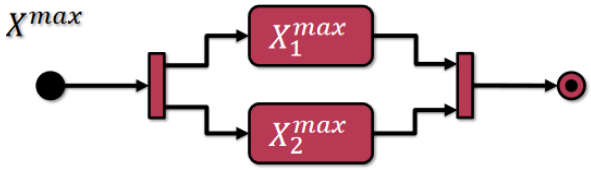
- Definíció
  - ***szekvenciális komponálás***



  - $X^{max} = \min(X_1^{max}, X_2^{max})$
    - egyik tevékenység hiába gyors, másik feltorlódhat
    - pl. Okmányiroda
  - *szűk keresztmetszet*
    - minimumhelyet adó tevékenység

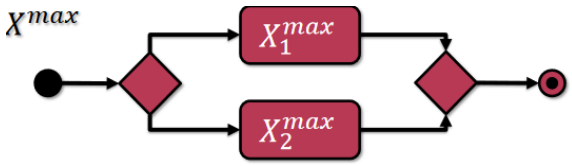
[K51] megjegyzést írt: Vagy ahhoz rendelt erőforrás.

  - ***párhuzamos komponálás***



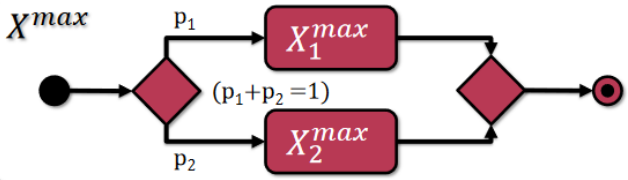
  - $X^{max} = \min(X_1^{max}, X_2^{max})$
    - egyik tevékenység hiába gyors, be kell várniuk egymást
    - pl. ZH javítás
  - szűk keresztmetszet

- ***komponálás szabad választással***



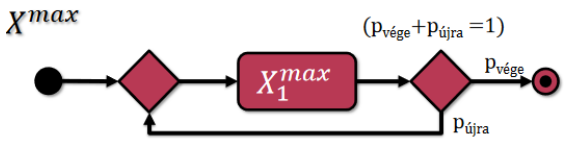
  - $X^{max} = X_1^{max} + X_2^{max}$
    - tokenek mindkét irányba mehetnek
    - ha egyik telítésben van, másik fogadhat tokent
    - pl. pénztárak

- ***komponálás kötött arányú választással***



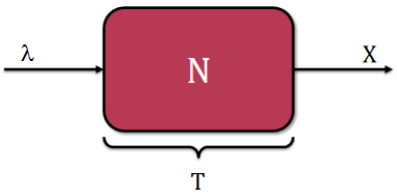
  - $X^{max} = \min(\frac{1}{p_1} \cdot X_1^{max}, \frac{1}{p_2} \cdot X_2^{max})$
    - tokenek p_1 és p_2 valószínűséggel választják az $1.$, $2.$ tevékenységet
    - teljes folyamatból $\frac{1}{p_1}$, $\frac{1}{p_2}$ jut az $1.$, $2.$ tevékenységre
    - pl. felhasználó viselkedése egy weblapon vásárlásnál
  - szűk keresztmetszet

- ***komponálás ciklussal***



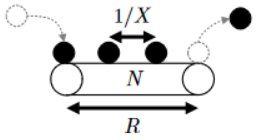
  - $X^{max} = \frac{1}{\frac{1}{p_{vége}}} \cdot X_1^{max} = p_{vége} \cdot X_1^{max}$
  - pl. felhasználó a rendszerben, 10% eséllyel kilép, 90% új kérés

- ***vizitációs szám***
  - folyamat a végrehajtása során átlagosan hányszor fut le az adott tevékenység/alfolyamat
    - választás esetén maga a döntési valószínűség
    - ciklus esetén a várható iterációk száma
  - *átbocsátóképesség a vizitációs szám ismeretében*
    - $X^{max} = \frac{1}{v} \cdot X_1^{max}$
  - *végrehajtási idő*
    - $T_{folyamat} = v \cdot T_{task}$
- ***Little-törvény***
  - $N = X \cdot T$
  - λ: érkezési ráta $\left[\frac{darab}{s}\right]$
  - X: átbocsátás $\left[\frac{darab}{s}\right]$
  - T: rendszerben töltött idő $[s]$
  - N: rendszerben lévő tokenek száma $[darab]$
  - tehát: rendszerben tartózkodó kérések átlagos száma egyenlő az átbocsátás és az átlagos rendszerben töltött idő szorzatával



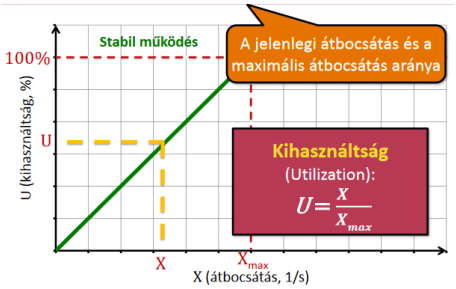
[K52] megjegyzést írt: $\lambda = X$ esetén

- például: futószalag
  - R ideig tart végighaladni rajta
  - $\frac{1}{X} = \frac{1}{\lambda}$ időnként ráteszünk egy új elemet
  - az első elem levételének pillanatában $\frac{R}{\frac{1}{X}} = R \cdot X$

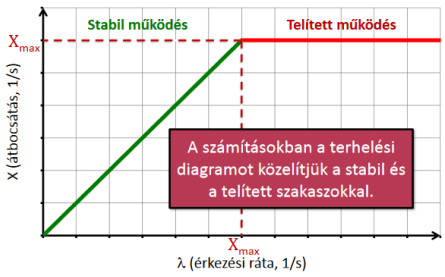


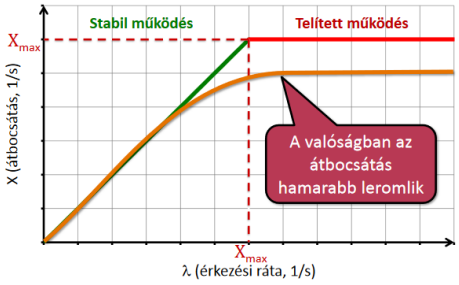
- Definíció
  - ***látogatások átlagos száma***
    - megadja, hogy egy tranzakció átlagosan hány kérést generál az i alrendszer (erőforrás) felé
    - alrendszerek, erőforrások teljesítményjellemzőit ezzel tudjuk átváltani rendszer jellemzőire és vissza
    - mértékegysége: $\frac{k}{tr}$ (kérés/tranzakció)
    - kiszámítás: $D_i = V_i \cdot S_i$
  - ***szolgáltatásigény***
    - egy tranzakció átlagosan mennyi ideig használja az adott alrendszert/erőforrást
    - rendszerszintű
    - jelölés: D_i, mint „service-demand”
    - mértékegysége: $\frac{s}{tr}$
  - ***erőforrásigény***
    - egy kérés átlagosan mennyi ideig használja az adott alrendszert/erőforrást
    - alrendszerszintű
    - jelölés: S_i, mint „resource demand”
    - mértékegysége: $\frac{s}{k}$

- ***kihasználtság***
  - megmutatja, hogy a globális teljesítménykorlátoktól milyen távol működik a rendszer
  - jelölés: U, mint „utilization”

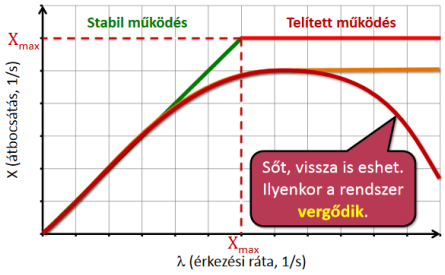


- ***közelítő vs. valós terhelési függvény***





- ***vergődés***
  - rendszer átbocsátása visszaesik a telített működés során



- ***kizárólagos erőforrás törvény***
  - ha kérések közül egyszerre legfeljebb egy futhat
    - pl. egyetlen szerver futtatja őket, azonos változót írnak
    - többi kérés sorban áll
    - T átlagos végrehajtási idő mellett:

$$X^{(1)}{}_{max} = \frac{1}{T}$$

[K53] megjegyzést írt: „Hány végrehajtás fér bele egy egységnyi időbe?"

- ***kizárólagos erőforrás kihasználtsága***
  - átbocsátás és az átbocsátóképesség aránya

$$X^{(1)}{}_{max} = \frac{1}{T} \quad \rightarrow \quad X^{(1)}{}_{max} \cdot T = 1 = U$$
$$\rightarrow U = X \cdot T$$

- ***Forced Flow törvény***
  - erőforrás, mint alrendszer átbocsátás kiszámítása a teljes rendszer átbocsátásából
  - $X_i = V_i \cdot X_0$ ebből a **Little-törvény**: $N_i = S_i \cdot X_i$
- ***kihasználtság törvény***

$$\boldsymbol{U} = \frac{X}{K} \cdot T = \frac{X \cdot T}{K} = \frac{\boldsymbol{N}}{\boldsymbol{K}}$$

$$U_i = \frac{N_i}{n_i} = \frac{S_i \cdot X_i}{n_i}$$

n_i: erőforrás legfeljebb ennyi kérést képes kiszolgálni

- maximum K darab kérés végrehajtása alatt
- ha $n = 1$, akkor N értéke közvetlenül megadja a kihasználtság értékét

$$U_i = S_i \cdot X_i$$

- Példa
  - erőforrás: disc
  - $40 \frac{kérés}{s}$
  - 1 kérés 0,0225 mp-ig tart

- mekkora a kihasználtság?

$$U = X \cdot T_{disc} = 40\frac{kérés}{s} \cdot 0{,}0225\ s = 0{,}9 = 90\%$$

- sorban állás is van a disc előtt
- disc: $40\frac{kérés}{s}$
- kérések átlagos száma a rendszerben: 4
- $T_{rendszer}$: átlagos rendszerben tartózkodási idő
- $T_{várakozás}$: átlagos sorban állási idő

$$N = X \cdot T \rightarrow T_{rendszer} = \frac{4\ kérés}{40\frac{kérés}{s}} = 0{,}1s$$

- átlagos sorban állási idő?

$$(T_{rendszer} - T_{disc}) = (0{,}1s - 0{,}0225s) = 0{,}0775s$$

- kérések átlagos száma a sorban?

$$(N_{rendszer} - N_{disc}) = 4\ darab - 0{,}9\ darab = 3{,}1\ darab$$

- Definíció
  - ***Zipf törvénye***

$$R_i \sim \frac{1}{i^{\alpha}} \qquad f \sim \frac{1}{p}$$

- R_i: az i szó előfordulási gyakorisága
- α: a korpuszra jellemző 1 érték
- egyszerűsítve $\alpha = 1$
  - f: (*frequency*) gyakoriság
  - p: (*popularity*) a szöveg „rangja” (csökkenő sorrendben)

  - ***Zipf törvénye pl. Web dokumentumokra***

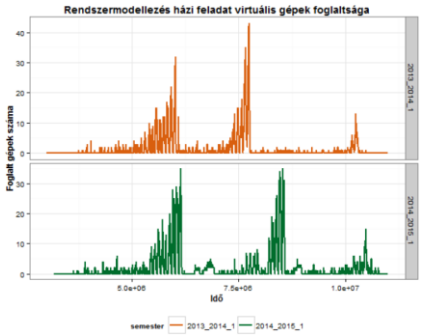
$$P = \frac{k}{r}$$

- P: hivatkozások (elérések)
- r: rang (1 = leggyakoribb)
- k: pozitív konstans

- Példa
  - weboldalak aloldalainak látogatottsága
  - slágerlisták
  - városok populációja rangsoruk szerint

▪ ***terhelés változása***

- átlagos értékekkel számoltunk
- a rendszer viselkedését a terhelés (*intenzitás*) függvényében néztük
- DE: valójában nem (feltétlenül) előre kiszámíthatóan változik a terhelés
- valójában:
  - a rendszer viselkedése *időben* változik
  - ennek műszaki hatásai vannak
    - » váltás feladatok közt, erőforrásfoglalás stb. (pl. OS)



▪ ***átbocsátóképesség***

- erőforráskészlet **felső határ**t szab az elvégezhető munka mennyiségének, az átbocsátásnak

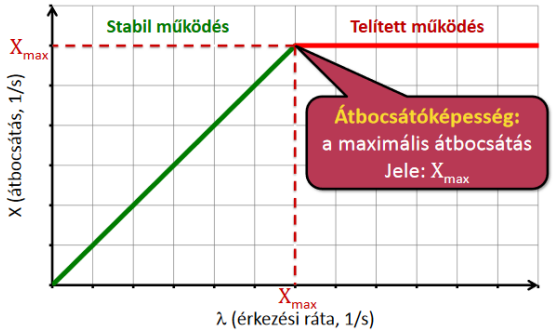
$$X_{max} = \frac{K}{T}$$

$$X_i^{max} = n_i \cdot \frac{U_i^{max}}{S_i} = n_i \cdot \frac{1}{S_i}$$

- $n_i = 1$ esetén

$$X_i^{max} = \frac{U_i^{max}}{S_i} = \frac{1}{S_i}$$

[K54] megjegyzést írt: Erőforráspéldány esetén.



- ***szűk keresztmetszet***
  - egyes erőforrásokat a teljes rendszer különböző mértékben használja
  - **egyikük** korlátozza ténylegesen a rendszer tényleges átbocsátóképességét

$$X_0{}^{max} = \min_i \left(\frac{X_i{}^{max}}{V_i} \right)$$

  - átbocsátóképességgel felírva a Little-törvényt, megkapjuk N_{max}, átlapolódás maximális mértékét a rendszerben amikor $N_{max} = 1\ tr$:

$$X_0{}^{max} = \frac{1\ tr}{R}$$

- ***szolgáltatásigény törvénye***
  - adott erőforrásra vonatkozó szolgáltatásigény meghatározása
  - egyetlen erőforráspéldány esetén $n_i = 1$

$$D_i = \frac{U_i}{X_0}$$

4. <u>Átlagos mértékek számítása mérési eredményekből</u>

- Definíció
  - ***mérési idő***
    - jelölés: T, mint „time”
    - mértékegysége: s

- ***tranzakciók száma***
  - mérési idő alatt elvégzett tranzakciók száma
  - jelölés: C_0, mint „count”
  - mértékegysége: tr vagy alrendszereknél k
- ***tranzakciók száma***
  - egyes erőforrások foglalási ideje a mért időtartamon belül
  - jelölés: B_i, mint „busy time”
  - mértékegysége: s

- egy egypéldányos, átlapolódásmentes erőforrás $(n_i = 1)$ átlagos kihasználtsága a mérés ideje alatt

$$U_i = \frac{B_i}{T}$$

- mérésidő alatt átlagos átbocsátás

$$X_0 = \frac{C_0}{T}$$

- tranzakciók átlagos szolgáltatásigénye

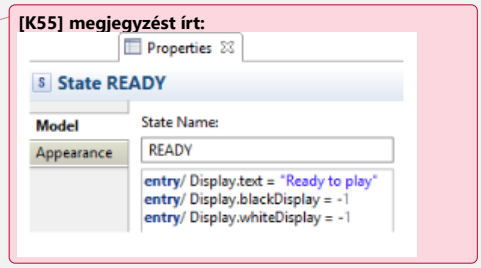
$$D_i = \frac{B_i}{C_0}$$

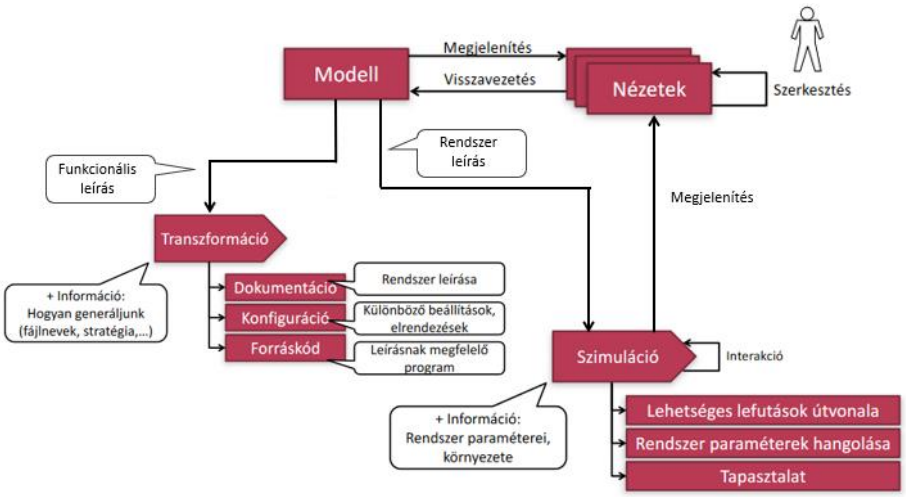
BACK

9. előadás:
Modellező eszközök, kódgenerálás

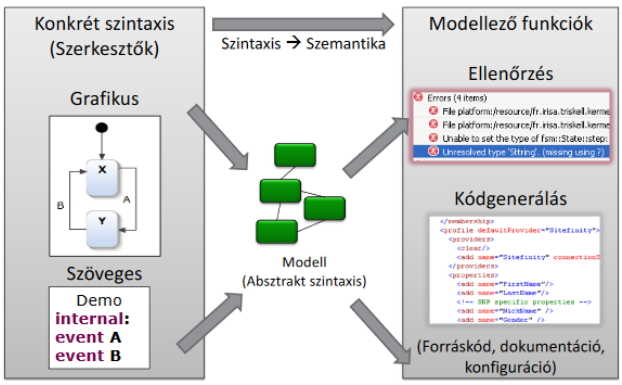
1. **Modellező eszközök funkciói**
   - szöveges
   - grafikus
   - egyéb szerkesztőfelület

[K55] megjegyzést írt:



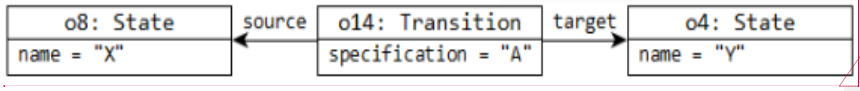


2. **Modellező eszközök**
   - Yakindu modellezési funkciói



- Definíció
  - ***absztrakt szintaxis***
    - szerkesztés alatt álló rendszermodell strukturális modellje
    - modellező program kezeli
- Példa
  - Yakindu modell

  - absztrakt szintaxis

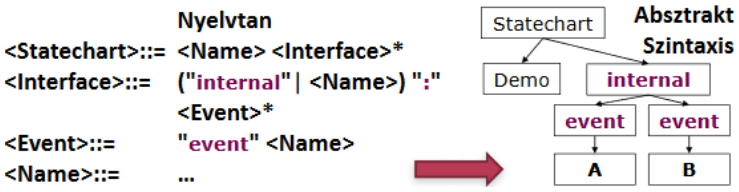


[K56] megjegyzést írt:
Neveket *stringként,* modellelemeket objektumként tároljuk. Kapcsolatokat „pointerekkel" jelöljük.

  - ***konkrét szintaxis***
    - cél: konkrét megjelenítés
    - *szöveges szintaxis* (programozási nyelv)

[K57] megjegyzést írt:
Megfelelő technológiákkal (pl. Xtext) könnyű saját programozási nyelvet csinálni.

      - feladata: szöveg ↔ modell, nyelvtani szabályok alapján



    - *grafikus szintaxis* (diagram)

[K58] megjegyzést írt:
Megfelelő technológiákkal (pl. Sirius) könnyű saját modellező nyelvet csinálni.

      - feladata: diagram ↔ modell, nézeti modell szabályok alapján
      - könnyebben átlátható, nehezebben írható
        - » feltétel a modellben teljesül → diagram létrejön
        - » diagram változik → modell változik
  - ***modellek validálása: szintaxisellenőrzés***
    - ***szintaktikai ellenőrzés***
      - modellező eszközök összekötik a logikailag egymásra épülő modellelemeket

- *szintaxisvezérelt szerkesztő*
  - szerkesztés közben hiba (`Couldn't resolve reference`)
  - fejlett szerkesztőeszközök (pl. „tab" -os lehetőségek felkínálása)
  - kód és diagram együtt
  - programozás: szerkesztés közben **hibás**
  - modellezés: szerkesztés közben **helyes**
- ***strukturált szerkesztő***
  - modell gráf vizsgálata
  - hibaminták keresése szerkesztés közben
- Példa
  - elérhetetlen állapot *(Node is not reachable.)*
  - hiányzó kezdőállapot
  - holtpont
  - változó értékadások stb.

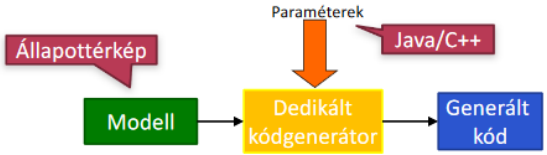
3. **Kódgenerálás**

- **motiváció**: fejlesztési idő rövidítése
- Példa
  - sakkjátékosok kezdeti gondolkodási ideje
    - kiinduló érték
    - idő változtatása
      - nem lehetséges
      - mindkét játékosnak egyszerre
      - játékosokra külön-külön
- **előnyei**
  - helyes kódot generál → tanúsítvány
  - gombnyomásra generálódik a kód minden változtatás után azonnal
  - optimalizálható átláthatóságra, hatékonyságra
- nehéz jól megírni (fentiek miatt)

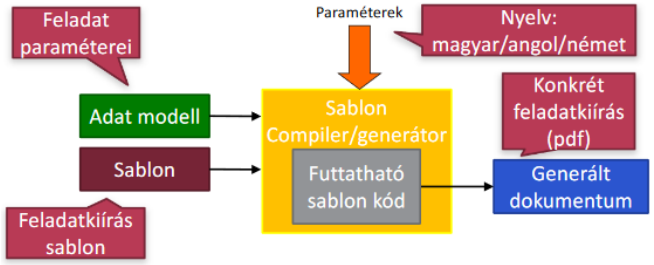
- **feladata**
  - modellnek megfelelő viselkedésű program automatikus előállítása
  - több megoldás létezik (tervezői döntések)
    - *interpretált*: modellt beolvassuk és végrehajtjuk
    - *generatív*: forráskód szintézise
    - programozási nyelvek: Java, C, C++
    - *optimalizálás*: memória vs. CPU, megfigyelhetőség vs. teljesítmény
  - dinamikus interpreter vs. kódgenerálás:

Dinamikus interpreter	Kódgenerálás
Gyors indulás, eredmények azonnal	Indulás generálás és fordítás után
Általában idő/memória overhead	Hatékony futtatásra optimalizálható
Futásidejű függőség	Változtatás után újra kell generálni
Tudja támogatni a modell változtatását végrehajtás közben	Kézzel változtatható a kód
Viselkedés mindig megfelel a modellnek	→ *Ez biztos jó?*

- **típusok**
  - ***dedikált kódgenerátor*** (pl. Yakindu)
    - modellező eszköz része
    - tanúsítványozhatóság
    - alacsony testreszabhatóság



- ***sablon-alapú kódgenerátor***
    - gyorsabb fejlesztés
    - komplex változások az életciklus során
        - sablon és a modell függetlenül változik

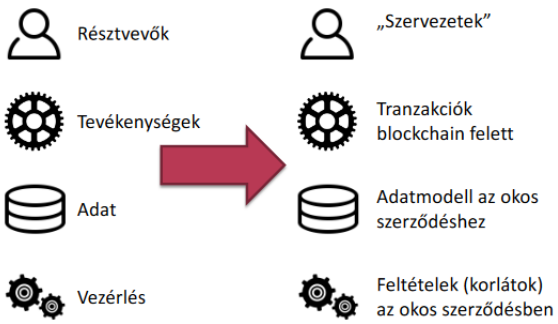


- **funkciók**
    - hatékonyság vs. újrafelhasználhatóság, érthetőség
    - nyomonkövethetőség
        - adott kódrészlet mely modellelem miatt van ott
        - *inkrementalitás* támogatása: csak ott változzon a kód, ahol a modell
        - hatékony újragenerálás
- **generált kód illesztése**
    - csak egy része az alkalmazásnak
    - kézzel nem módosítjuk
    - generált és kézzel írt kód külön file-ba/mappába
    - illesztés: *ragasztó kód* (glue code)
        - generált kód hívása
        - leszármazás generált kódból

4. Üzleti folyamatok végrehajtása blockchain alapon

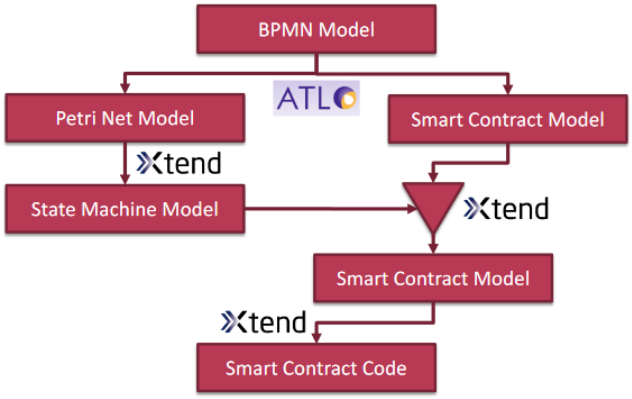
- **röviden**
    - elosztott, letagadhatatlan, többrésztvevős tranzakciók
- **felhasználási területek**
    - (kriptovaluták)
    - árucsere
    - szállítmányok követése

- **motiváció**
  - *modell alapú fejlesztés*
    - magasabb absztrakciós szint
    - produktivitás
      - modellek (szerződésminták) újrahasznosítása
      - tervezés egyszerűsítése
    - automatizált fejlesztés
    - minőségbiztosítás
      - modell validáció, modellellenőrzés, modell alapú tesztelés
  - „*blockchainesítés*”
    - meglevő megoldások (részben) blockchainre vitele
- (BPMN elements in blockchain platforms)



- **célok**
  - üzleti folyamatok alapján okos szerződések generálása
    - nem feltétlen a végrehajtó környezet szintjén
    - nehezen ellenőrizhető
    - kód és specifikáció összerendelése
  - a fejlesztés részleges automatizálása
    - „boilerplate” kód generálása
    - hibalehetőségek számának csökkentése
      - üzleti logika váza
      - meglévő lépések becsatolása (?)

- **transzformációs folyamat**

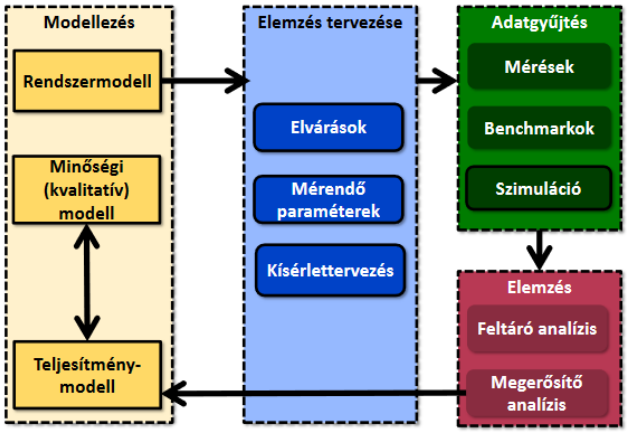


BACK

10. előadás:

Modellek paraméterezése: regresszió, benchmarkok

1. **Rendszermodelltől teljesítménymodellig**

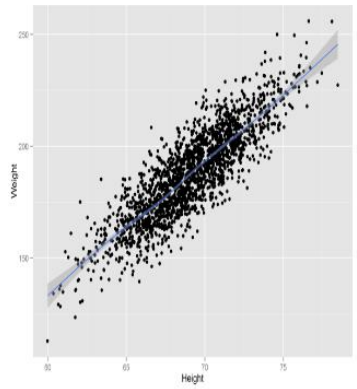


   - **adatok hihetőség**
     - érzékenységvizsgálat
       - a modell kimeneti paraméterei mennyire érzékenyek a bementei paramétereinek változására
       - „parameter sweep”: egy paraméter vizsgálata adott tartományban
     - ***ökölszabály***: adatok hihetősége
       - mérés_bizonytalansága (szórás) ~ mérések_száma^2
       - (kellő mennyiségű adat esetén) → valószínűség-számítás

2. **Matematikai becslések: regressziós módszerek**

   - **regresszió (*f függvény)***
     - bemenet: az attribútumok értéke
     - kimenet: megfigyelések legjobb közelítése
     - „ökölszabály”

- Példa
  - testtömeg/magasság együttes eloszlása



- ***regressziós módszerek***
  - alapelv

$$Y_t = f(\bullet) + \varepsilon_t$$
$$Y = f(X_1, \dots, X_n)$$

    - Y_t: véletlen változó
    - f: közelítés
    - ε_t: hiba
    - Y: jósolt esemény
    - X_n: megfigyelhető változók
  - *átlagos hiba (mean error)*

$$ME = \frac{\sum_{t=1}^{n}(Y_t - F_t)}{n}$$

    - Y_t: becsült érték
    - F_t: mért érték
  - ***lineáris regresszió***
    - egyszerű lineáris függvény illesztése az adatokra

$$Y = a + b \cdot X$$

- *legkisebb négyzetek módszere*
  - keressük azokat az a, b paramétereket, amelyekre

$$SSE = \sum_{t=1}^{n} {\varepsilon_t}^2 = \sum_{t=1}^{n} (Y_t - F_t)^2$$

  - cél:

$$\sum_{t=1}^{n} (Y_t - F_t)^2 = \sum_{t=1}^{n} [(Y_t - (a + b \cdot X_t)]^2$$

  - legjobban illeszkedő egyenes
  - DE: Anscombe's quartet
    - » minőségileg különböző adatok
    - » azonos regressziós egyenes
- *korrelációs együttható* (négyzete)
  - változó becsült és tényleges értékének kapcsolata
  - 0 és 1 közti érték
    - » 0: nincs kapcsolat
    - » 1: függvényszerű kapcsolat
    - » R: $-1 \ldots 1$ (kapcsolat iránya)

$$R^2 = \frac{\sum_{t=1}^{n} \left(F_t - \overline{Y}\right)^2}{\sum_{t=1}^{n} \left(Y_t - \overline{Y}\right)^2}$$

- *nemlineáris módszerek*
  - exponenciális megközelítés

$$Y_t = a \cdot b^t$$

  - Web forgalom növekedéséhez jól illik
  - exponenciális terhelés becslőfüggvény

$$Y_t = a \cdot e^{bt}$$

▪ ***mozgó átlagok módszere***

- rövid távú előrejelzésre jó
- egyszerre egy értéket ad meg

[K59] megjegyzést írt: Itt a: eltolás, b: meredekség

[K60] megjegyzést írt: Sum of Squared Errors, minimális

- becsült érték az utolsó n érték átlaga

$$F_{t+1} = \frac{\sum_{i=t}^{t-n+1} Y_i}{n}$$

- Y_t: t. időpontban mért érték
- F_{t+1}: becsült érték
- n: tipikusan 3 és 10 között van (becslés hibája ne legyen túl nagy)

▪ ***exponenciális csúszóablak***

- egy értéket ad meg, előző méréseket átlagolva
- későbbi mérés (és mérési hiba) nagyobb súllyal
- rövid távú előrejelzésre

$$F_{t+1} = F_t + \alpha(Y_t - F_t)$$

- F_t: t. időpontra becsült érték
- Y_t: t. időpontban mért érték
- $F_t - Y_t$: mérési hiba t. periódusban
- α: súlyozás ($0 \leq \alpha \leq 1$, gyakorlatban $0{,}05 \leq \alpha \leq 0{,}3$)

3. Benchmarking

○ Definíció

▪ ***benchmarkolás***

- egy program (*programok, vagy más műveletek*) futtatása
- szabványos tesztekkel vagy bemenetekkel
- egy objektum relatív teljesítményének felmérése érdekében

▪ ***ismételhetőség (repeatability)***

- benchmarkot lehessen egymás után többször is futtatni
- → mérési eredmények szórása csökkenthető legyen

▪ ***reprodukálhatóság (reproducibility)***

- benchmark legyen hasonló környezetben, hasonló eszközökkel megismételhető

▪ ***érthetőség/általánosított felhasználói eset***

- átlag felhasználó számára értelmezhető legyen az eredmény

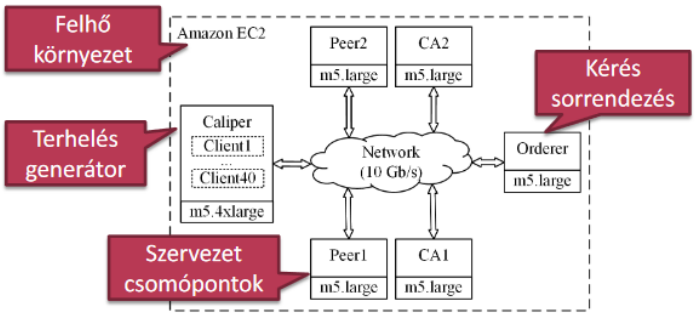
- ***relevancia (relevance)***
  - benchmarkban megvalósított terhelési profil hasonlít arra a valós terhelésre, amely alatt a rendszer teljesítményéről információt szeretnénk kapni
  - biztosítsuk, hogy
    - azt mérjük, amit kell
    - terhelésgenerálás jellege közelítse a valódi terhelést
    - minimalizáljuk a zavaró tényezőket
      - » pl. gyorsítótárak ürítése, többi futó folyamat leállítása
- ***TPC-C***
  - *OLTP benchmark*
    - OLTP (Online Transaction Processing)
    - szoftverek és hardverek OLTP teljesítményét méri
  - *komplex benchmark*
    - több tranzakció típus
    - összetett adatbázis séma
    - széles skálán mozgó adathalmaz méret
  - *mért metrikák*
    - áteresztőképesség ($tpmC$)
    - ár-teljesítmény arány (≈ hatékonyság) $\frac{\$}{tpmC}$
  - **mintaadatbázis**
    - nagykereskedelmi beszállító cég rendelései
    - 9 különböző tábla
  - **5 féle tranzakció típus**
    - rendelés, fizetés, szállítás, rendelés státusza, raktár állapota
    - frissítés, beszúrás, törlés, megszakítás
  - **ACID tranzakciók**
    - Atomicity, Consistency, Isolation, Durability

[K61] megjegyzést írt: Transactions Per Minute

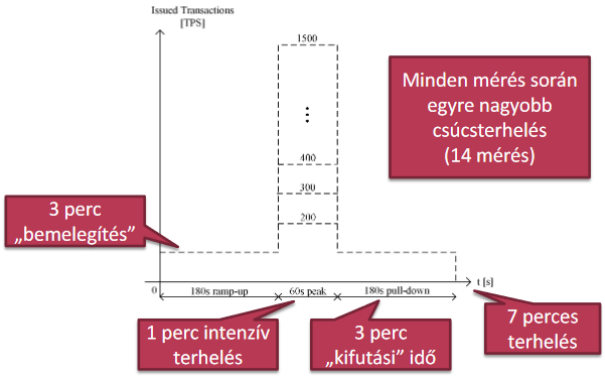
- **változatos felhasználói kérések szimulációja**
  - tranzakció típusok kezdeményezése adott valószínűséggel
- **időkorlát az egyes tranzakció típusokra**

4. **Teljesítménytesztelés**

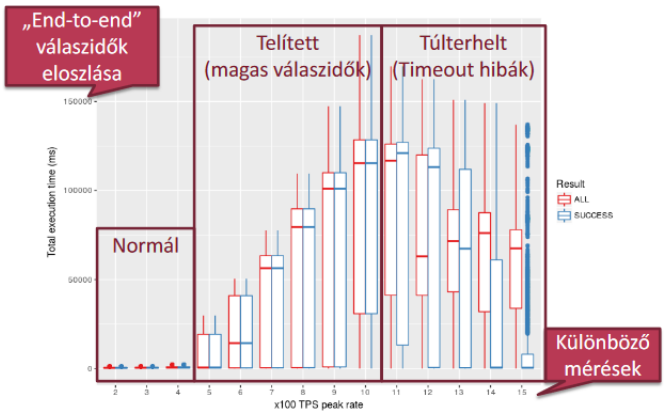
   - Definíció
     - ***Hyperledger Fabric***
       - konzorciális elosztott főkönyv technológia (DLT)
       - komplex infrastruktúra
         - szervezetek, főkönyvek, telepített okos szerződések



     - ***terhelésprofil***



- ***válaszidők eloszlása***



5. Benchmarkolás

- **egy jó benchmark ismérvei**
  - *gyártó-független*
    - különböző gyártók termékeinek összehasonlítása
  - *jól definiált cél*
    - valós felhasználási eseteket takar
  - *megismételhető*
    - a rendszer bármikor determinisztikusan újramérhető
  - *aktívan karbantartott*
    - követi az új technológia trendeket, felhasználási esetek evolúcióját
  - *nyílt szabványokon alapul*
    - szabadon elérhető, szakemberek által átvizsgált, az ipar által elfogadott
  - *egyszerű, általános metrikákat mér*
    - eltérő megvalósítású, azonos célú rendszerek összehasonlítása

- ***Emerald SEPA ICT benchmark***
  - SEPA ICT: Single Payment Area, Instant Credit Transfer – Gyors Nemzetközi Átutalások
  - **Emerald alapelvek**
    - nem lehet egyszeres hibapont
    - a csomópontok földrajzilag elosztottak
    - egy minimum válaszidőt tartania kell
    - szabványos átutalás csomagok használata
    - minden csomópont küld és fogad utasításokat
  - **Emerald metrikák**
    - *késleltetés (latency)*
      - tranzakció nyugtázásáig eltelt idő $[s]$
    - *ráta (rate)*
      - másodpercenként nyugtázott tranzakciók $[\frac{tx}{s}]$
    - *skálatfaktor (scale)*
      - tranzakciókat kezdeményező csomópontok száma
    - *átbocsátás (througput)*
      - átlagos ráta · skálafaktor $[\frac{tx}{s}]$
  - **Emerald eredmények**

Name	Latency	Rate	Scale	Throughput	Badge
Emerald (Irish pilot)	<10	100	6	600	emerald hecto
BitCoin	>10	7	12,000	84k	emerald kilo
SEPA ICT (EBA STEP2)	<10	250	1200	300k	emerald kilo
Ethereum (main net)	>10	16	25,000	400k	emerald kilo
Ripple (XRP)	<10	1500	500	750k	emerald kilo
SWIFT (FIN)	<10	400	11,000	4.4m	emerald mega
Visa	<10	65,000	16,300	1.1bn	emerald giga

Minősítés átbocsátás alapján

DLT implementációk!!

Hagyományos átutalás rendszerek

Konzorciális DLT implementációk (Hyperledger Fabric)? ☺

- ***Cassandra Benchmark***