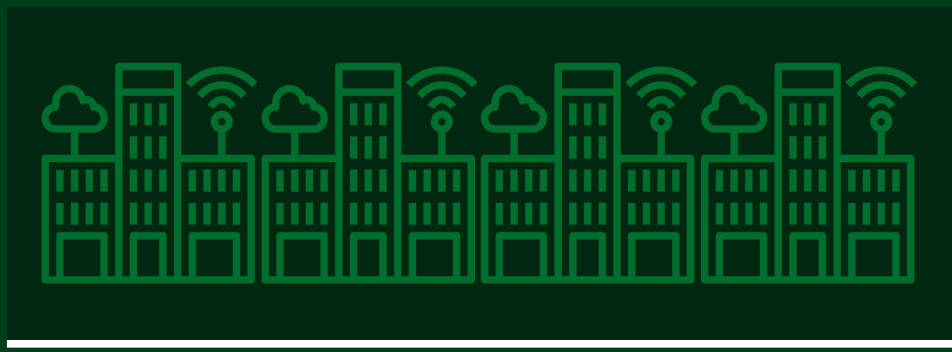
OKOSVÁROS-TECHNOLÓGIÁK XXI.

Mérésalapú teljesítménymenedzsment az okos szolgáltatásokban



Gönczy László – Jakab László –
Klenik Attila – Kocsis Imre

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Szociális Alap

BEFEKTETÉS A JÖVŐBE

NEMZETI KÖZSZOLGÁLATI EGYETEM
BUDAPEST

OKOSVÁROS-TECHNOLÓGIÁK
A technológia fejlődésének irányai és hatása
XXI. kötet

Sorozatszerkesztő:
Sallai Gyula

Gönczy László – Jakab László –
Klenik Attila – Kocsis Imre

MÉRÉSALAPÚ TELJESÍTMÉNY-MENEDZSMENT AZ OKOS SZOLGÁLTATÁSOKBAN

Nemzeti Közszolgálati Egyetem
Közigazgatási Továbbképzési Intézet
Budapest, 2020

A kötet a Nemzeti Közszolgálati Egyetem **KÖFOP-2.1.2-VEKOP-15-2016-00001** „**A jó kormányzást megalapozó közszolgálat-fejlesztés**” projektje keretében, a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Karán létesült „**Okos város – okos közigazgatás**” kutatóműhelyben (2017/162/BME-VIK) készült.

Szakmai lektor:
Patricza András egyetemi tanár, BME-VIK

Szerkesztette:
Gönczy László egyetemi adjunktus, BME-VIK

A kézirat lezárásának dátuma:
2018. június 30.

TARTALOMJEGYZÉK

1. BEVEZETŐ

Jelen tanulmány az okos szolgáltatások folyamattervezésének támogatását mutatja be a folyamatok mérhető minőségi kritériumainak definíciójára, kiértékelésére és javítására koncentrálva. A minőség ebben az esetben elsődlegesen a felhasználó számára érzékelhető, teljesítményben és a folyamat szolgáltatásbiztonságában megjelenő szempontokat, illetve a folyamat esetlegesen megfigyelhető hibáit jelenti. Feltételezzük, hogy egy elosztott környezetben működő, heterogén infrastruktúrán alapuló, külső adatforrásokkal, felhasználói inputtal és adott esetben emberi döntésekkel működő környezetben is cél a felhasználó számára a legmagasabb minőségi normáknak megfelelő szolgáltatást nyújtani, legyen szó akár egy közigazgatási jellegű folyamatról, akár adatkiértékeléssel, felhasználók adatvezérelt megszólításával foglalkozó okos városi folyamatról.
A tanulmány áttekinti a vonatkozó, szoftverminőséggel foglalkozó szabványokat és értelmezi azokat, elsődlegesen okosvárosi környezetben. Ennek eredményeként megfogalmazhatók a rendszerrel szemben támasztott követelmények, és mérések segítségével később kiértékelésük is lehetséges, ezáltal a rendszerről alkotott kép objektívebb lehet.
Ezek után áttekintjük a rendszer teljesítményének mérőszámait, mérési és kiértékelési módszereit. Manapság a legtöbb informatikai rendszerben számtalan napló (log) keletkezik, így akár kiegészítő mérési pontok definiálása (ún. felműszerezés) nélkül is lehetséges a rendszer teljesítményét adatvezérelt módon kiértékelni. A gyakorlatban ennek sokszor az az akadálya, hogy nem állnak rendelkezése a megfelelően megtervezett mérések, illetve az azokból származtatott adatok. A mérések tervezésével szintén foglalkozik a tanulmány.
Ezek után áttekintjük azt, hogy ha (akár túlterheltség, akár funkcionális hibák, akár időszakos hibák vagy támadások miatt) a folyamat egyes elemei nem megfelelően működnek, például egy kérésre nem érkezik válasz, vagy rossz válasz érkezik, akkor ennek mi lehet a hatása, illetve hogyan lehet a hibaterjedést módszeresen modellezni és kiértékelni annak érdekében, hogy a folyamatot már a tervezés során felkészíthessük az esetleges hibákra.
Végül áttekintjük azt is, hogyan lehet a hibatűrő rendszerekben már régóta használt eszköztárat átültetni üzleti folyamatokba, és hogyan lehet ezáltal a folyamatok hibatűrését javítani, milyen, a felhasználó számára is érzékelhető változást jelent, ha egyes, előzetes elemzéssel azonosított kritikus lépéseket lecserélünk ún. hibatűrő tervezési minták alkalmazásával olyan részfolyamatokra, amelyek a különböző hibák ellen legalább részleges védelmet biztosítanak, illetve a hibák terjedését megakadályozhatják, valamint segíthetnek az ún. kárbehatárolási régió (damage confinement region) csökkentésében.

A Szerzők:

Dr. Gönczy László, okl. mérnökinformatikus és mérnök-közgazdász, a Budapesti Műszaki Egyetem Villamosmérnöki és Informatikai Kara Méréstechnikai és Információs Rendszerek Tanszékének adjunktusa. PhD-fokozatát a BME-en 2019-ben szerezte. Fő kutatási területe üzleti és kritikus IT-rendszerek tervezése és elemzése, különös tekintettel adatvezérelt módszerek alkalmazására teljesítmény és teljesítőképesség meghatározásában.

Prof. Dr. Jakab László, a BME Villamosmérnöki és Informatikai Karának egyetemi tanára, dékánja (2016–2019), az Okos város – okos közigazgatás kutatóműhely szakmai vezetője. Villamosmérnök diplomáját a BME-n szerezte meg 1981-ben. A műszaki tudomány kandidátusa 1992-ben, habilitált doktor 2013-ban, az MTA doktora címet pedig 2014-ben szerezte meg. Jelenleg az Elektronikai Technológia tanszék munkatársa.

Klenik Attila, okl. mérnökinformatikus, a Budapesti Műszaki Egyetem Villamosmérnöki és Informatikai Kara Méréstechnikai és Információs Rendszerek Tanszékének doktorandusz hallgatója. Fő kutatási területe az elosztott informatikai infrastruktúrák megfelelő teljesítményre tervezése, illetve meglévő infrastruktúrák teljesítményének elemzése.

Dr. Kocsis Imre, okl. mérnökinformatikus, a Budapesti Műszaki Egyetem Villamosmérnöki és Informatikai Kara Méréstechnikai és Információs Rendszerek Tanszékének adjunktusa. PhD-fokozatát a BME-en 2019-ben szerezte. Fő kutatási területe az elosztott megvalósítású informatikai szolgáltatások ellenállóképességre tervezése. Kutatásainak fő alkalmazási területei az „edge felhő” elemeket alkalmazó kiberfizikai rendszerek, illetve az elosztott főkönyvi rendszerek (Distributed Ledger Technology, DLT).

2. SZOLGÁLTATÁSMINŐSÉG KÖVETELMÉNYEI AZ OKOS VÁROS FOLYAMATAIBAN

Mind az okos város, mind pedig az okos kormányzati alkalmazások célkitűzése olyan komplex szolgáltatások tervezése és karbantartása, amelyeknek a felhasználó számára nyújtott szolgáltatásának minősége garantált.

2.1. A követelményrendszer tervezése

Mint minden komplex rendszertervezési feladat, ez is gyakran összetett – részben akár egymásnak is ellentmondó és közöttük kompromisszumot igénylő követelmények egyidejűleg teljesülését garantáló – rendszertervezési folyamatot igényel.

A követelménytervezési folyamat első lépése a tervezendő rendszer által megvalósítandó logikai-funkcionális követelmények listájának meghatározása.

2. A következő lépés a kapcsolódó extrafunkcionális követelmények (például tervezett felhasználószám melletti teljesítmény) meghatározása.
3. A következő lépés a listában szereplő követelmények számszerűsítése. A tervezés során ugyanis a minőségi kívánalmak teljesíthetősége önmagában igényli mennyiségi jellemzőik meghatározását is. (Egy nagy kapacitású rendszer esetében ugyanis nyilvánvalóan más technikai megoldásokat kell választani, mint egy csak esetenként és egyedi kérésekre használatos esetén).
4. A specifikációs folyamatnak természetesen lényeges eleme a tervezési teret behatároló olyan megszorítások integrálása, mint a rendszer tervezett költsége, hiszen a kényszerek és a korlátozó feltételek együttesen szabják meg a lehetséges megoldások tervezési terét.
5. A mennyiségi jellemzőkkel gazdagított követelménylista alapján már az okos szolgáltatásokra is alkalmazható idő-, költség- és minőségbecslő eljárások állnak rendelkezésre (számos zárt modell mellett például a részleteiben is publikált COCOMO/COSYSMO/COQUALMO család), amelyek képesek a tervezett megvalósítandó projekt bekerülési költségének becslésére. (Ezek teljesen hasonlóak a más műszaki szakmáknál szokásos ökölszabályokhoz. Például egy gyakorlott útépítő beruházó az út hossza, szükséges forgalmi kapacitása, teherbírása, a terep nehézsége stb. alapján jó biztonsággal ad néhány 10 %-os eltérésű becslést a várható beruházási költségre).

A fenti folyamatot gyakran több alternatívára, akár több iterációban is végre kell hajtani, például az igények és költséghatárok összehangolásához.
A fenti tervezési ciklus végeredménye egy olyan, a megvalósítani kívánt rendszerrel szembeni elvárásokkal összhangban levő követelményrendszer, amely a tervezési és megvalósítási, valamint az üzemeltetési folyamat számára alapdokumentum.

A jelen kismonográfia olyan szisztematikus eljárásokat foglal össze, amelyek a rendszer építkezése során a követelmények meghatározását és számszerűsítését támogatják. A fenti folyamat erősen a fókuszba helyezi a követelmények számszerűsítését. Ez különösen fontos az okos * alkalmazások zöménél, hiszen a nyilvános okos alkalmazásoknak gyakori jellemzője, hogy nagyszámú ügyfelet szolgálnak ki. Ennek megfelelően – mint minden tömegkiszolgáló alkalmazást – azok teljesítményét méretezni kell. A szolgáltatás specifikálása így az átbocsátóképesség (az időegység alatt elintézendő ügyek száma), az egyes ügyletek maximális feldolgozási ideje stb. értékek definiálását is igényli.
Hasonlóan, a korábbi kismonográfiákban részletezetteknek megfelelően, minden fontos szolgáltatásbiztonsági kritérium (például a rendelkezésre állás, megbízhatóság) is számszerűsíthető és számszerűsítendő.
A szolgáltatás koncepcionális tervezése során tehát meg kell határozni a mennyiségi követelményeket, majd a részletes tervezés és méretezés során mind a funkcionális, mind pedig a számszerű követelményeket teljesítő rendszert kell létrehozni.
Különösen a kritikus alkalmazások (például segélykérő hívások kezelése és a szükséges bevetések irányítása) teljesítményének méretezésénél annak alapja a feltételezhető maximális terhelés. A rendszerbe annyi erőforrást kell építeni, hogy a reálisan elképzelhető maximális terhelés esetén is garantált legyen zavartalan működése. Ilyen alkalmazásoknál ugyanis a teljesítménykorlátokból fakadó hibás működés (például a szolgáltatás elérhetetlensége) általi kockázat elfogadhatatlan, és mindennemű gazdasági szempontot felülír.

A szokásos alkalmazások esetében azonban a gazdasági hatékonyság és a szolgáltatásminőség, valamint -biztonság között kompromisszumot szokás kötni.

- Ha nagy terhelés felléptekor a felhasználói igény átmeneti visszautasítása és későbbi megismételt benyújtása elfogadható, a méretezés célja a mennyiségi követelmények még éppen megfelelő szintű kielégítéséhez szükséges minimális erőforrásigény biztosítása.
- Abban az esetben, ha a terheléscsúcsok előre jelezhetők (mint például határidős államigazgatási feladatok a hagyományos adóbevallásnál), ezeket az erőforrásokat a várható csúcsnak megfelelően megerősítik.

2.2. Tervezés és üzemeltetés

A szolgáltatás előírt minőségének fenntartására vonatkozó követelmény annak teljes életciklusára érvényes. Az üzemeltetés során a változások két nem teljesen független osztályba osztható okból jöhetnek létre:

- A hosszabb távon érvényesülő változások fő forrása a felhasználói szokások módosulása. Például egy új szolgáltatás bevezetésekor annak elterjedése fokozatosan növekvő terhelést generál mindaddig, amíg annyira közkeletűvé nem válik, hogy elér egy többé-kevésbé telítési jellegű munkapontot.
- Ez azt jelenti, hogy a hosszabb távú működési tapasztalatok (ez a trendszerű változások esetében tipikusan a hosszabb időtartamra vett átlagok, illetve esetlegesen a csúcsok) alapján a rendszer tervezése során tett feltételezéseket időszakosan felülvizsgálva a szolgáltatást megalapozó kapacitást szükség esetén hozzáigazítják az aktuálisan várható igényekhez.
- A rövid idejű változások elsődleges forrása a felhasználói igények (például a terhelés) lökésszerű változása vagy valamely erőforrás meghibásodásból fakadó kiesése miatti kapacitáscsökkenés.

- Ezekre egy jól tervezett, modern okos alkalmazás – a későbbiekben taglaltak szerint – ma már tud reagálni. A fejlett megközelítések a puszta megfigyelésen túllépve szabályozó beavatkozással automatikusan kompenzálják a nemkívánatos eltéréseket. A rövid idejű változásokra való megfelelő reagálás biztosításához pedig a rendszert működési időben folyamatosan monitorozni kell, ellenőrizve a mennyiségi követelmények teljesülését.

Az ilyen megközelítéseket az IBM által a kétezres évek elején kidolgozott autonóm számítástechnika koncepció alapozta meg (IBM 2006).
Ennek alapgondolata a szolgáltatásnak, illetve az azt megalapozó infrastruktúrának az ipari irányítástechnika eszköztárában szokásos felügyelő szabályozássá alakítása.

1. Ez azt jelenti, hogy mind a felhasználónak nyújtott szolgáltatás, mind pedig az alkalmazás működését lehetővé tevő infrastrukturális elemek és szolgáltatások jellemző metrikáit mérőszondákkal begyűjtik.
2. A rendszer központjába helyezett szabályozó logika eltérés esetén beavatkozik, kompenzálva az eltérések negatív hatását.

Ez a megoldás mindmáig koncepcionálisan meghatározó a felhő számítástechnikai szolgáltatásokat nyújtó infrastruktúrák felügyelő jellegű irányításában.
Az autonóm számítástechnika annyiban is túllép a klasszikus megközelítésen, hogy nemcsak defenzív módon reagál az esetleges negatív hatásokra (például egy túlterhelést megelőzendő többleterőforrást biztosít az alkalmazás számára); hanem a terheletlen időszakban képes az aktuálisan felesleges erőforrásokat más feladatokra átcsoportosítani. Az *analízis*, illetve *analízis + optimalizálás* nyújtotta intelligencia dinamikusan képes a szolgáltatás minőségére és biztonságára vonatkozó követelmények teljesítését, illetve a működés gazdaságosságát biztosítani.
Összefoglalva: a szolgáltatásminőség megfelelőségének eldöntéséhez szükséges az, hogy a működés során a rendszer mérje a mennyiségi követelményekhez kapcsolódó mértékeket, sőt ennek a mérésnek meg kell alapoznia a követelményeknek megfelelőséget eldöntő vizsgálatokon túl az esetlegesen szükséges beavatkozás módjának és mértékének meghatározását is. Ezek a mérések közül a felhasználó által érzékelt metrikák a szolgáltatási szint megállapodás (SLA = Service Level Agreement) alapjai.

2.3. A mértékek szerepe

Az okos alkalmazások létrehozása és működtetése kooperatív folyamat. Ennek megfelelően a folyamat résztvevői között szabatos és számonkérhető követelményrendszer alapján kell folytatni a kommunikációt, az egyes lépésekben kiadott feladatok specifikálását, ellenőrzését és elfogadását. Megjegyzendő, hogy az ismertetett szempontok nagy része alapvetően nem szoftverspecifikus, és más folyamatokra is alkalmazható.
A professzionális rendszer- és szoftvertechnika egyik alapelve, hogy a mennyiségi jellemzőkkel rögzített követelmények egyértelműek; feltéve, hogy a mérendő mennyiség definíciója is szabatos.

A következőkben az okos alkalmazások hátterében rejlő és annak intelligenciáját biztosító szoftverek minősége mérésének módszertanára vonatkozó *IEEE 1061-1998 - IEEE Standard for a Software Quality Metrics Methodology* szabvány (IEEE Computer Society SETC 1993) alapján bemutatjuk azt, hogy mi a mennyiségi jellemzők jelentősége az egyes szereplők számára. Történelmileg ez a szabvány volt az első, amelyet az infor-

matikai ipar elterjedten elfogadott és használt. Ma már ezt a szabványt lényegesen kifinomultabb, a közben eltelt két évtized tapasztalatait és technológiai fejlődés trendjeit is ötvöző továbbiak váltották fel, de alapgondolatai mindmáig tükröződnek a szoftveralapú rendszerek minőségbiztosítási folyamataiban.

A fenti szabvány célkitűzése egy, a teljes fejlesztési és üzemeltetési életciklus összes érintett szereplője számára szóló közös és koherens metodika. Ennek megfelelően egységes keretrendszerbe foglalva veszi számba:

- a szoftver beszerzésekor (megrendelésekor),
- specifikálásakor (követelményeinek meghatározásakor),
- fejlesztésekor,
- használatakor,
- a szolgáltatás- és terméktámogatáskor,
- karbantartásakor és
- auditálásakor

végzendő tevékenységeket.

A szabvány ugyanakkor keretjellegű, így nem ad kötelezően alkalmazandó folyamatmodellt, bár függelékében ajánl ilyet. A későbbi fejezetekben egy ilyen elterjedt módszerre adunk példát.

A szabvány fő használati esetei az alábbiak, szereplők szerinti bontásban:
A **megrendelő** számára támogatást nyújt a kialakítandó rendszer szolgáltatásminőségi, illetve -biztonsági követelményeinek azonosításában, szabatos definiálásában, méretezésében és a követelmények fontossági sorrendjének megállapításában.

- Ez a fázis például közbeszerzés esetén azt jelenti, hogy szisztematikus módon összegyűjtötték az összes, a közbeszerzési felhívásban foglalt követelményrendszert megalapozó szempontot.
- A priorizálás jelentősége az, hogy műszaki vagy gazdasági okok miatt egyszerre teljesíthetetlen követelmények esetén vezeti az ellentmondás feloldását. (Hogyan lehet a követelményeket a realizálhatóság szintjére csökkenteni, illetve mely szolgáltatásoknál visszalépni, vagy implementálásukat elhalasztani).

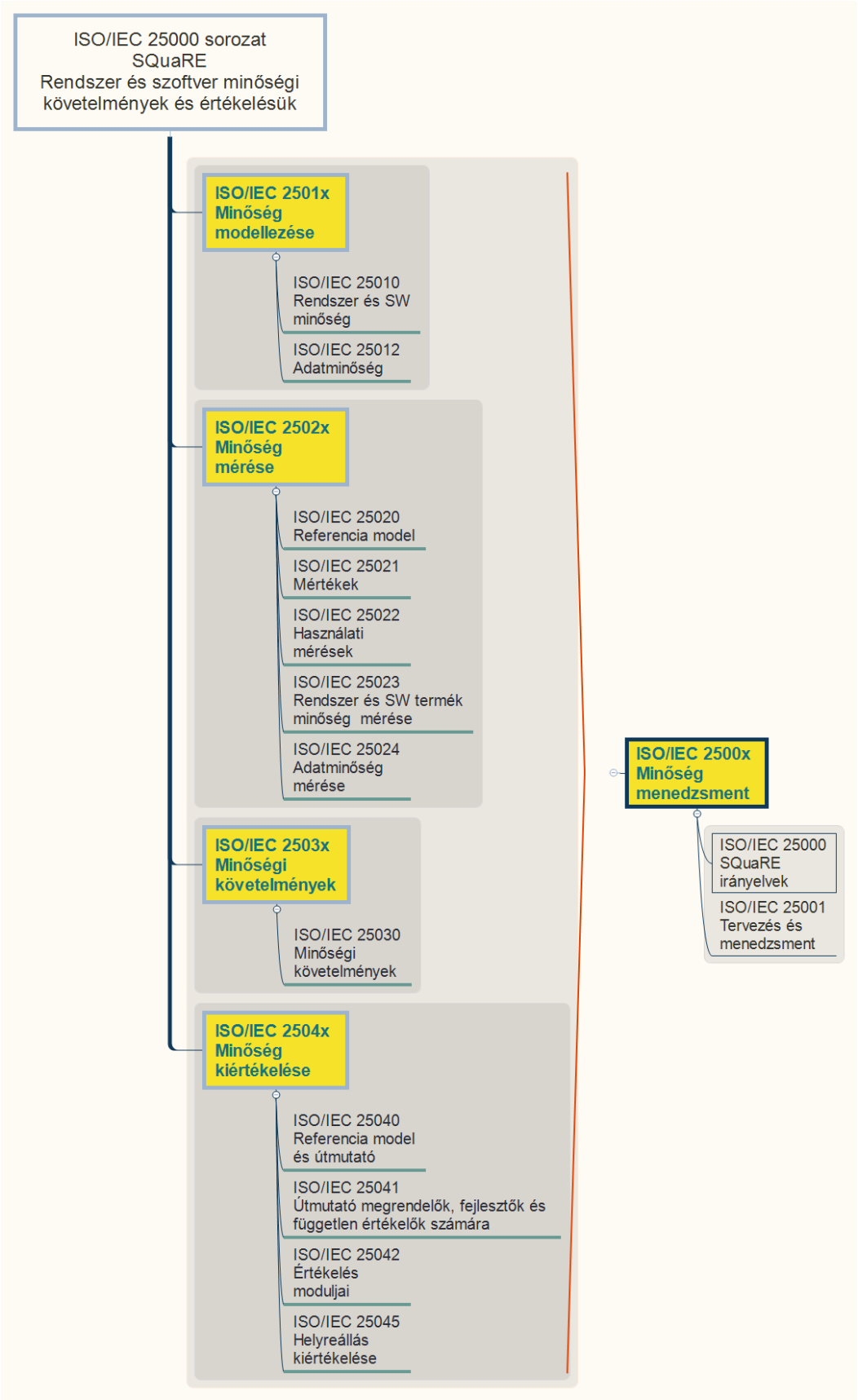
A **rendszerfejlesztő** számára az egyértelműsített követelményrendszer alapul szolgál az alkalmazandó fejlesztési és minőségbiztosítási módszertan kiválasztására, az alkalmazott technikák, bevont komponensek és szolgáltatások lehetséges körének azonosítására és az előzetes műszaki-gazdasági projektterv előkészítésére, illetve ennek alapján a kivitelezési ajánlat kidolgozására.

- Kiemelendő, hogy a projektmenedzsment nemzetközi tapasztalatai alapján a projektek munkaráfordítása, időigénye és elvárható minősége szemszögéből egyaránt a követelményrendszer teljessége, szabatossága és érthetősége a kritikus faktor.
- A tapasztalatok alapján levont és a szoftveriparban széles körben alkalmazott ökölszabály szerint: ha egy hibát csak a megvalósítási folyamat eggyel későbbi fázisában vesznek észre az első lehetséges felfedezéshez képest, akkor a javítás költség-vonzata akár egy nagyságrenddel is nőhet.
- Ennek megfelelően a szabatos követelményspecifikáció jelentős költségmegtakarítást jelent, sőt miután az utólagos belenyúlás mindig minőségrontó, az elvárható szolgáltatásminőség jó megalapozását is szolgálja.

A **minőségellenőrzést,** illetve -**szabályozást** végző, valamint az **auditáló** szervezetek számára a mennyiségi jellemzőkkel ellátott követelményrendszer objektív alapot szolgáltat az annak való megfelelőség eldöntésére.
A **rendszerkarbantartó** számára fókuszálja a figyelmet és ráfordításokat a kritikus jellemzőkre (például mely jellemzők vannak az elfogadhatóság határa közelében), és támogatja a módosítások elvégzését a szolgáltatás fejlődési életciklusa során.
Objektív alapot teremt a **felhasználói** elégedettség, illetve a többletkívánalmak felméréséhez.
A fenti szabvány a minőség attribútumainak fogalmát hozzáköti a mérhetőséghez. Célul tűzi ki, hogy a szoftver egészének elfogadása (elfogadható, marginális/javítható, illetve elfogadhatatlan kategóriába sorolása) az egyes (rész)metrikák értékének ismeretében objektív alapon történhessék.
Az objektív értékelés alapja az egyes metrikák elfogadási tartományának egyértelmű definiálása. Természetesen a szabvány megengedi a közvetlen (mért) metrikák mellett a származtatott metrikák használatát is. A szabvány a metrikák egyértelműsége érdekében magukat a mérési eljárásukat is szabatosan definiálja.

2.4. Az ISO/IEC 25000 szabványcsalád

A szoftvertermékek és a szoftveralapú szolgáltatások előállításának iparszerszerűvé válásával mindinkább szükségessé vált az ennek a területnek a specifikumait figyelembe vevő szabványok kialakítása.
Ma a szoftver és ráépülő szolgáltatások minőségbiztosításában a legkiforrottabb alapot az ISO/IEC 25000 szabványcsalád adja, amely SQuaRE (System and Software Quality Requirements and Evaluation) címen egységes keretbe foglalja a szoftver- és a szoftveralapú szolgáltatások minőségének metrológiáját. Ennek főbb elemeit az 1. ábra mutatja (ISO/IEC 25012:2008 2008; ISO-IEC 25010:2011 2011; ISO-IEC TS 25011:2017 2017).
A szabványcsalád fő elemei egységes megközelítésben (ISO/IEC 2500x SQuaRE) a minőség modellezésével (ISO/IEC 2501x), mérésével (ISO/IEC 2502x), a minőségre vonatkozó követelmények megfogalmazásával (ISO/IEC 2503x) és kiértékelésével (ISO/IEC 2504x) kapcsolatos alapokat foglalja egységes keretbe), ezen kívül léteznek kisebb jelentőségű speciális kiegészítő szabványok is a családban ISO/IEC 2505x-2509x).
A szabvány négy fő blokkja (ISO/IEC 2501x-2504x), megalapozza a minőségre tervezés tipikus folyamatmodelljét, a minőségi követelmények definiálását és megfogalmazását, a tervezési-fejlesztési idejű alkalmazását, illetve kiértékelését (1. ábra).



1. ábra. Az ISO/IEC 25000 szabványcsalád.
Forrás: a szerzők saját szerkesztése

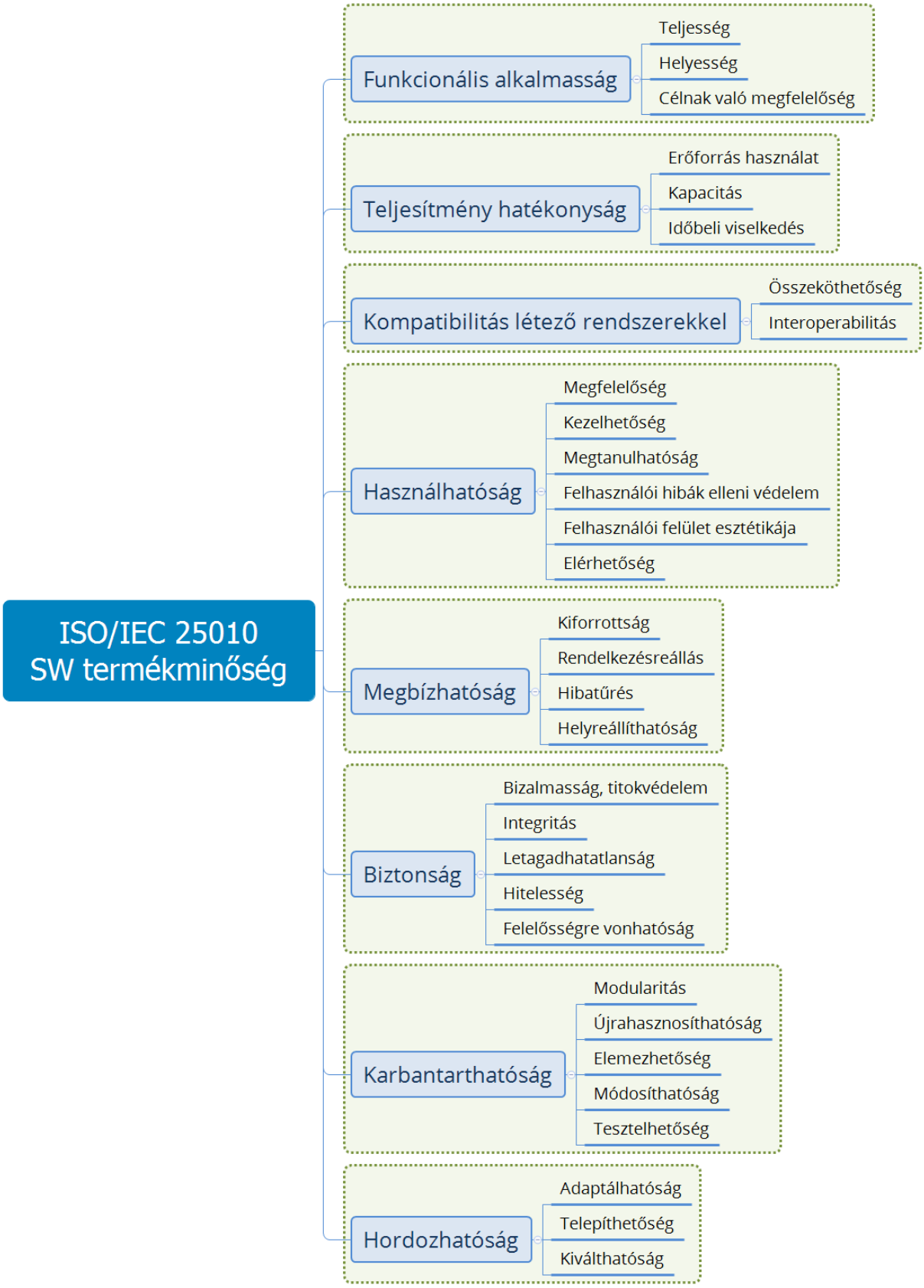
A szabvány megkülönbözteti a felhasználó által érzékelt ún. használati tulajdonságokat és mértékeket, illetve a termék vagy szolgáltatás technikai minőségjellemzőit (2. ábra).

2. ábra. Minőségi mértékek csoportjai.
Forrás: a szerzők saját szerkesztése

A műszaki specifikáció tervezési folyamata így akár interpretálható úgy is, mint a felhasználónak nyújtandó szolgáltatásminőség fokozatos leképezése az okos szolgáltatást megvalósító termékek és szolgáltatások termékminőségi tulajdonságaira és metrikáira. Komplex követelményrendszerek esetében – mint minden komplex rendszernél – alapvető annak teljessége és ellentmondásmentessége. A komplexitás kezelésének szokásos módja a dekompozíció, a probléma kisebb részekre bontása.
Maga a szabvány is mind a felhasználó által érzékelt használati tulajdonságokat és mértékeket, mind a termék vagy szolgáltatás technikai minőségjellemzőit különböző aspektusokra, az azokhoz rendelt tulajdonságokra és mértékekre bontja.
A használati tulajdonságok átfogják a tipikus szolgáltatások fő extrafunkcionális tulajdonságait. Hozzáteendő, hogy számos részterületen ezek a tulajdonságok mintegy „behivatkoznak” alkalmazásiterület-specifikus szabványokat is (például az élet- és vagyonvédelem aspektusánál az estleges katasztrófák elkerülését garantáló biztonságosság esetében egész szabvány család létezik).
Jól látható ugyanakkor, hogy a szabvány célul tűzi ki az objektíven mérhető, illetve becsülhető gazdasági-technikai jellegű mértékek mellett a humán felhasználók igényeinek számbavételét is.

3. ábra. Felhasználó által észlelt szolgáltatásminőség.
Forrás: a szerzők saját szerkesztése

Az ilyen szolgáltatási jellemzők meghatározása a fejlesztés során problematikus lehet, hiszen a tényleges és méréssel igazolt értékek csak a használatbavétel után derülnek ki (még akkor is, ha a szolgáltatást próbaüzem keretében megvizsgálják). Ilyen esetben a leképezés elsősorban a „jó ipari gyakorlat" benchmarkszerűen újraalkalmazott mértékeinek átvételén alapul.
A szolgáltatásokra és szoftvertermékekre vonatkozó szempontrendszer ugyanakkor alkalmas a rendszerintegrációval létrejövő szolgáltatások kezelésére is. Az okos alkalmazások ugyanis nemcsak földrajzilag fognak át egy nagyobb régiót, hanem jellegzetesen szervezetileg is bonyolultak. Az okos alkalmazások gyakran több résztvevő által nyújtott szolgáltatások integrációjával, illetve különböző forrásból származó termékek és adatok fúziójával jönnek létre. Ennek megfelelően az okos szolgáltatások specifikálása, implementálása, felügyelete és karbantartása a szereplők közötti kapcsolódási felületen számon kérhető, precíz specifikációt igényel. A rendszer kiadódó mértékei a részmetrikákból gyakran csak egy igényes, matematikailag is megalapozott modellezési és kiértékelési eljárással származtathatók, amint erre a további fejezetek példát adnak.
Ezért a követelményrendszer tervezése során a felhasználók számára kínált okos szolgáltatás funkcionalitását, szolgáltatásminőségi, valamint -biztonsági mértékének definiálása után, azokat fokozatosan finomítva és hierarchikusan felbontva lehet az egyes (rész-) szolgáltatásoktól elvárt minőséget meghatározni.
A szoftverfejlesztésben szokásos egymást kiegészítő és támogató módon a folyamat- és termékmetrikák használata (4. ábra). Kialakult gyakorlata van az elsődleges mért termékmetrikák alapján (például a szoftverben adott idő alatt felfedett hibaszám) végzett statisztikai szolgáltatásminőség-becslésnek is. Ezeket a speciális megfontolásokat a mértékrendszer csak eredményükben tükrözi.
Az okos rendszerek esetében gyakran kívánalom a szolgáltatások minőségének és biztonságának folyamatos ellenőrzése és garantálása, azaz a metrikarendszer nemcsak a fejlesztési, hanem az üzemeltetési fázisra nézve is követelményeket szab. Az okos rendszerek mindinkább beépített szolgáltatásként támogatják a metrikák monitorozását, értékelését és naplózását, sőt az elfogadhatatlan eltéréseknél a beavatkozást.



4. ábra. A szoftver termékminőségének szempontrendszere.
Forrás: a szerzők saját szerkesztése

2.5. A metrikarendszer kialakításának folyamata

A korábbiak szerint a jó követelmény- és metrikarendszer kritikus az okos szolgáltatások kialakításának és működtetésének folyamatában.
A továbbiakban egy olyan, a követelményrendszerek tervezésében klasszikusnak számító megközelítést mutatunk be, amely egyszerűsége, szervezettsége és közérthetősége miatt mindmáig az egyik legelterjedtebben alkalmazott metodika.
A Cél-Kérdés-Mérték (GQM = Goal-Question-Metric) paradigmát Basili és Rombach eredetileg a NASA egyik projektjének értékelésére dolgozták ki a nyolcvanas évek végén (Basili–Rombach 1988), de azóta alkalmazása általánossága és közérthetősége miatt messze a szoftveriparon túlnyúlóan is széleskörűen elterjedt.
A GQM-módszer alapgondolata az, hogy a megvalósítandó szolgáltatás **céljából** kiindulva az egyes aspektusokat tükröző és mennyiségileg jellemezhető **kérdések** halmazán keresztül valósítja meg a követelményrendszer számszerű **mértékekkel** történő kiegészítését.

3. TERHELÉSMODELLEZÉS ÉS TELJESÍTMÉNYELEMZÉS

A teljesítményelemzés több szempontból is kritikus részét képezi egy rendszer fejlesztésének. Az általános rendszerfunkcióktól eltérően az előírt teljesítménykényszereknek való megfelelést már a tervezés legelső fázisaiban is figyelembe kell venni, ugyanis utólagos garantálásuk jellemzően jelentős idő- és költségbefektetéssel jár. A teljesítményelemzésnek azonban több célja is lehet (FEITELSON 2015).

Egyrészt, a tervezés során előfordulhat, hogy több, funkcionálisan azonos rendszerterv közül kell kiválasztani a ténylegesen megvalósítandót, amelyek azonban jelentősen eltérhetnek az extrafunkcionális követelményeknek (mint például teljesítményelőírásnak) való megfelelésük tekintetében. Ilyenkor az egyes rendszerterv-alternatívákat ki kell értékelni, például a (várhatóan) nyújtott teljesítményük szempontjából, még a költséges implementációs fázis előtt, és a legjobban teljesítőt (figyelembe véve az idő- és költségkényszereket) kiválasztani.

Másrészt, egy már elkészített rendszer esetén is végezhetünk teljesítményelemzést, amely során meghatározhatjuk a rendszernek (vagy egy harmadik fél által szolgáltatott alrendszernek/komponensnek) azt a konfigurációját (azaz a beállítható paramétereinek értékeit), amellyel a legjobb teljesítményt nyújtja. Ez a legtöbb esetben nem teljesen automatikus folyamat, ugyanis a paraméterek és felvehető értékeiknek nagy száma miatt az összes paraméterérték-kombináció elemzése nem lehetséges. Ilyenkor szakterület-specifikus szakemberek bevonásával és múltbeli tapasztalatok segítségével szűkíteni kell a paraméterérték-kombinációk számát.

Harmadrészt, egy rendszer éles használatba helyezése előtt lehetőségünk van teljesítményelemzés segítségével felmérni, hogy a (megfelelő architektúraterv alapján épített és helyesen konfigurált) rendszernek mekkora erőforrás-kapacitás szükséges ahhoz, hogy megfeleljen a vele szemben támasztott követelményeknek. Ezt a lépést tekinthetjük speciális eseteként a megfelelő konfiguráció megkeresésének, csak most a konfiguráció alatt a szükséges hardvereszközök mennyiségének és minőségének meghatározását értjük.

Jellemzően három fő tényezőt kell figyelembe vennünk, amikor egy számítógépes rendszer teljesítményéről beszélünk:

- a rendszer felépítését, amely architekturális szinten befolyásolja a teljesítményt,
- a rendszer implementációját, ahol a tényleges (programozás szintű) megvalósítás részletei hatnak ki a teljesítményre,
- valamint a terhelést, amelynek kitesszük a rendszert működés közben.

Az első tényező esetén számtalan, széles körben ismert rendszer- és szoftvertervezési minta áll rendelkezésre (Gamma 1995), amelyek alaposan megtervezett és bevált megoldásokat szolgáltatnak gyakran előforduló tervezési problémákra. A második pont során felmerülő kérdésekre és problémákra az algoritmuselmélet területe (Sedgewick–Wayne 2011) szolgáltat olyan adatstruktúrákat és algoritmusokat, amelyek hatékonysága és helyessége bizonyított, mind elméleti, mind alkalmazási szemszögből.
Jelen fejezet a ritkán hangsúlyozott, de ugyanannyira fontos harmadik tényezővel, a rendszer terhelésével és annak modellezésével, elemzésével foglalkozik. Egy elkészített rendszer a környezetével együtt alkot teljes egészet, így ha a fejlesztés során ezt a környezetet nem megfelelően (vagy kimondottan rosszul) becsüljük meg, akkor a teljesítményelemzés során levont következtetéseink használhatatlanok lesznek. Továbbá elképzelhető, hogy az elkészített rendszer jól teljesít egy adott terhelés során, viszont rosszul egy másik terhelési minta hatására. Tehát bátran állíthatjuk, hogy a megfelelő terhelésmodellezés elengedhetetlen része egy jól funkcionáló, robusztus és alkalmazkodóképes rendszer megépítéséhez. Robusztusság alatt azt értjük, hogy a rendszer a várttól eltérő környezeti tulajdonságok vagy bemenetek esetén is helyes, a specifikációnak megfelelő viselkedést tanúsít.

3.1. Terhelések karakterizálása

Amikor egy rendszer munkaterheléséről beszélünk, akkor nem egy egyértelműen behatárolt fogalomról van szó. A terhelés tulajdonságai és céljai alapján több aspektust is érdemes a modellezés és elemzés során szem előtt tartani. A következő alfejezetek néhány jellemző típusát mutatják be a munkaterheléseknek.

3.1.1. Szakterület-specifikus terhelések

Attól függően, hogy mi a célunk az elemzés és kiértékelés során, más és más részletességi szinten kell modellezni a rendszer terhelését. Ha például egy operációs rendszer ütemezési stratégiáit szeretnénk magas szinten kiértékelni, akkor a terhelést több, bizonyos ideig futó folyamatok együtteseként modellezzük. A „munkaelemeknek” tehát csak azt a tulajdonságát vesszük figyelembe, hogy mennyi ideig tartanak.
Ha azonban a terhelésnek a processzorra gyakorolt hatását szeretnénk vizsgálni, akkor sokkal részletesebben kell modellezni az egyes folyamatokat. Ennek oka, hogy processzor szinten már olyan tényezők és fogalmak is megjelennek, mint például memória-hozzáférési mintázatok és CPU-utasítások, amelyek például hatást gyakorolnak a gyorsítótárak eredményes kihasználására, vagy a döntési elágazások prediktálásának (Yeh–Patt 1991) a hatékonyságára, jelentősen befolyásolva a végrehajtás teljesítményét. Befolyásoló tényezők hasonlóan széles tárháza jellemző a háttértárak teljesítményelemzésére is.
Ezek alapján már a teljesítményelemzés elején el kell dönteni az elemzés célját, hiszen a munkaterhelés-modell részletességének szükséges szintje befolyásolja a teljes elemzés folyamatát, kezdve az adatgyűjtés részletességétől egészen a konkrét modellek absztrakciós szintjének meghatározásáig.

3.1.2. Statikus és dinamikus terhelések

A teljesítményelemzés során fontos tényező a munkaterhelést alkotó elemek/feladatok beérkezésének a gyakorisága, azaz rátája. A terhelés „végessége” alapján kétféle típusról beszélhetünk: statikus és dinamikus terhelésről.
A *statikus* terhelések során egy előre rögzített munkamennyiséget dolgoz fel a rendszer, és ha ezzel végzett, akkor befejeződött a kiértékelés. Ha viszont a terhelés folyamatosan („végtelenítve”) érkezik, tehát soha nem fejeződik be, akkor *dinamikus* terhelésről beszélünk. A dinamikus terhelések részletesebb rendszerelemzést tesznek lehetővé, ugyanakkor használatuk jellemzően bonyolultabb a következők miatt:

- Egy dinamikus terhelés definiálásához az összes lehetséges munkaelem-típus azonosításán kívül szükség van azok gyakoriságának eloszlására, ami nem feltétlenül áll rendelkezésünkre.
- Statikus terhelés esetén egy „frissen indított” rendszer teljesítményét nézzük, ami általában nem tükrözi a rendszer hosszú távú (stacionárius) teljesítményét. Dinamikus terheléssel lehetőség nyílik olyan anomáliák tanulmányozására is, amelyek például csak a rendszer elöregedése során mutatkoznak (Smith–Seltzer 1997; Trivedi–Vaidyanathan 2002; Denning 2006).

3.1.3. Benchmarkok

Az informatikai rendszerek fejlesztése során egyre jellemzőbb az általános célú kommerciális („off-the-shelf”) komponensek felhasználása ahelyett, hogy a fejlesztők saját maguk valósítanák meg újra a már létező funkciókat. Egyazon funkcióra azonban több alternatív implementáció is létezik. Szükség van egy módszertanra, amely objektív módon összehasonlítja az egyes megoldásokat, segítve ezáltal a választást.
Egy ilyen összehasonlítás helyességének nélkülözhetetlen alapja, hogy az egyes megoldásokat egyforma körülmények között értékeljék ki, beleértve ebbe az azonos munkaterhelést is. Egy *benchmark* nem más, mint egy szabványos munkaterhelés, amely objektív összehasonlítási alapot biztosít különböző komponensmegvalósítások kiértékelése során. A benchmarkok jellemzően statikus munkaterhelések, amelyek jól definiált munkaelemek „visszajátszásából” állnak.

Egy jó benchmark (Huppler 2009) a következő tulajdonságokkal rendelkezik:

- Gyártófüggetlen, lehetővé téve különböző gyártóktól származó termékek összehasonlítását.
- Jól meghatározott célja van, azaz valós használati eseteket fed le.
- Megismételhető, ezáltal a tesztelés alatt álló rendszer teljesítménye determinisztikus módon felmérhető.
- Aktívan karbantartott, követi az új technológiai trendeket, és szükség esetén frissített, alkalmazkodva ezáltal a tipikus használati esetek fejlődéséhez.
- Nyílt szabványokon alapszik, ami azt jelenti, hogy a benchmark publikusan elérhető, specifikációja és implementációja lektorált, szakmailag elbírált, valamint az ipar által összehasonlítási alapként elfogadott.
- Egyszerű, általános metrikákat definiál. Ez teszi lehetővé különböző módon megvalósított rendszerek összehasonlítását, egészen addig, amíg ugyanazt a használati esetet fedik le.
- Adaptív, tehát nemcsak egyetlen esetre használható, hanem újrahasználható különböző környezetekben.

A fenti pontoknak eleget tevő benchmarkok definiálása kihívással teli folyamat (Weicker 1991), így erre a feladatra több független szervezet is létrejött. A két legismertebb a Systems Performance Evaluation Consortium (SPEC[1]) és a Transaction Processing Performance Council (TPC[2]).
A SPEC számítógéprendszerek kiértékelésére alkalmas benchmarkcsomagokat definiál, amelyek közül talán a legismertebb a processzorarchitektúrákat célzó SPEC CPU-csomag. A TPC benchmarkjai leginkább adatbázisrendszerek összehasonlítását teszik lehetővé. A TPC-C benchmark például online tranzakciókat definiál eladók és raktárak között, amihez szolgáltat egy platformfüggetlen adatbázissémát is, amely megvalósítható különböző adatbázis-kezelő rendszerekben.

3.2. Munkaterhelések modellezése

A benchmarkok pontos munkaterhelést definiálnak, amelyek aztán különböző platformokon valósulnak meg, ezáltal közös alapot szolgáltatva az objektív összehasonlításukra. Azonban ha nincs szükség ennyire részletekbe menő munkaterhelés-meghatározásra, akkor lehetőségünk nyílik bizonyos általánosításokat tenni. A modellezés általános értelemben véve egy dolog egyszerűsített, a felesleges részleteket nélkülöző leírása.
Ennek megfelelően a munkaterhelés-modellezés során a célunk egy egyszerű, általános modell megalkotása, amely alapján aztán mesterséges terheléseket tudunk generálni. A generált terhelés esetén továbbra is fontos, hogy megfelelő módon reprezentálja a valós rendszereknél előforduló terheléseket, hiszen továbbra is teljesítményfelméréseket szeretnénk végezni, csak ezúttal egy egyszerűsített terhelésmodell alapján.

3.2.1. A terhelésmodellezés módszertana

Nem újkeletű felismerés, hogy a terhelésmodellezés alapjául a tényleges terhelés során mért adatok kell hogy szolgáljanak (Agrawala–Mohr–Bryant 1976; Ferrari 1972; Sreenivasan–Kleinman 1974). A mért adatok alatt jellemzően az előforduló események bizonyos tulajdonságainak naplózását értjük, mint például a beérkezési idejük és erőforrásigényük. A mérésalapú megközelítés fontossága abban rejlik, hogy a valós terhelések általában különböznek a feltételezett, tisztán matematika modellektől, és ez a különbség nem elhanyagolható.

Ha rendelkezésre áll az időbélyegekkel ellátott eseménynapló a megfigyelt terhelésről, akkor jellemzően két alternatív megközelítés alkalmazható a további teljesítményelemzéshez (Cheng 1969; Ferrari 1972; Sherman–Baskett–Browne 1972):

- Az eseménynaplót egy az egyben felhasználjuk egy szimulációhoz, amely során „visszajátsszuk” a rögzített eseményeket.
- Bizonyos általánosításokkal és egyszerűsítésekkel élve elkészítünk egy terhelésmodellt, amely mind matematikai analízis, mind szimulációs célokra is használható.

Eseménynapló-visszajátszást leginkább akkor alkalmazunk, ha a terhelés túl bonyolult ahhoz, hogy egy egyszerűsített modell keretében meg tudjuk őrizni a reprezentativitását. Tipikusan ez az eset áll fent, ha gyorsítótárak terveit szeretnénk elemezni, amikor is

[1] https://www.spec.org/
[2] http://www.tpc.org/default.asp

az alkalmazás memória-hozzáférési mintája meglehetősen bonyolult (Kessler–Hill–Wood 1994; Koldinger–Eggers–Levy 1991; Smith 1982; Uhlig–Mudge 1997).

A terhelésmodellezésnek az a célja, hogy az eredeti alkalmazás vagy alkalmazási kör jellemzőinek megfelelő, új és reprezentatív terhelésmintát lehessen létrehozni. Eltekintve attól a (nem feltétlenül elhanyagolható) ténytől, hogy a részletes adatgyűjtés költséges vagy nem is mindig megoldható, a modellezés több szempontból is előnyösebb:

- Ha adott egy számítógépes hálózat, amely 100 gépből áll, akkor az erről a rendszerről gyűjtött eseménynapló nem alkalmazható egy olyan hálózatra, amely például 50, vagy 200 gépből áll.
  A modellezés során azonban lehetőség nyílhat általánosításra (absztrakcióra), és például a hálózatban fellelhető számítógépek számát a modell egyik paraméterévé tehetjük. Ezután a modellt tetszőleges méretű hálózatok elemzésére is használhatjuk (természetesen csak kellő körültekintéssel, hiszen a modell érvényességének egyéb feltételei is lehetnek).
- Egy népszerű szolgáltatás napi forgalmának naplója akár több millió bejegyzést is tartalmazhat. Nyilvánvaló, hogy ebben az esetben nem az egyes bejegyzések pontos tulajdonságainak az ismerete lényeges, hanem a forgalom mögött fellelhető globális mintázat (például napi csúcsidőszakok vagy szezonális viselkedések a felhasználók részéről).

Ezeket a mintázatokat szeretnénk statisztikai módszerekkel úgy közelíteni, hogy lehetőleg minél kevesebb paraméter használatával egy reprezentatív, de az eseménynaplónál jelentősen egyszerűbb modellt kapjunk.

3.2.2. Terhelésmodellezés teljesítmény kiértékeléshez

A terhelésmodellezésnek az a célja, hogy az eredeti alkalmazás vagy alkalmazási kör jellemzőinek megfelelő új és reprezentatív terhelésmintát lehessen létrehozni. A rögzített eseménynaplót vagy az abból származtatott statisztikai modellt használjuk fel arra, hogy a vizsgálandó rendszer környezetét „szimuláljuk”, hiszen a teljesítménykényszereknek való megfelelésről kizárólag akkor beszélhetünk, ha előre rögzítjük a rendszerre jellemző különböző terhelési (felhasználók által tanúsított viselkedés) profilokat.
Nem elhanyagolható a kérdés, hogy mikor érdemes a rögzített eseménynaplót használni a teljesítmény kiértékelése során, illetve mikor előnyösebb az eseménynapló alapján felállított statisztikai modelleket alkalmazni.
A legjelentősebb előnye a visszajátszás-alapú teljesítményelemzésnek, hogy hűen tükrözi a rendszer valós terhelését. Ez első olvasatra egyértelműnek tűnhet, viszont ne feledkezzünk el arról, hogy a rögzített események olyan rejtett mintázatokat is magukban foglalnak, amelyekre a rendszer fejlesztői és a kiértékelést végző szakemberek eleinte (vagy egyáltalán) nem is gondolnának.
Ha modellek alapján végeznénk a teljesítményelemzést, akkor ezek a kimaradt (például felhasználói viselkedés) mintázatok torzíthatják, vagy akár félrevezetővé tehetik a kiértékelés után levont következtetéseket, amelyekre jelentős üzleti döntések épülhetnek. A visszajátszás során erre nem kell figyelni, az eseménynapló tartalmaz minden részletet, akár tisztában vagyunk velük, akár nem.

Ennek ellenére a hosszú távon bevált gyakorlat mégis az, hogy az eseménynapló alapján a terhelést statisztikailag jól reprezentáló modelleket készítünk, ugyanis számos előnyük van a visszajátszott naplóhoz képest (Downey–Feitelson 1999; Feitelson 2002; Singh–Segall 1982):

- rugalmasságuk,
- célzott és behatárolt módosíthatóságuk,
- alkalmasabbak érzékenységvizsgálat végzéséhez,
- méréstechnikai szempontból könnyebben kiértékelhetők,
- általánosabbak és/vagy általánosíthatók
- és tömörebbek.

A statisztikai modellek *rugalmasabbak*, mint az eseménynaplók. Az állítás igazolásához elég azt végiggondolni, hogy pontosan mit is jelent az, hogy rögzített eseményekkel dolgozunk. Ahhoz, hogy előállítsunk egy teljesítményelemzéshez használható eseménynaplót, megfigyelünk egy rendszert működés közben, és feljegyezzük a rendszer határához érkező kéréseket.
Ebből az következik, hogy a rögzített terhelés egyetlen adott, konkrét rendszerre specifikus. Ez a rendszer lehet egy korábbi, hasonló rendszer, amelynek éppen egy új verziója van fejlesztés alatt, vagy az aktuális rendszer egy konfigurációja. Vegyük példának azt az esetet, amikor az új, online ügyintézést biztosító rendszerünket ki szeretnénk értékelni a korábbi rendszer egy olyan konfigurációjában rögzített eseménynapló alapján, amely során a kiszolgálóhálózatunk 40 számítógépből állt.
Ha viszont az eseményeket az új rendszer egy olyan konfigurációjának a kiértékelésében szeretnénk felhasználni, amely több (vagy kevesebb, de teljesítményben „erősebb”) számítógépet tartalmaz, akkor az eredmény nem biztos, hogy érvényes/használható lesz az újrendszerre.
Ellenben ha statisztikai modelleket alkalmazunk a kiértékelés során, akkor egy helyesen felépített modellnek akár paramétere is lehet a kiértékelendő rendszer mérete (tehát például a kiszolgálóhálózatot alkotó számítógépek száma). Ha rendelkezésre áll egy ilyen modell, akkor tetszőleges méretű rendszer kiértékeléséhez tudunk megfelelő terhelést generálni.
Következő előnye a statisztikai modelleknek, hogy a terhelést könnyen *célzott és behatárolt módosítások*nak vethetjük alá. Egy eseménynapló esetén nem triviális, hogy milyen változtatásokat kell eszközölni, hogy egy új, de valamilyen szempontból mégis eltérő terhelésprofilt kapjunk.
Kézenfekvőnek tűnhet például a terhelést úgy növelni, hogy a rögzített események között eltelt időt csökkentjük, tehát az események/kérések előfordulási rátáját növeljük. Azonban ez magában rejti azt a veszélyt, hogy az eredeti időtartamra jellemző viselkedési mintákat is módosítjuk.
Például ha egy eseménynaplóban megfigyelhető az a jellegzetes minta, miszerint a nappali terhelés jelentősen nagyobb, mint az éjszakai terhelés, akkor a kérések beérkezési rátájának növelésével (tehát az eseménysor „összenyomásával”) a nappali/éjszakai terhelésmintát is torzítjuk. Egy statisztikai modell segítségével lehetőségünk nyílik úgy növelni a rendszer terhelését, hogy emellett a viselkedési minták „arányosságát” (mind időben, mind pedig intenzitásban) is megtartjuk.
A modellek módosíthatóságának (vagy konfigurálhatóságának) egyik legjelentősebb felhasználási módja az úgynevezett érzékenységvizsgálat. Ilyenkor szisztematikusan változtatjuk a modell egy vagy több paraméterét, és megfigyeljük a változások valamely rendszer szintű jellemzőre (például teljesítményre) gyakorolt hatását.

Az érzékenységvizsgálatnak több célja is lehet. Ha azt tapasztaljuk, hogy egy modellparaméter változtatásával a megfigyelt rendszerjellemző egyáltalán nem, vagy csak nagyon kis mértékben változik, akkor ez a paraméter lehet, hogy elhagyható a modellből, ezáltal egyszerűsítve azt. Ha viszont az derül ki, hogy egy paraméter értékének változására jelentősen változik például a rendszer teljesítménye is, találtunk egy olyan paramétert, amelyre „érzékeny" a rendszer.
Az érzékeny paraméterek több szempontból is kiemelt szerephez jutnak. Egyrészt az ilyen paraméterek (tehát az általuk reprezentált környezeti viselkedés) megváltozását az éles rendszerben folyamatosan figyelni és jelezni kell, hiszen már egy kis változás is nagy hatással lehet a rendszerre. Másrészt az érzékeny paramétereknek már a rendszertervezés során is nagy a jelentősége.
Modellezéskor nem biztos, hogy minden paraméter értékét ismerjük, vagy pontosan meg tudjuk becsülni. Ha az érzékenységvizsgálat eredménye az, hogy a rendszer jelentősen érzékeny egy olyan paraméterre, amelyet csak megbecsülni tudtunk, akkor plusz költséget és időt kell áldozni arra, hogy ezt a paramétert már tervezéskor sokkal jobban meg tudjuk becsülni. Felhasználói viselkedések becslése esetén ennek egyik módszere lehet például egy részletes közvéleménykutatás a nyújtott szolgáltatással kapcsolatban.
A terhelésmodellek *méréstechnikai szempontból* is előnyt élveznek a rögzített eseménynaplókkal szemben. A rendszer teljesítményének kiértékelését többször is el kell végezni ahhoz, hogy statisztikailag megalapozott következtetéseket tudjunk levonni. Ha csak egy eseménylog áll rendelkezésünkre, akkor kizárólag ugyanazon terhelés mellett tudjuk újra kiértékelni a rendszert.
Azonban ha rendelkezünk egy modellel a terhelésről, akkor lehetőségünk nyílik a kiértékelést többször is megismételni mintázatilag azonos, de statisztikai szempontból mégis különbözőnek titulálható terhelésekkel, ami már megfelelő alapként szolgál az eredmények konfidencia-intervallumainak meghatározására.
A modellek következő előnye az általánosságukban (vagy jobban mondva az általánosíthatóságukban) rejlik. Ha egy eseménynaplót használunk a terhelés alapjául, akkor fennáll a veszélye annak, hogy a rendszert csak erre a konkrét viselkedésmintára értékeljük ki.
Tekintsük példának azt az esetet, amikor egy rendszer éjszakai és nappali terhelése jelentősen eltér, mint például egy telefonközpont vagy egy ország elektromos hálózata esetén. Az éjszakai és nappali időszak közötti váltás többféleképpen lejátszódhat. Munkanapokon a reggeli órákban az elektromos hálózat terhelése hirtelen megugrik a közel egy időben felkapcsolt lámpák és egyéb háztartási gépek bekapcsolása miatt. Munkaszüneti napokon ez a terhelésváltozás sokkal egyenletesebb, nem figyelhető meg ennyire hirtelen ugrás. Egy terhelésmodellel lehetőségünk van több változásmintát is kiértékelni, elkerülve ezáltal annak a veszélyét, hogy túl specifikus viselkedéshez igazítjuk a rendszert.
A specifikus viselkedés egyik speciális esete a megfigyelt eseményekben fellelhető *zaj*, ami a méréstechnika tudományában gyakran előfordul, és nem szabad elhanyagolni. Zajként tekinthetjük például azt a jelenséget, amikor egy ügyintéző először rosszul tölt ki valamilyen űrlapot, emiatt újra ki kell töltenie. Általánosságban véve ez az először sikertelen, majd újra végrehajtott kérések kategóriája. Alkalmazástól függően ez a kérés lehet zaj a rögzített eseménysorban, vagy lehet egy gyakran előforduló eset, amikoris az ismételt végrehajtás hatása nem elhanyagolható. Míg az eseménynapló ezt az információt nem tartalmazza, addig egy terhelésmodellbe beépíthető az ismételt végrehajtás mint viselkedési minta.
A terhelésmodellek nem elhanyagolható előnye az eseménynaplókkal szemben a *tömörségük*. Míg egy terhelésmodell általában néhány paraméter segítségével leírható,

addig egy rögzített eseménynapló több millió kérést is tartalmazhat. A mintavételezett rendszer méretétől függően már az eseménysor eltárolása sem könnyű feladat, a visszajátszásról nem is beszélve. Továbbá a terhelésmodellek letisztultsága és szabályossága révén előfordulhat, hogy sokkal rövidebb szimulációk segítségével is értékelhető eredményeket kapunk (Eeckhout–De Bosschere 2001; Eeckhout–De Bosschere–Neefs 2000).
A terhelésmodellek újabb előnyét jelentik a *módosíthatóságuk*, akár a kiértékelés közben is. Terhelésmodell használata esetén lehetőségünk nyílik visszacsatolást építeni a szimulációba, amely során a további terheléskor figyelembe vesszük az eddigi kérések állapotát (Shmueli–Feitelson 2006).
Ennek tipikus példája, amikor az egyes kérések nem teljesen függetlenek egymástól, hanem például az egyik kérés végrehajtási módja (vagy az, hogy egyáltalán végrehajtjuk-e) függ egy előző kérés eredményétől (például egy felhasználó hiánytalanul töltötte ki az űrlapot). Az ilyen jellegű visszacsatolások esetén úgynevezett „zárt hurkú” (*closed loop*) kiértékelésről beszélünk.
Eseménynapló-alapú visszajátszás esetén ilyen függőségeket nem tartalmaz a log, csak események/kérések sorozatát, amiket egymástól függetlenül (a sorrendiséget leszámítva) hajtunk végre a rendszerrel. Ilyenkor „nyílt hurkú” (open loop) mérésről beszélünk.

3.2.3. A terhelésmodellezés egyéb felhasználási területei

Habár a terhelésmodellezést leggyakrabban egy rendszer teljesítményfelmérésekor alkalmazzák, számos más felhasználási esete is létezik. Ezek egy része továbbra is a teljesítményhez kötődik valamilyen formában, de ettől független rendszeraspektusok elemzésére és megfigyelésére is használhatunk terhelésmodelleket. Az alábbi felhasználási területek a legmeghatározóbbak a teljesítményfelmérésen kívül:

- a rendszer megfelelő kapacitásra tervezése,
- jövőbeli teljesítmény prediktálása,
- automatikus, futásidejű újrakonfiguráció
- és anomália detektálása.

A teljesítményelemzés egyfajta kifordított verziójaként fogható fel egy rendszer *kapacitásra tervezése*. Amíg a hagyományos teljesítmény-értékelés esetén azt vizsgáljuk, hogy egy adott terhelés hatására a rendszer az aktuális konfigurációjában milyen teljesítménykarakterisztikákat mutat, addig kapacitástervezéskor a probléma fordított. Szeretnénk egy olyan rendszerkonfigurációt találni, amellyel „megfelelő” teljesítménnyel vagy minőségben ki tudjuk szolgálni a rendszert érő terhelést (Menasce et al. 2004). A megfelelőség fogalmát a rendszer specifikálásakor pontosan definiálják, többnyire elvárt szolgáltatási szint (service-level agreement, SLA) vagy szolgáltatási minőségi (quality of service, QoS) kényszerek formájában.
A kapacitástervezés különösen fontos szerepet játszik a rendelkezésre álló erőforrások megfelelő felhasználásában/kiosztásában. Egyrészt ügyelni kell arra, hogy a rendszer megfelelően ki tudja szolgálni az „átlag” terhelést, amihez kellő mennyiségű erőforrást kell biztosítani. Másrészt viszont túl sok erőforrás allokálása azt eredményezheti, hogy nagy részük kihasználatlanul üzemel, tehát felesleges kiadásnak számít. A megfelelő mennyiségű erőforrás kiosztása különösen fontos alappillére a felhőalapú számítási infrastruktúráknak (Ganapathi et al. 2010).
Egy rendszer szükséges kapacitásának méretét jelentősen befolyásolják a terhelésének bizonyos karakterisztikái, mint például a terhelés intenzitás. A kapacitástervezés során

felmerülő egyik kihívás, hogy a terhelés intenzitása és az adott terhelés kiszolgálásához szükséges kapacitás között általában bonyolult (nem egyszerűen lineáris) kapcsolat áll fenn.
Tovább nehezíti a dolgot, hogy a terhelés intenzitása időről időre jelentősen ingadozhat (Casale et al. 2012; Casale–Mi–Smirni 2010). A terhelés „börsztösségének" (burstiness) eredménye, hogy a valós megfigyelt terhelés általában el fog térni az adott időre számolt átlagos terheléstől. Ennek megfelelően, ha csak az átlagterhelésre méretezzük a rendszer kapacitását, akkor a nagy intenzitású terhelések időszakában a rendszer nem fog elég erőforrással rendelkezni ahhoz, hogy megfelelő minőségű szolgáltatást nyújtson.
A terhelésmodellezést szintén gyakran alkalmazzák éles rendszerek jövőbeli *teljesítményének prediktálásához*. Ha meg tudjuk jósolni egy rendszer teljesítményét a közeli jövőben, akkor lehetőségünk nyílik az erőforrások hatékony kiosztására, illetve az aktuális kiosztás módosítására annak érdekében, hogy fenntartsuk a megfelelő szolgáltatásminőséget. Ezt a megközelítést futásidejű szabályozásnak vagy újrakonfigurálásnak is hívják.
A legegyszerűbb esetben csak megfigyeljük a rendszer aktuális terhelését (például a kérések érkezési rátáját, vagy a kéréstípusok eloszlását), és ennek megfelelően bizonyos módosításokat végzünk a rendszeren, hogy az új terhelésmintázatot is (ha esetleg változás állt be az eddigiekhez képest) megfelelően ki tudja szolgálni. A módosítások több mindent is jelenthetnek, kezdve például a rendszer bizonyos paramétereinek hangolásától (Feitelson–Naaman 1999; Zhang et al. 2005) egészen akár a rendszer fontos részét alkotó algoritmusok közötti váltásig (Talby–Feitelson 2005).
Ennek a szabályozási folyamatnak a kifinomult verziója az *automatikus futásidejű újrakonfiguráció*. Ha egy rendszer terhelése hirtelen megnövekszik, akkor nem minden esetben engedhető meg, hogy megvárjuk az emberi beavatkozáson alapuló szabályozást. Természetesen az is előfordulhat, hogy egyszerűen nem szeretnénk emberekre bízni ezt a feladatot, mert egyrészt a módosítások könnyen automatizálhatók, másrészt nem igényelnek emberi jóváhagyást (például a kiosztott erőforrás-kapacitást csak 5%-kal kell megnövelni, ami még egy előre rögzített kritikus határon belül van).
Az automatikus újrakonfigurációhoz már nem elég csak az aktuális terhelés megfigyelése. Míg manuális esetben a beavatkozó rendszermérnök tisztában van az eszközölt változások hatásaival, addig az automatikus esetben erre az emberi tudásra/tapasztalatra nem számíthatunk. Éppen ezért az automatizált beavatkozó mechanizmusnak valahonnan tudnia kell, hogy különböző változások milyen hatással lesznek az újrakonfigurált rendszer teljesítményére, figyelembe véve az aktuális terhelést.
Ez a plusztudás a rendszer teljesítménymodelljének formájában kerül integrálásra a szabályozó mechanizmusba. Ezáltal a terhelés megváltozásakor lehetőség nyílik egy automatikus analízisre, amely során az aktuális terhelés karakterisztikáit kombináljuk a rendszer teljesítménymodelljével. Az analízis során lehetőség van kiértékelni, hogy a teljesítménymodell bizonyos paramétereinek megváltoztatása (amelyek jellemzően megfelelnek a valós rendszer egy-egy paraméterének) milyen hatást fog gyakorolni a teljesítményre (Zhang et al. 2003).
Fontos figyelembe venni, hogy egy optimális megoldást találó, kimerítő analízis időigényes folyamat lehet (feltéve, hogy egyáltalán lehetséges), így általában közelítő, szuboptimális megoldásokkal is megelégszünk. A teljesítményelemzés során azonosított érzékeny változók jó heurisztikát adnak arra, hogy mely paramétereket érdemes egyáltalán változtatni ahhoz, hogy jobb teljesítményt nyújtson a rendszer.

A terhelésmodellezésnek talán egy kevésbé nyilvánvaló felhasználási módjának tekinthetjük az *anomáliadetektálást*. Ebben az esetben a terhelésmodell célja, hogy leírja a rendszer *normál* működési tartományában jellemző terhelést. Ha ez az információ/modell rendelkezésünkre áll, akkor az üzembe helyezett rendszer megfigyelésével lehetőségünk adódik jelezni, ha az aktuálisan észlelt terhelés eltér a referenciaterheléstől. Ez az eltérés jellemzően két féle hibára engedhet következtetni:

- A rendszert túlterheléses támadás érte (Burgess et al. 2002; Chandola–Banerjee–Kumar 2009; Stolfo et al. 2006; Vieira et al. 2010), ami veszélyezteti a megfelelő minőségű szolgáltatás nyújtását a „hagyományos" felhasználók felé. Az elosztott szolgáltatásmegtagadással járó támadások (distributed denial-of-service, DDoS) során például a rendszert több (akár több száz vagy ezer) forrásból intenzív terhelés éri, amiket (megfelelő védelem hiányában) megpróbál kiszolgálni, felhasználva eközben az összes rendelkezésre álló erőforrást. Ennek tipikus következménye, hogy a nem rosszindulatú forrásból érkező kérések kiszolgálását már nem tudja elvégezni a rendszer, de akár teljesen össze is omolhat (ami például egy segélykérő szolgáltatásnál katasztrofális következményekkel is járhat).
- A rendszert, vagy annak egy komponensét egy újabb verzióra frissítjük, ami azonban hibát/hibákat tartalmaz. A hiba hatására a rendszer abnormális viselkedést produkálhat (változatlan terhelés mellett), ami eltér az eddig megfigyelt és a modellezés során referenciaként rögzített viselkedéstől (Barford et al. 2006; Cherkasova et al. 2008). Hasonló effektus figyelhető meg rosszul (újra)konfigurált rendszerek esetén.

A tisztán terhelésmodell-alapú anomália detektálás alkalmazásánál körültekintően kell eljárnunk. Egy megnövekedett intenzitású terhelés nem feltétlenül jelenti azt, hogy a rendszert támadás érte. Ha egy olyan online ügyintéző szolgáltatást tekintünk, amely egy egész ország számára közös határidőt szabott egy feladat elvégzésére (például adóbevallás), akkor az emberek nagy valószínűséggel a határidő előtti rövid időintervallumban (napok, vagy akár órák) fogják közel egyszerre végrehajtani ezt a feladatot. De hasonló jelenség figyelhető meg webboltok esetén a karácsonyi időszakban, amikor hirtelen megnő az online vásárlások (tehát kérések) száma a szolgáltatással szemben. Ilyen esetekben nem elég csupán a terhelés intenzitásának a változását detektálni, hanem a rendszer viselkedésének több aspektusát is szorosan figyelni kell. Ha több ilyen megfigyelő mechanizmus „riaszt be" egyszerre (Stolfo et al. 2006), akkor viszont már nagy eséllyel tényleges támadás érte a rendszert, és nem csak egy szezonális viselkedésváltozásról van szó.

Az elmúlt években egyre inkább elterjedt a mesterséges intelligencia anomáliadetektálási célokra történő felhasználása. A megközelítés alapgondolata, hogy a rendszer normál viselkedéstartományát „megtaníttatjuk" a megfigyelő komponenssel, ami azonnal jelez, ha olyan viselkedést észlel, ami a megtanult referenciaviselkedéstől eltér.

A módszer egyik kiemelkedő előnye, hogy olyan támadási esetek észlelése is lehetséges, amelyekkel ezelőtt még nem találkoztunk. Ez azért lehetséges, mert a hagyományos védelmi mechanizmusokkal ellentétben itt nem a támadási módszereket kell felismerni (amelyek többnyire előbb jelennek meg, mint az ellenük kidolgozott védelmek), hanem a normál működést, amit behatóan ismerünk.

3.2.4. A terhelésmodellek típusai

Az előző fejezetekben bemutattuk a terhelések és terhelésmodellek alkalmazásának előnyeit és főbb felhasználási területeit. Rávilágítottunk, hogy a modellezés alapjának szerves része a megfigyelt rendszerről gyűjtött adatok elemzése. Az adatelemzés végterméke egy tipikusan statisztikai jellegű modell (Law–Kelton 1991), ugyanakkor ez nem azt jelenti, hogy a modellezés folyamata kizárólag a megfigyelések statisztikai kiértékeléséből állna.
Habár a statisztika kiterjedt eszköztára lehetővé teszi különböző adathalmazokban rejlő mélyebb összefüggések elemzését és feltárását, azonban általában lehetetlen pusztán statisztikai módszerekkel minden, a terhelésmodell pontos karakterizálásához szükséges információt feltárni.
A különböző statisztikai szoftverek használata bevett módja annak, hogy akár nagyobb adathalmazokról is gyorsan áttekintést kapjon az adatelemző (érthetők ez alatt például egy megfigyelt változó nevezetes értékei, mint az átlag vagy medián). Az adatelemzés ezen aspektusában a számítógépek minden kétséget kizáróan jobban teljesítenek, mint az emberek, akiknek például egy több százezer megfigyelésből álló adatsor átlagának kiszámolása jelentős időbe telne.
Az emberi agy azonban egy területen még mindig felülmúlja a legjobb szoftvereket is, és ez a terület nem más, mint a vizuális minták felismerése (*pattern recognition*). Az adat „mintázatának” fontosságát legjobban az Anscombe négyesének nevezett adatsorok szemléltetik. Az adatsorok jellegzetessége, hogy az alap leíró statisztikai jellemzőik (mint például az átlag és a szórás), de még a lineáris regressziós egyeneseik is megegyeznek.
Azonban ha koordináta-rendszerben (egész pontosan pontfelhődiagramon) ábrázoljuk az adatokat, akkor azonnal látszik már ránézésre is, hogy fundamentálisan különböző adatsorokról beszélünk, dacára annak, hogy a leíró statisztikáik megegyeznek. Ez szolgál alapjául annak, hogy az adatelemzés folyamatába bevonjuk a vizuális elemzési módszerek használatát is (jellemzően különböző diagramok formájában). Ennek megfelelően az adatelemzés ezen lépését vizuális felderítő adatanalízisnek is szokták hívni (Anscombe 1973; Tufte 2001; Tufte–Goeler–Benson 1990; Tufte et al. 1998; Tukey 1977; Borgelt–Höppner–Klawonn 2010), amely erősen támaszkodik az elemző szakember tudására és tapasztalatára (Chatfield 2002).
Az alkalmazott statisztikai és vizuális módszerek végterméke egy terhelésmodell lesz. Azonban attól függően, hogy a rendszer terhelését adó környezet mely részét ragadjuk meg a modellben, kétféle terhelésmodellezés-megközelítésről beszélhetünk: *leíró* vagy *generatív* modellekről.
A leíró modellek esetében a megfigyelt terhelést modellezzük, jellemzően úgy, hogy a terhelés statisztikai tulajdonságait és a kérések attribútumait ragadjuk meg a modellel. Ilyen attribútumok lehetnek például: a kérés által igényelt számítási kapacitás (rövid vagy hosszú ideig futó feladat); a kérés háttértár-elérési mintája (inkább CPU-igényes, vagy gyakran fordul például a merevlemezhez szükséges adatért); illetve a kérés hálózati kommunikációs jellegzetességei (Kotsis 1997; Sreenivasan–Kleinman 1974).
A származtatott statisztikai jellemzők alapján végül valószínűségi eloszlással közelítjük a terhelés jellemzőit, amely alapján később mesterséges/szintetikus terhelést tudunk generálni. A valószínűségi modell továbbá lehetővé teszi, hogy a rendszert (a terhelésével együtt) különböző matematikai elemzéseknek vessük alá.
A generatív terhelésmodellezés során a terhelést úgymond közvetett módon modellezük. Nem magukról a kérésekről készítünk statisztikai modellt, hanem arról a

folyamatról, amelynek eredménye a terhelés. Tekintsünk egy webboltot példaként. A rendszerbe érkező kérések többnyire keresésjellegű műveleteket foglalnak magukba. A felhasználók egy ideig keresik a megfelelő árucikket, majd végül fizetnek. A fizetési funkció használata tehát ritkábban fordul elő, mint például egy adott kategóriába tartozó árucikket ár szerint növekvő sorrendben történő listázása. A leíró modellek esetén nem biztos, hogy helyesen sikerül modelleznünk ezt a mintázatot.
Ehelyett nem magát a rendszerbe érkező terhelést modellezzük, hanem a felhasználók viselkedését. Egy ilyen modellbe könnyen be tudunk illeszteni olyan tudást, ami azt írja le, hogy egy felhasználó egy árucikk keresése után 90% valószínűséggel további árucikkeket fog keresni, 5% valószínűséggel megszakítja a vásárlást (mert meggondolta magát), és csak 5% valószínűséggel fogja igénybe venni a fizetés funkciót. Ha helyesen készítünk el egy ilyen viselkedésmodellt, akkor annak „lejátszásával" automatikusan olyan szintetikus terhelés generálódik, amely jól közelíti a megfigyelt, eredeti terhelést (innen ered a módszer generatív elnevezése).
A generatív terhelésmodellezés egyik előnye, hogy viszonylag egyszerűen képesek vagyunk módosítani, tehát lényegében konfigurálható tenni a rendszerbe érkező terhelést. Megtehetjük például, hogy a fenti példában említett arányokat úgy módosítjuk, hogy a további keresés valószínűségét 50%-ra csökkentjük, míg a vásárlás valószínűségét 45%-ra növeljük. Ezzel szimulálhatjuk például azt, ha egy népszerű termék éppen akciós, és a vásárlók a nézelődés helyett kimondottan a leárazott termék megvásárlása céljából használják az oldalt (abban az esetben például, ha nincs idő egyéb termékeket nézni, mert a leárazott árucikk csak korlátozott számban érhető el).

3.3. Terhelésadatok gyűjtése és kezelése

A reprezentatív terhelésmodellek készítésének elengedhetetlen előfeltétele, hogy valós, vagy nagyon pontosan megbecsült és felmért terhelések megfigyelése alapján készüljenek. A modellezéshez szükséges terhelésadatok többféle módon állhatnak a rendelkezésünkre.
Egyrészt nem kizárt, hogy a terhelésadatok már léteznek, mert már egy létező rendszerhez szeretnénk terhelésmodellt készíteni, és a rendszer részletesen naplózta az általa kiszolgált kérések megfelelő tulajdonságait. Ha nem ez a helyzet, akkor nem kerülhető el az adatgyűjtés végzése.
Erre leginkább passzív vagy aktív instrumentációs (felműszerezési) módszereket használnak. Bármilyen forrásból is álljon rendelkezésre az adat, mindenképpen fontos lépése a modellezés előkészítésének, hogy megtisztítsuk az adatot a zavaró, nem reprezentatív megfigyelésektől, amelyek adott esetben jelentősen torzíthatják a modellt.
A következő alfejezetek az adatgyűjtés és adatfeldolgozás módszereit tekintik át ismertető jelleggel.

3.3.1. Létező adatok felhasználása

A könyvelési és tevékenység naplók a legjellemzőbb példái annak, hogy a terhelésről már rendelkezésünkre állnak adatok (Sreenivasan–Kleinman 1974). A rendszerek többsége nyilván tart egy naplót/logot az elvégzett feladatok listájáról és azok bizonyos tulajdonságairól, ha másért nem, akkor hibaelhárítási célokra (vagy a kötelezően előírt auditok miatt).
Az adatok megléte önmagában még nem garantálja azok felhasználhatóságát. Attól függően, hogy mi a terhelésmodellezés célja, más és más aspektusait kell megfigyelni

a terhelésnek. Ha ezekhez az aspektusokhoz nem áll rendelkezésre kellően részletes tevékenységnapló, akkor nincs más választás, mint különböző instrumentációs módszerek segítségével elvégezni a szükséges részletességű adatgyűjtést.
A webes kiszolgálók remek példái az olyan rendszereknek, amelyek már üzembe helyezésük óta nagy valószínűséggel naplózzák a feldolgozott kéréseket. Ezt jól példázza az is, hogy a hagyományos HTTP internetes forgalmak naplózására egy közös logformátum alakult ki, amely emberek által is olvasható módon a kérések bizonyos tulajdonságait egy szöveges fájl külön soraiba menti.
A naplózott tulajdonságok között van jellemzően a kérés küldőjének IP-címe, amely azonosítja a klienst és közelítőleges tartózkodási helyét, persze nem minden esetben (Maier–Schneider–Feldmann 2011; Rosenstein 2000). A kérések naplózásának időpontja (időbélyege) is olyan információ, amit szinte kivétel nélkül minden rendszer feljegyez (és szerves része szinte minden logformátumnak). Ha a rendszer által használt webkiszolgáló keretrendszerek beépített naplózási mechanizmusa nem szolgál elég részletes információval, akkor erről a rendszer fejlesztőinek kell gondoskodni.

3.3.2. Adatgyűjtés instrumentációval

Ha nem állnak rendelkezésre előzetesen naplózott adatok a terhelésmodellezéshez, akkor el kell kezdeni az adatgyűjtést az éles rendszer megfigyelésével. Ennek módja jellemzően az instrumentáció (felműszerezés), amely során a rendszert olyan képességekkel vértezzük fel, hogy képes legyen a tevékenységeinek rögzítésére. Az instrumentációs két lehetséges megközelítési módja a passzív és aktív instrumentáció. A választott módszertől függetlenül lényeges figyelmet kell fordítanunk arra, hogy se a rendszer logikai viselkedését ne befolyásolja a megfigyelés ténye, se a teljesítményét ne rontsa, hiszen utóbbi esetben a mért adatok nem lesznek reprezentatívak.
*Passzív felműszerezés*ről akkor beszélünk, amikor magát a rendszert nem módosítjuk, hanem külső komponensek segítségével/csatolásával figyeljük meg a rendszerben zajló tevékenységeket azok megzavarása nélkül. Ennek jellegzetes példája a hálózati kommunikáció megfigyelése, amikor is a kommunikációs csatorna „lehallgatásával" szerzünk tudomást a rendszer által elvégzett tevékenységekről (Leland et al. 1995; Tsai–Fang–Chen 1990; Gribble–Brewer 1997; Finamore et al. 2011; García-Dorado et al. 2012). Természetesen a kommunikáció megfigyelésének megvannak a korlátjai.
Egyrészt az alkalmazott kommunikációs protokolloktól függően nem biztos, hogy megfelelő mértékben meg tudjuk különböztetni az egyes tevékenységtípusokat. Ha a kapcsolat megfelelően titkosított a felhasználó és a rendszer között, akkor az olyan alkalmazásszintű információk, mint például az igénybe vett szolgáltatás típusa, rejtve maradnak a kommunikációt monitorozó külső komponens elől (amely csak annyit lát, hogy a rendszerünk hálózati címére érkezett egy kérés).
Másrészt a hálózati csatorna megfigyelése azt jelenti, hogy kizárólag a rendszerünk (vagy komponenseinek) határán zajló kommunikációt látjuk. Ez közel sem ad elég információt arról, hogy a rendszerünkön belül mi történik, ami függvénye lehet például a kérés típusának, illetve paramétereinek (az első 10 vagy esetleg 100 találatot szeretné lekérni a felhasználó).
Ha a passzív instrumentáció segítségével nem tudunk elég részletes adatokat gyűjteni a terhelésmodellezéshez, akkor a rendszerünket kell módosítani úgy, hogy saját tevékenységeit a megfelelő részletességgel rögzíteni tudja. Ebben az esetben már *aktív instrumentációról* beszélünk, ami a passzívhoz képest egy intruzív módszer, hiszen a rendszer módosításával jár.

Egy rendszer aktív felműszerezése életciklusának több fázisában is megtörténhet. A nyilvánvalóan teljesítménykritikus rendszerek esetén (mint például általános adatbázismenedzsment rendszerek) már a tervezési fázis során integrálhatók a részletes naplózási mechanizmusok.
Ha a terhelésmodellezésre (és ezáltal az adatgyűjtésre) akkor van szükség, amikor már üzembe helyezték a rendszert, akkor általában a rendszer újraindítása és az instrumentált verzióra frissítése szükséges (Nieuwejaar et al. 1996). Ha a rendszer bizonyos komponensei több példányban is üzemelnek (mert például 10 darab webkiszolgáló szerverrel tudunk csak megfelelő minőségű szolgáltatást nyújtani), akkor a frissítés megoldható komponenspéldányonként. Ez lehetővé teszi a teljes rendszerleállás elkerülését, sőt, megfelelő tartalékkapacitással rendelkező rendszerek esetén a szolgáltatásminőség romlása is elkerülhető. Bizonyos esetekben az instrumentáció az üzemelőrendszeren dinamikusan is elvégezhető (Buck – Hollingsworth 2000; Hollingsworth – Miller – Cargille 1994) anélkül, hogy bármelyik komponens frissítésére vagy újraindítására lenne szükség.
Saját fejlesztésű és felhasználású komponenseknél/rendszereknél könnyen megoldható a rendszer utólagos felműszerezése. A helyzet azonban bonyolultabb az olyan általános célú komponenseknél, amelyeket mások által fejlesztett rendszerek használnak fel almodulként. Ebben az esetben az utólagos instrumentáció után fel kell hívni a modult használók figyelmét, hogy frissítsék a komponenst (feltéve, ha szükségük van adatgyűjtésre).
A kiterjedt méretű hálózatok elterjedésével (például közösségi oldalak) egyre inkább jellemző az általános célú komponensek/keretrendszerek fejlesztése közben azok teljesítményének tudatos figyelembevétele. Ennek megfelelően a legtöbb komponens beépített mechanizmusokat tartalmaz a teljesítményének részletes megfigyelésére és naplózására. Ezt az irányt remekül szemlélteti, hogy olyan bonyolult „komponensek” is elérhetővé tesznek ilyen mechanizmusokat, mint például processzorok (Chen et al. 1996; Sprunt 2002a, 2002b) és operációs rendszerek, amelyek esetén a megfigyelhetőséget úgynevezett teljesítményszámlálók (performance counters) biztosítják. A teljesítményszámlálók lehetővé teszik, hogy tetszőleges monitorozó alkalmazás (igény esetén, „feliratkozás” után) értesüljön a komponens fontos belső eseményeiről (mint például egy kérés kiszolgálásának a végéről, vagy a percenként kiszolgált kérések átlagos számáról).

3.3.3. Az adatgyűjtés és adattárolás kihívásai és bevett gyakorlatai

Adatgyűjtés során a rendszer egy esemény hatására feljegyzi annak megtörténtét és kívánt attribútumait. Ez a feljegyzés első lépésként egy memóriabeli műveletet jelent, amely rövid időre lefoglalja a rendszer erőforrásait (egész pontosan a processzort). Második lépésként ezeket a feljegyzéseket ki kell írni egy stabil tárra, hogy a későbbiekben feldolgozzuk őket terhelés- vagy teljesítménymodellezés céljából. Ez a lépés legalább egy nagyságrenddel több ideig tart, mint a memóriába történő írás.
A helyzet javítható, ha naplózáshoz (a manapság egyre inkább elterjedt) SSD-ket (solid state drive) használunk, amelyek írási sebessége nagyobb, mint a hagyományos merevlemezek esetében. Azonban az SSD-k teljesítménye még így is elmarad a memóriába írásétól, nem beszélve azokról az esetekről, amikor a feljegyzéseket egy (akár hálózaton keresztül elért) adatbázisba írjuk a későbbi feldolgozás megkönnyítése érdekében, tovább növelve a naplózás erőforrás- és időigényét.
Hosszabb futásidejű tevékenységek naplózása során ez a plusz erőforrás- és időráfordítás nem torzítja sem a rendszer viselkedését, sem a gyűjtött adatok

reprezentativitását. Ha azonban nagy bekövetkezési rátájú eseményeket szeretnénk naplózni, akkor az adatgyűjtés ideje összemérhető lehet a ténylegesen elvégzett munkáéval.
Ez nyilvánvalóan nemkívánatos jelenség, hiszen ilyenkor a rendszer viselkedése (például végrehajtási idők terén) nem egyezik meg azzal az esettel, amikor már nem végzünk adatgyűjtést. Tehát tekinthetjük úgy, mintha egy „A" rendszer terhelésmodellje alapján szeretnénk kiértékelni egy „B" rendszert, ami téves következtetésekhez fog vezetni.
Az adatgyűjtés során potenciálisan okozott intruzivitás minimalizálására többféle módszer létezik (teljesen megszüntetni nyilvánvalóan nem lehet).
Az első lehetőség adatpufferek használata. A megközelítés lényege, hogy nem minden esemény rögzítésekor történik meg a háttértárra való kiírás. Az eseményeket előfordulásukkor csak a gyors műveletvégzést biztosító memóriába mentjük, majd bizonyos mennyiségű összegyűlt esemény után egyszerre írjuk őket a háttértárra, ugyanis nagyobb adatmennyiség egy menetben történő kiírása hatékonyabb, mint sokszor kisebb mennyiségű adatot menteni.
Ha a puffernek szánt memóriaterület megtelik, akkor a puffer tartalma kiírásra kerül a háttértárra, és a terület felszabadul újabb események rögzítése céljából. Azonban ügyelni kell arra, hogy a puffer ürítése közben (ami valamennyi időt igénybe vesz) keletkező eseményeket is rögzíteni tudjuk.
Erre megoldás a duplapufferelés, amikor két puffert használunk. Az adatgyűjtés elején az egyik puffert kezdjük el használni, egészen addig, amíg az megtelik. Ekkor a másik puffert kezdjük el használni. Még mielőtt a második puffer megtelne, az első puffer tartalmát kiírjuk a háttértárra, ezzel kiürítve azt. Tehát a pufferürítés ideje alatt mindig rendelkezésre fog állni egy tartalék puffer, így megszakítás nélkül tudjuk tovább rögzíteni az új eseményeket. Az események gyakoriságától, a pufferek méretétől és a háttértárra írás sebességétől függően a cserepufferek számát tetszőlegesen növelhetjük annak érdekében, hogy mindig rendelkezésre álljon egy még be nem telt puffer.
Az intruzivitás problémájára szintén megoldás lehet, ha nem(csak) minimalizáljuk, hanem kompenzáljuk az adatgyűjtés rendszerre gyakorolt hatását. Ha modellezni tudjuk az instrumentáció hatását, akkor lehetőségünk nyílik, hogy a mérési eredményekből ezt „levonjuk" (Malony–Reed–Wijshoff 1992).
Az adatgyűjtés rendszerre gyakorolt zavaró hatásán kívül egy másik felmerülő probléma a gyűjtött adat mennyisége. Az eddigi megközelítések azt feltételezték, hogy egy esemény bekövetkeztekor rögzítjük annak megtörténtét és szükséges tulajdonságait. Azonban ha a rendszerben több komponens több eseményét figyeljük meg, akkor a rendszer terhelésétől függően hamar azt vesszük észre, hogy nagyon rövid idő alatt óriási (több GB vagy akár TB) méretű eseménynaplót generálunk. A tárolási problémán kívül a napló feldolgozása is kihívást jelent, illetve az események intenzitásának növekedésével az intruzivitás is jelentős lesz.
A gyakori események miatti nagy adatmennyiség problémájának megoldására egy olyan adatgyűjtési módszert szoktak alkalmazni, amely úgymond független az események gyakoriságától. Ez az adatgyűjtési (illetve megfigyelési) módszer a *mintavételezés* (sampling).
Az eseményalapú adatgyűjtéstől eltérően a mintavételezés során a megfigyelhető adatokat csak bizonyos időközönként (és helyen) mintavételezzük. Például ha egy rendszer terhelését szeretnénk megfigyelni, akkor az eseményalapú megközelítés során minden kérés idejét és szükséges tulajdonságát rögzítjük. Internetes gerinchálózatok esetén (például amik kontinenseket/országokat kötnek össze) ez óriási adatmennyiséget generálna, amelyek feleslegesen részletesek. Ehelyett alkalmazhatunk mintavételezést, amely során például naponta több alkalommal

15 percre megfigyeljük a forgalmat, és a teljes nap forgalma helyett ezekből a szegmensekből próbálunk modellt építeni a napi terhelésre.
A megközelítés hátránya, hogy nincs bevált módszer arra, hogy mikor is érdemes mintavételezni. Ha nincs előzetes ismeretünk a megfigyelendő rendszerről, akkor a véletlenszerű mintavételezéssel érdemes kezdeni, és ellenőrizni, hogy mennyire reprezentatív a véletlenszerűen gyűjtött adat. Ha azonban tudjuk, hogy a terhelés valamilyen (például ismétlődő) mintázatot tartalmaz, akkor ezt figyelembe kell venni a „véletlen" mintavételezés során, hogy ne torzítsuk el ezt a szabályosságát a terhelésnek modellezés közben.

3.3.4. Gyűjtött adatok reprezentativitása

A terhelésmodellezés alapfeltevése, hogy az adat, ami alapján elkészítjük a modellt, jól reprezentálja a rendszer terhelését. Ha ez nem így van, akkor a modell alapján félrevezető vagy egyenesen rossz következtetéseket fogunk levonni a további elemzések során. A gyűjtött adatok reprezentativitása több okból is sérülhet. A fejezet ezeket az eseteket gyűjti össze és elemzi.
Gyakori probléma, hogy a gyűjtött adatok hiányosak. Egyszerűbb esetben ez könnyen észrevehető, például amikor egy feladat naplózásánál a végrehajtási idő nincs kitöltve. Ennek több oka is lehet, például az, hogy a feladatot nem sikerült elvégezni, ezért a végrehajtási idő mezője nem lett kitöltve. Ugyanakkor az is előfordulhat, hogy az érték mentése közben történt hiba, és a feladat tulajdonságai csak részlegesen kerültek mentésre.
Vannak azonban esetek, amikor a gyűjtött adatok alapján azt gondolnánk (ránézésre), hogy az adathalmaz teljes. Habár a megfigyelt eseményeket valóban megfelelően rögzítettük, a hiányos adatok másik esete az, hogy bizonyos eseményeket eleve nem figyeltünk meg. Ez előfordulhat például akkor, ha a felhasználó és a rendszer között található egyéb mechanizmus is közbe ékelve, például egy gyorsítótár (Rosenstein 2000; Fonseca et al. 2002; Fonseca–Almeida–Crovella 2005). Ebben az esetben bizonyos kérések már a rendszer határán kívül kiszolgálásra kerülnek, emiatt az alkalmazott adatgyűjtési módszer egyszerűen nincs is tisztában a megtörténtükkel.
A fenti egyszerű példa remekül illusztrálja, hogy egy rendszer megfigyelése annak alapos ismeretét igényli, legalább architekturális szinten. A gyorsítótár hatásának figyelembevételéhez szükségünk van kliensoldali, de legalább kliensközeli adatokra is, amikkel kompenzálni tudjuk a rendszerbe érkező csökkentett kérésszámot.
Az adatok teljessége esetén sem garantálható azok használhatósága, mert előfordulhat, hogy a rögzített adat egyszerűen rossz. Ennek egyik tipikus oka az adatrögzítés mechanizmusába bekerült fejlesztői hiba. Ha például egy (előjeles) negatív számot előjel nélküli típusként tárolunk el, akkor a számítógép számábrázolása miatt értelmezhetetlen eredményt kaphatunk (Arlitt–Jin 2000).
Azonban nem minden eset ennyire nyilvánvaló. Egy felmérés (Schneider et al. 2012) során HTTP-kéréseket és -válaszokat figyeltek meg. A kérések a fejlécükben tartalmazzák a küldött adat méretét és típusát is. A felmérés arra világított rá, hogy a fejlécben szereplő adatméret átlagosan 3,2-szerese volt a tényleges adatméretnek. Ezen kívül 35%-uk nem olyan adattípust tartalmazott, mint amit a fejlécben megjelöltek. Az ilyen hibák detektálásához rendelkezni kell a kérés fejlécének alapadatain kívül a teljes kérés tartalmával, ami nem minden esetben kivitelezhető, illetve utólag nem megszerezhető.
Manapság az adatok ilyen jellegű helyességellenőrzéséhez felhasználhatók szabványos adatgyűjtő komponensek, amelyeknek megadhatók egyedi validációs szabályok. Ilyen megoldások használata csökkenti a fejlesztési költséget, hiszen nem

szükséges egy „egyedi” megoldás készítése, elegendő egy létező, bevált és alaposan tesztelt megoldás kiegészítése.
Furcsa kombinációja az adatok reprezentativitási és helyességi problémájának az az eset, amikor az adat valamilyen formában módosításra került az esemény/feladat eredeti állapotához képest. Ezek a módosítások bizonyos esetekben előre tudhatók.
Például ha egy szolgáltatás igénybevétele során egy nagyobb fájlt szeretnénk a hálózaton átküldeni, akkor a hálózat szállítási rétege ezt több kisebb szeletre vágja, és több csomagban küldi át (amelyek a fogadó oldalon az alkalmazás számára átlátszó módon összeállításra kerülnek). Ebben az esetben egy felszínes hálózati szintű elemzés azt a következtetést engedi levonni, hogy több kisebb fájl kerül küldésre. A helyes következtetéshez a teljes képet kell látnunk, tehát fel kell térképezni az esetleges összefüggéseket az egyes csomagok között.
Nem ennyire nyilvánvaló azonban az a gyakori, a módszer során elkövetett hiba, amikor az adatokat (például más mértékegységre) konvertálás után rögzítjük. Tipikus esete ennek a végrehajtási idők rögzítése. Ha a rendszerben nagyon gyors lefutású feladatok is előfordulnak, de a végső adatokat például csak milliszekundum pontossággal, egész számként mentjük, akkor előfordulhat, hogy a konvertálás során a mikroszekundum nagyságrendbe eső feladatok végrehajtási idejét nullaként mentjük el.
A reprezentativitás problémájának különleges eseten az adatok aktualitásának kérdése. Előfordulhat, hogy az adatgyűjtés pillanatában szinte tökéletesen sikerült reprezentálnunk a terhelést (és ez alapján egy használható modellt készítettünk), viszont a terhelési minta rövid időn belül (hetek, hónapok) megváltozik. Ez leggyakrabban új rendszereknél észlelhető, amikor is egyre több felhasználó kezdi el használni a szolgáltatást (Hotovy 1996). Másik példa egy új funkció hozzáadása a rendszerhez, amely az eddigiekhez képest (jelentősen) már karakterisztikájú terhelést von maga után.
Nagy kiterjedésű (például az internethez köthető) terhelési minták esetén is megfigyelhető az ilyen változások hatása, például amikor az email- és FTP-alapú fájlküldés mellett megjelentek a közvetlen webletöltések és a számítógépek közötti közvetlen (peer-to-peer) fájlcserélős alkalmazások (Feldmann et al. 1998; Henderson–Kotz–Abyzov 2008). De a weboldalak felépítési módszereiben történő változások (például reklámok beágyazása, dinamikus tartalmak egyenkénti betöltése) is hatással volt a rendszerek terhelésmintázatára (Hernández-Campos–Jeffay–Smith 2003).
Az alfejezetben bemutatott problémák jól szemléltetik, hogy egy reprezentatív terhelés, adat vagy modell előállítása nem triviális feladat. A reprezentativitás egzakt kiértékelése túlmutat a monográfia keretein, de irányelvként használhatjuk a következő megközelítést: egy reprezentatív terhelésnek/adatnak/modellnek *ugyanarra az eredményre kell vezetnie*, mint annak a forrásnak, amit reprezentálni/helyettesíteni próbál (Ferrari 1984; Uhlig–Mudge 1997; Eeckhout–Vandierendonck–De Bosschere 2003; Kurmas–Keeton–Mackenzie 2003).

3.3.5. Adatszűrés és adattisztítás

Az adatgyűjtés végeztével ritkán kapunk olyan adathalmazt, amely közvetlenül felhasználható például terhelésmodellezéshez (de ez általánosságban is igaz a legtöbb adatelemzési feladatra). Négy főbb oka lehet annak, hogy valamilyen szempontból módosítani kell a gyűjtött adatokat (Feitelson–Tsafrir 2006).
Hibás vagy használhatatlan adatok. Ilyen adatnak minősül például, ha egy esemény attribútumait csak részlegesen sikerült rögzíteni valamilyen okból. Ha egy feladat végrehajtásának csak a kezdési idejét rögzítettük, de a befejezési idejét nem, akkor ez az adat a végrehajtási idők alakulásának modellezésére használhatatlan lesz, így el kell távolítani.

Haszontalan adatok. Ide tartoznak például olyan feladatok rögzített adatai, amelyek végrehajtása sikertelen volt, vagy félbeszakadt (Cirne–Berman 2001; Weinreich–Obendorf–Herder 2006). Az ilyen adatok eltávolításának legfőbb oka az szokott lenni, hogy csak a „jól" elvégzett feladatokat szeretnénk figyelembe venni. Azonban meg kell jegyeznünk, hogy a rendszerekben előfordulnak „rossz" feladatok is, így ezek hatását mérlegelni kell, mielőtt eltávolítanánk őket az adathalmazból.
Az elemzéshez szükségtelen adatok. Előfordulhat, hogy az elemzés és modellezés során nincs szükségünk minden adatra, amelyet gyűjtöttünk. Tipikus példája ennek, amikor egy specifikus feladattípus érkezéseit és végrehajtási idejét szeretnénk elemezni (Barford et al. 2006; Feitelson–Tsafrir 2006; Walter–Wallace 1967; Zilber–Amit–Talby 2005). Ilyenkor a többi feladattípusra vonatkozó adatsorokat elhagyhatjuk, de ettől függetlenül észben tartva azt, hogy az ugyanazon a rendszeren futó feladatok egymásra hatással lehetnek például a közös erőforrásért való versengés miatt.
Az előző két ponthoz hozzá kell tennünk azonban, hogy manapság sokkal fontosabb az adatok megléte, mint a hozzájuk kötődő tárolási költségek. Tipikus példája ennek a hibadiagnosztika, amihez szükség lehet olyan adatokra, amelyek eredetileg nem voltak célpontjai az adatgyűjtésnek, ugyanakkor utólag szükség lehet rájuk.
Ritka, üzemtartományon kívüli („outlier") adatok elhagyása. Az adatgyűjtés során előfordulhatnak olyan események, amelyeket a várt terheléshez/viselkedéshez képest „abnormálisnak" ítélünk meg. Ezek elhagyásával a modellezés során csak a „normális" működési módot elemezzük (Cherkasova et al. 2008; Chiang–Vernon 2001; Feitelson–Tsafrir 2006; Nurmi–Brevik–Wolski 2007).
Az ilyen eseményeknek több oka is lehet, például erőforráshibák miatt megnőtt a rendszer válaszideje, vagy a rendszer támadás alatt volt, ami miatt megnőtt az eldobott kérések száma (és az erőforrások kihasználtsága). Az ilyen esetekben megfigyelt adatok elhagyását az indokolja, hogy a ritka és tranziens események figyelembevétele olyan eredményekhez vezethet az elemzés során, amelyek nem a normál működési állapotra készítik fel a rendszert.
Továbbá, ha például egy túlterheléses támadást tekintünk, akkor a modell alapján többlet-erőforrással tudnánk kompenzálni a hatását, ami költséges. Ehelyett érdemes külön felkészülni az ilyen eshetőségekre ismert támadásdetektáló módszereket használva az erőforrástöbblet helyett (ugyanis egy kiterjedt forrásból érkező támadás potenciálisan bármekkora erőforrás-kapacitást ki tud meríteni).

3.3.6. Megalapozott becslés

Az eddigiek során végig azt feltételeztük, hogy a modellezéshez rendelkezésre áll egy adathalmaz a rendszer viselkedéséről (amit például a rendszer működése során gyűjtöttünk). Azonban nem ez a helyzet olyan rendszerek esetén, amelyek még fejlesztés alatt állnak.
A teljesítményelemzéshez kötődő és szükséges feladatokat, mint például a terhelésmodellezés és rendszerteljesítmény-modellezés, már a fejlesztés korai lépéseiben el kell kezdeni. Ha egy elkészített rendszerről utólag derül ki, hogy nem képes teljesíteni a megszabott teljesítménykényszereket, akkor ennek orvoslása jelentős újratervezéssel és fejlesztési munkával járhat, ami költséges.

Fejlesztés alatt álló vagy teljesen új rendszerek esetén (Dixon–Sherwood 2008) ebből kifolyólag nincs más választásunk, mint megbecsülni például a terhelést, amit a rendszernek a későbbiekben ki kell majd szolgálnia. A becslésnek jellegzetesen két megközelítése van.

- Ha nem egy új technológiai újításról van szó, hanem csak egyszerűen egy újonnan fejlesztett rendszerről beszélünk, akkor nagy valószínűséggel felhasználhatjuk a már létező, működésében és funkcionalitásában hasonló rendszerekről szerzett tapasztalatainkat. Fontos azonban mérlegelni, hogy milyen jellegű információk hordozhatók a két rendszer között.
  Egy új telefonszolgáltató esetén például élhetünk azzal a feltételezéssel, hogy a felhasználók hívási szokása, miszerint nappal többet telefonálnak, nem fog változni. Ellenben nehéz megbecsülni, hogy az új szolgáltatót eleinte vagy hosszabb távon mennyien fogják használni, tehát a terhelés intenzitásáról nincs előzetes információnk.
- Ha egy jellegében új rendszert fejlesztünk, akkor nem támaszkodhatunk más rendszereknél szerzett előzetes tapasztalatokra (Barber 2004). Ilyen esetben leggyakrabban olyan becslésekkel élünk, amelyek a matematikai elemzések során kényelmesen kezelhetők (és ezáltal valószínűleg egyszerűbbé is teszik a számításokat). Az ilyen becslések veszélye, hogy nagy eséllyel nem tükrözik a ténylegesen várható értékeket.

A korábbi tapasztalatokon alapuló műszaki becslések esetén mindenképpen szükséges érzékenységvizsgálatot végeznünk. Ha azt az eredményt kapjuk, hogy egy becsült paraméter értékének megváltozása jelentősen befolyásolja a rendszer valamely fontos tulajdonságát (tehát érzékeny a paraméter értékére), akkor meg kell próbálni növelni a becslésbe fektetett bizalmunkat (például különböző felmérések készítésével a potenciális felhasználói bázis körében). A becslés pontosításával járó többletköltség még mindig kevesebb lesz valószínűleg, mint egy rossz becslés alapján fejlesztett rendszer utólagos javítása.

4. ESETTANULMÁNY

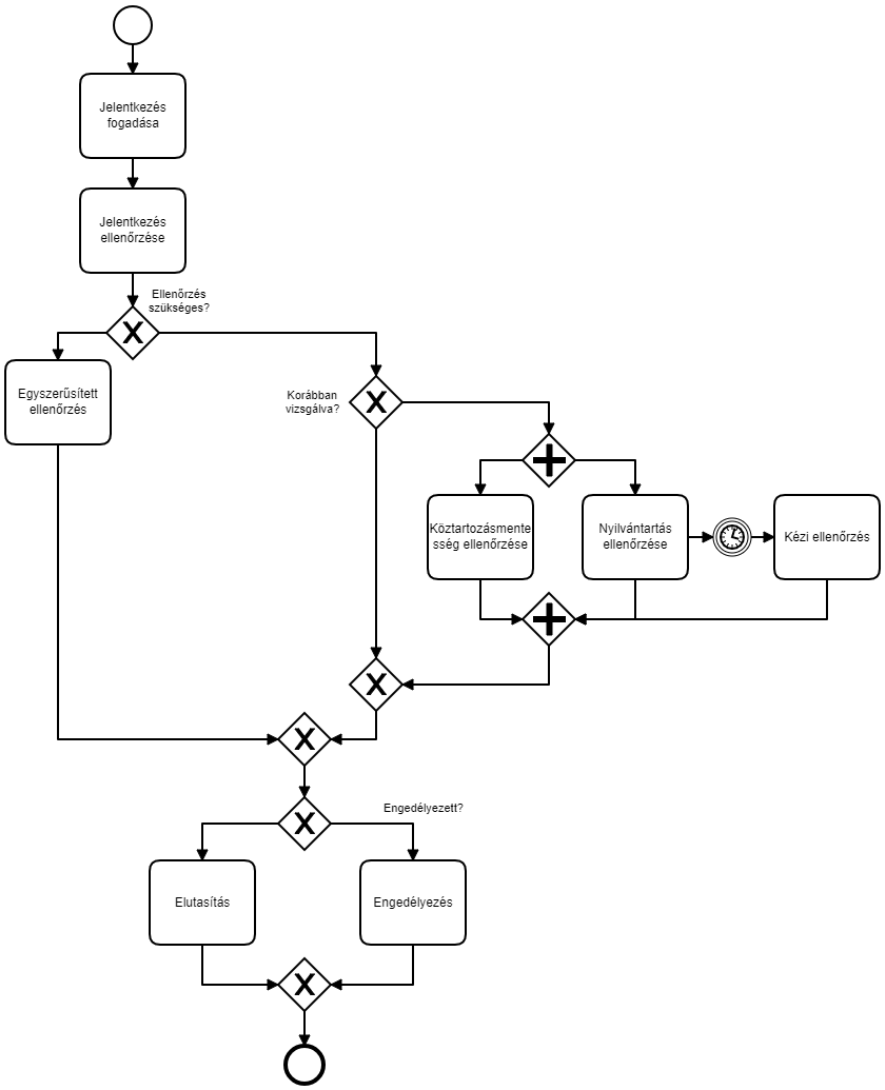
Az alábbiakban bemutatunk egy olyan reprezentatív esettanulmányt, amelyen mind a hibaterjedési elemzéseket, mind a különböző (szolgáltatásbiztonsági és teljesítmény-szempontból elemzett) hiányosságokat orvosló folyamatjavítási módszereket ismertetjük. Az esettanulmány reprezentatív abból a szempontból, hogy bemutatja egy okos szolgáltatás tipikus tervezési feladatait, ugyanakkor célját tekintve is releváns, illeszkedik a korszerű okos szolgáltatások feladataihoz
Az esettanulmányban bemutatjuk egy elektronikus kérvény feldolgozásának a folyamatát, amely mind ember által végrehajtandó lépéseket (a BPMN terminológia szerint: Human Task), mind automatizált lépéseket tartalmaz. Utóbbiak igénybe vesznek mind lokális (azaz a szolgáltatást nyújtó szervezet informatikai infrastruktúráján és szervezetén belül működő), mind távoli erőforrásokat. Utóbbiak tipikusan webszolgáltatás (XML Web service) formájában érhetők el.
A példában okos városi szolgáltatások, illetve alkalmazások engedélyezését támogató folyamatot modellezünk. A világ számos városában (például London vagy New York) bevett folyamata van annak, hogy lehet a város által biztosított adatforrásokra épülő alkalmazást készíteni, illetve engedélyeztetni. Az engedélyezés szerepe alapvetően az, hogy igazolja, hogy az alkalmazás jogszerűen használja az egyébként tipikusan ingyenesen elérhető adatokat, megfelel a város által támasztott felhasználási szabályoknak, ideértve az erkölcsi/etikai, valamint az adatvédelmi követelményeket, és megfelelően utal az adatok forrására, szükség szerint hivatkozva azokat. Bizonyos esetekben (például a londoni tömegközlekedési információkat felhasználó alkalmazások esetében) az adatforrásra való hivatkozás, adott esetben a logó megjelenítése vagy akár az alkalmazás publikálása szintén ahhoz a feltételhez kötött, hogy az adott alkalmazás megfeleljen a felhasználási elveknek.
Az így regisztrált applikációk az élet számos területén segítséget nyújthatnak, a közlekedés és a programok tervezésén túl akár a nagyvárosi élet önszerveződő közösségeinek támogatásával (például piactér, közösségi megvalósítású projektek stb.).

Az engedélyeztetési folyamat az alábbi lépésekből áll:

- Bejelentés fogadása (Receive request).
- Bejelentés ellenőrzése (Validate request).
- Ezek után döntés következik, amelynek alapja a bejelentés tartalma. Amennyiben az alkalmazás által felhasználni kívánt adatok köre, mennyisége és célja ezt indokolja, akkor a bejelentés további vizsgálata szükséges.
- Amennyiben a döntés eredményeképpen az alkalmazás egyszerűsített ellenőrzése is elegendő, kézi, egyszerűsített vizsgálat eredményeként áll elő a döntés.
- Amennyiben az alkalmazás részletesebb vizsgálata szükséges az engedély megadásához, két párhuzamos lépés kerül végrehajtásra:

- Feltételezzük, hogy kiemelt információkat csak olyan alkalmazás kezelhet, amely köztartozásmentes, így ennek webszolgáltatáson keresztül történő ellenőrzése szükséges.
- Egyben ellenőrizzük, hogy a lokális nyilvántartás szerint az alkalmazás készítője (magánszemély vagy cég) szerepel-e azon cégek/magánszemélyek nyilvántartásában, akik ki vannak zárva az információ felhasználói köréből (például korábbi visszaélés vagy tisztázatlan tulajdonosi szerkezet miatt). Ez lokális (az adott szervezet informatikai infrastruktúráján belül gondozott) adatok alapján ellenőrizhető.

Végül bármilyen módon történt a döntés, az engedélyezésről/elutasításról a kérvényezőt értesíteni kell. Ennek modellezése történhetne egyetlen lépésben is, azonban az értesítésnek lehetnek olyan vonzatai (például az engedélyezett alkalmazásokról másolatot kap egy központi szerv), amelyek indokolják a szemantikailag különböző döntések különböző lépésként történő modellezését. Itt kiemelendő, hogy – tekintve, hogy a BPMN támogatja a hierarchikus modellezést – ezek a lépések a modellezés finomítása esetén akár alfolyamatként kibonthatók, amelynek felépítése értelemszerűen különbözhet az eltérő döntések esetén.



5. ábra. Esettanulmány: kérvény elbírálási folyamata.
Forrás: a szerzők saját szerkesztése

5. TELJESÍTMÉNYHIBA-ÉRZÉKENYSÉG VIZSGÁLATA MODERN HIBATERJEDÉS-ELEMZÉSI MÓDSZEREKKEL

Napjaink IKT-rendszereinek jellemzője, hogy nagymértékben elosztottak – a szolgáltatások megvalósításában nagyszámú fizikai és logikai komponens vesz részt – és komponáltak: szolgáltatásokat más szolgáltatások intelligens bevonásával, azok működésére hagyatkozva valósítanak meg. A kompozíció tipikusan magával vonja azt is, hogy az egyes szolgáltatásokon az őket igénylő további szolgáltatások „osztoznak", azt megosztott szolgáltatásként, illetve erőforrásként (shared service, vagy shared resource) használják. Különösen igaznak tekinthetőek ezen jellemzők az „okos város" (Gil-Garcia–Pardo–Nam 2015), és azon belül, illetve ahhoz kapcsolódóan az okos közigazgatás kontextusában; valamint tágabb értelemben a kiberfizikai rendszerek (Cyber-Physical Systems, CPS [Rajkumar et al. 2010]) és a „dolgok internete" (Internet of Things, IoT) területein.

Mint azt Jakab–Kocsis–Gönczy 2018a és Jakab–Kocsis–Gönczy 2018b részletesen bemutatja, a komplex informatikai függőségi rendszerekre támaszkodó, ám egyben szolgáltatásbiztonsági és teljesítménymegfelelőségi elvárásoknak is alávetett rendszerek tervezésében és üzemeltetésében alapvető fontosságúak a szolgáltatásra vállalt szolgáltatásiszint-szerződések (Service Level Agreement, SLA). A tervezés és üzemeltetés során elsőrendű cél kell hogy legyen az SLA-k megsértésével fenyegető kockázatok (risks) tudatos kezelése. Ennek szokásos an elemét képezi a) a lehetségesnek tekintett ok-okozati hatásmechanizmus-láncok strukturált felmérésre; b) a vonatkozó eseményvalószínűségek becslése; c) a hatásmértékek becslése és d) amennyiben szükséges, a kockázatkezelési mechanizmusok azonosítása és bevezetése. Jakab–Kocsis–Gönczy 2018a áttekinti az informatikai teljesítményelégtelenség kezelésének szokásos mechanizmustípusait (hangsúlyosan kapacitásmenedzsment fókusszal), Jakab–Kocsis–Gönczy 2018b áttekinti a rendszerszintű hatásmechanizmus-elemzés legfontosabb megközelítéseit, kitérve az informatikai rendszerek teljesítményhibáinak terjedésére is.

A korábbi munkák azonban jórészt érintetlenül hagyták az érzékenységvizsgálat (sensitivity analysis) és az elemzéssel integrált mechanizmustervezés, illetve mechanizmuskiválasztás kérdéskörét. Jelen fejezetben fő célunk annak bemutatása, hogy ezen feladatok a Jakab–Kocsis–Gönczy 2018b tanulmányban bemutatott, modern hibaterjedés-elemzési megközelítéssel integrált módon jól támogathatók. Mint eddig is, tárgyalásunk erősen demonstrációs jellegű. és nem feltételez elméleti előképzettséget; a kapcsolódó matematikai és fogalmi keretrendszert rendre Pataricza 2008a és Pataricza 2008b tárgyalják, néhány kiterjesztését pedig Kocsis 2018.

Egyszerű demonstrációs példaként a már korábban bevezetett munkafolyamat szolgál. A hibaterjedés-elemzés itt alkalmazott általános elveit adatfolyam-hálókkal Pataricza 2008a részletesen tárgyalja és formalizálja. Urbanics et al. 2014) adaptálta az elvet kifejezetten üzleti munkafolyamatokra.

5.1. Hibaterjedés munkafolyamat-végrehajtás során

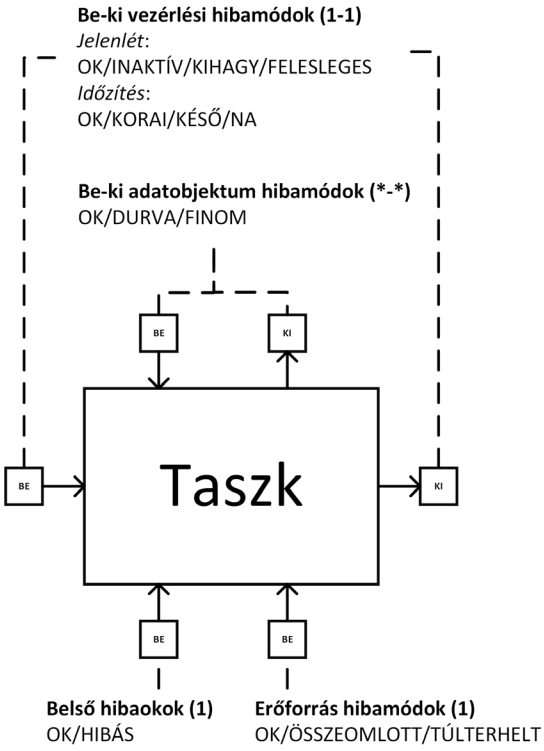
Mint azt Jakab–Kocsis–Gönczy 2018b-ben részleteztük, a komponált rendszerek hibaterjedésének az alkotóelemekre vonatkozó – hibaterjedési szabályrendszerek komponálásán alapuló – vizsgálatához alapvetően a következők határozandók meg a modellezés során.

1. Az egyes komponensek bemeneti hibákat és belső hibamódot potenciális kimeneti hibákra transzformáló hibaterjesztési szabályai.
2. Az összekapcsolt komponensek közötti „hibaátadást” leíró szabályok (bővebben lásd például [Bonfiglio et al. 2017]).

Részletesebben az 1. pontot tárgyaljuk; a 2. aspektus a példa bemutatása során magától értetődően fog adódni (specifikusan: a „hibaterjedés dinamikája” munkafolyamat-végrehajtás során jórészt követi a munkafolyamat-végrehajtás taszkok közötti aktivációátadás logikáját, azzal a kiegészítéssel, hogy az inaktivitást és felesleges aktivitást is szükséges explicitté tennünk az elemzés során).
A munkafolyamatok esetén érdemes megkülönböztetnünk a folyamat „érdemi” lépéseit adó taszkokat és a folyamat-végrehajtást vezérlő egyéb elemeket, mint a feltételes elágazások és a párhuzamos végrehajtású ágak. E két kategória között hibaterjedés-elemzés szempontjából két fontos különbség is adódik. Egyrészről a taszkok jellemzően rendelkezhetnek belső hibaokokkal, mint például hibás szoftver vagy rosszul betanított vagy csak egyszerűen hibát vétő, a taszkot végrehajtó operátor. A logikai elemek esetén, melyek végrehajtását egy dedikált munkafolyamat-végrehajtó motor végzi, jellemzően nem feltételezünk belső hibaokot (azaz a munkafolyamat-motor hibáját).
Másrészről bár a taszkokra adhatunk egy „általános”, alapértelmezett hibaterjedési szabályrendszert, de a rájuk vonatkozó szabályok optimális esetben a taszk működésének ismeretéből, a használt erőforrások tulajdonságaiból és az alkamazott hibatűrési megoldásokból adódnak (ideértve a teljesítményvédelmi mechanizmusokat is). Erre a példa-munkafolyamat „kézi ellenőrzés” taszkja ad példát, de mint látni fogjuk, a többi taszk „alapértelmezett” hibaterjedési szabályrendszere is tovább szűkíthető lenne, ezzel lehetségesen kisebb és „pontosabb” megoldás-hipotézishalmazhoz vezetve.

5.1.1. Taszkkomponens és hibamódmodell

A taszkok hibaterjedésének leírásához alkalmas a 6. ábrán bemutatott egyszerű komponensmodell.
Az ábra azt szemlélteti, hogy a hibaterjedés szempontjából alapvetően négy faktor befolyásolja egy taszk viselkedését: a vezérlési be- és kimenetek, a használt erőforrás(ok) hibaállapota, a használt adatobjektum(ok) hibaállapota és saját belső hibáinak (amennyiben ezek jelenléte feltételezhető) aktivációs állapota.



6. ábra. Taszk-komponensmodell munkafolyamatok hibaterjedés-elemzéséhez. Forrás: a szerzők saját szerkesztése

Hibaterjedés szempontjából a vezérlési ki- és bemeneteken alkalmazható hibaállapotokra – Patricza 2008a terminológiáját követve – könnyen adhatunk egy kváziszabványos szótárat. Érdemes szem előtt tartanunk, hogy egy taszk akkor kerül aktiválásra (végrehajtásra), amikor a munkafolyamat-szintű vezérlési logika ezt szükségessé teszi („ráfut"). Hasonló módon, amikor egy taszk végrehajtása befejeződik, a munkafolyamat-szintű végrehajtás a kimenő (vezérlési) élén halad tovább.

A munkafolyamat-taszkok közötti vezérlésátadás (és adott esetben adatátadás) hasonlatos ahhoz, mint ahogy az operációs rendszerekben „csővezetékekkel" (pipe) a folyamatok (processzek) kimenetét átirányíthatjuk más folyamatok bemenetére, ezzel összefűzve azokat egy hálózattá. Lényeges különbség azonban, hogy míg az így összekapcsolt folyamatok párhuzamosan futnak, mintha folyamatosan „folynának" az adatok a hálózatban. Ezzel szemben a munkafolyamat-modellek taszkjai önmagukban „lefutnak", és csak ezután lép tovább a hálózat, a végrehajtást mint „tokent" továbbléptetve.

Mivel a komponensenkénti hibaterjedés-leírás alapgondolata a specifikáció szerinti és a valódi viselkedés közötti eltérések szigorú kategorizálása, a hibaállapot-szótár elemeit a vezérlés „beérkezésének" és „továbbításának" helyes és helytelen kategóriái adják. Így a helyes aktivációbeérkezésen és -továbbadáson túl (az ábrán „OK") megkülönböztetünk a következő eseteket.

Kihagyás (omission): amikor jellemzően egy, a taszkot a topológiában megelőző erőforrás vagy a taszk hibája miatt a vizsgált taszkig nem jut el a vezérlés (míg a „helyes" viselkedés során ennek meg kellene történnie); vagy amikor saját vagy futtató erőforrásának belső hibája miatt nem adja tovább a vezérlést.

Felesleges aktiváció (commission): a munkafolyamatok egyik alapelemét adják az adatérték-vezérelt elágazások. Így ha egy külső vagy belső hibaokból adódóan a munkafolyamat-végrehajtás „rossz ágon indul el", akkor ezen aktivációk abban térnek

el a specifikáció szerinti viselkedéstől, hogy az érintett taszkok futtatásának a „helyes” viselkedés során nem szabadott volna megtörténnie.

Inaktív: az „OK” hibamóddal együtt a legmagasabb szintű „nincs vezérlésátadási hiba” fogalmat fedi, és azt fejezi ki, hogy egy adott taszk bemenetére végrehajtási vezérlés helyesen nem érkezik, vagy pedig, hogy egy taszk helyesen nem adja tovább a végrehajtási vezérlést. Informálisan: helyes, hogy egy bemeneten vagy kimeneten nincs tevékenység, mert a hibamentes referenciában sincs. Munkafolyamatok hibaterjedés-modellezésénél ezen eset alkalmazása jellemzően szükséges; hiszen míg például egy kiesett, taszkvégrehajtást támogató számítógép hatása egy aktiválandó taszk vezérlési kimenetére jellemzően „kihagyás”, addig egy inaktív taszkra nincs hatással. (És ekképpen a munkafolyamat szintjén értelmezve legalábbis az adott folyamat-végrehajtási példány kontextusában egy lappangó hiba.)

A helyes aktivációk esetén – de csak azok esetén – ezen túl szokásosan megkülönböztetjük azon eseteket is, amikor az aktiváció a specifikáció szerinti működéshez képest megkésve, vagy ellenkezőleg: túl korán történik.

Ez utóbbi, bár kontraintuitívnak tűnhet, bizonyos esetekben valóban okozhat, illetve jelezhet egyértelműen negatív szolgáltatási szintű hibahatást. Egyrészt a hibás tartalmú, illetve eredményű végrehajtás sokszor jóval gyorsabb lefutású, mint a specifikáció szerinti; másrészt különösen vezérlési jellegű feladatoknál a „túl korán” kiadott parancsok komoly vezérlési jellegű hibákhoz vezethetnek.

Ezen vezérlési hibamódokat egy minden egyéb szempontból hibamentes – lásd a következő bekezdést – taszk jellegzetesen egyszerűen „továbbítja” a vezérlés-kimenetére. Ez azonban nem minden esetben van így; például egy kihagyást egy „watchdog” képességekkel rendelkező, akár önmagát, akár a korábbi taszkok működését ellenőrző taszk transzformálhat OK továbbadott vezérléssé (bár a továbbadott adatok viszonylatában a hibát nem feltétlenül képes elfedni). Az egyszerű továbbadás azonban egy jó alapértelmezett szabály.

Egy taszk vezérlés-hibaterjesztési tulajdonságait – a hibaterjedés számításához alkalmazandó szabályokat – a bejövő vezérlési hibakategóriákon túl alapvetően befolyásolják azonban még további tényezők is. Ha a taszkot futtató erőforrás kiesett, akkor az a helyes vezérlésbemenetet kihagyássá alakítja át; az inaktívat helyben hagyja; a feleslegeset viszont (helyesen) inaktívra módosítja. Eközben egy túlterhelt erőforrás jellemzően csak a helyes bemeneti aktiváció-hibamódok időzítési tulajdonságait módosítja; egy már eleve késő bemenetet nem módosít, de a helyes vagy korai aktivációknál kikényszeríti vagy nemdeterminisztikusan megengedetté teszi a rendre „lassabb” kimeneti hibaállapot-kategóriákat.

A taszk belső hibaok-aktivációi további modulációkat tesznek szükségessé. Ezek forrása igen sokféle lehet; például kézi végrehajtású taszkoknál adminisztrátori/operátori tévedés, különböző programhibák, konfigurációs hibák és így tovább. Jellemzően ezek hatása vagy a kimeneti adatokban – amelyeket munkafolyamatok esetén jellemzően explicit adatobjektumokkal modellezünk – vagy a vezérlésátadás „megakasztásában” jelentkezik (például OK → kihagyás, felesleges → inaktív), és kihatással lehet a helyes végrehajtások időzítési tulajdonságaira is. Modern, kompozíciós szemléletű hibaterjedés-elemzés során, hacsak nem rendelkezünk további ismeretekkel a hiba hatásairól, mindezen hibamódokat mind lehetségesként értelmezzük. Amennyiben hatásuk a követelményekből adódóan nem tolerálható, úgy vagy olyan védelmek beépítésére van szükség, amelyek minden, lehetségesnek feltételezett hibamódot megfelelően kezelnek, illetve elfednek, vagy megvizsgálandó, hogy mely hibahatások zárhatók ki megnyugtatóan.

Végül befolyásolja a vezérlési hibaterjedést – és maga is a teljes hibaterjedési lánc egy komponensét alkotja – a taszkok által olvasott és írt adatobjektumok helyessége,

illetve hibás volta is. Az adatobjektumok alkalmazása nem kötelező a példánk keretét adó BPMN munkafolyamat-leíró nyelvben, egy sor hibatípus analitikus „követhetőségéhez" azonban fontos jelenlétük felvétele, amennyiben a taszkok között vezérlési függőségen túl valóban fennáll adatfüggőség is (azaz szükséges lehet az adatfolyamok explicitté, a modell által is hordozottá tétele). Az adathibamódok tekintetében szokásosan megkülönböztetjük legalább az ún. „durva" (coarse) és „finom" (subtle) hibákat (Bondavalli–Simoncini 1990). A kategorizálás alapgondolata, hogy az adathibák egy részét a rendszer elemei képesek adatellenőrzéssel detektálni – ezek tartoznak a durva kategóriába. (Érdemes megjegyeznünk, hogy az adatok folyamatos, vagy legalábbis rendszeres futásidejű „auditja" egy alapvető hibatűrési minta az informatikában; lásd például Hanmer 2013). Definíció szerint detektálható hibák például az explicit hibajelzések, de tipikusan detektálhatók a rosszul formált, illetve az alkalmazási terület kényszereinek nem megfelelő adatok is. Klasszikus és egyszerű példája az utóbbinak a kül- és beltéri hőmérsékletmérés: bár reprezentációs szinten mind a -500, mind a +500 Celsius-fok leírható megfigyelés, az előbbi fizikailag lehetetlen, míg az utóbbi gyakorlatilag elképzelhetetlen az alkalmazási területen. Ezzel szemben a finom adathibák azok, amelyeknek detektálására a rendszer adott pontján nincs algoritmikusan alkalmazható megoldás – vagy amelyek detektálására a rendszer nincs műszakilag felkészítve.

A bejövő és kimenő adathibamódok, valamint vezérlési hibamódok egymásra hatása potenciálisan ismét csak sokféle lehet. Egy bejövő adathibákra – például nem megfelelő formátumú adat – nem megfelelő módon felkészített taszkmegvalósítás vezethet kihagyási hibákhoz (például szoftverösszeomláson keresztül) a helyes aktiváció során; másrészről egy megfelelő védelmekkel felszerelt, elégségesen ellenálló (resilient) komponens nemhogy nem omlik össze nem megfelelő bemenetre, de még az adathiba elfedésére (maszkolására) vagy korlátozott javítására is képes lehet.

A fent körvonalazott logikát követve felállítható egy olyan, formalizált logikai szabályrendszer, amely meghatározza a lehetségesnek tekintett bemeneti, belső és kimeneti hibamód-kombinációkat. Ezt ebben a munkában – jellegéből adódóan – nem közöljük. Kiemelendő azonban, hogy bár a befolyásoló tényezők közötti „keresztkapcsolatok" összetettnek tűnhetnek, azok többszintű, „ha... akkor" stílusú szabályokkal strukturáltan bevezethetők. Igaz az is, hogy az alapvető szabályrendszer újrahasznosítható minden egyes modellezett taszkra; azt taszkonként jellemzően

a) hierarchikusan finomítjuk – vagy kategóriakészletben, vagy a taszkot magát strukturálisan tovább finomítva (lásd Pataricza 2008a);
b) szűkítjük, hipotetikusnak felismert kombinációkat kizárva;
c) néhány esetben pedig egyes döntési ágait követhető módon „lecseréljük".

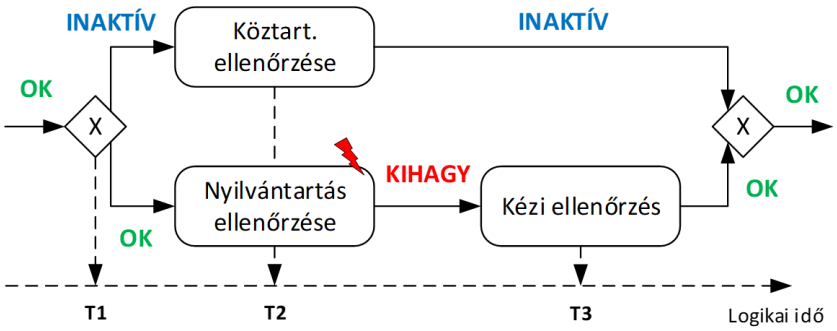
Megjegyzendő még, hogy összetettebb feladatokra nem elkerülhető a fent tárgyalt hibamódokból hibamód-futáskategóriák, ún. szindrómák képzése (mint azt a Jakab–Kocsis–Gönczy 2018a műben ismertettük). Erre jellemzően visszacsatolásokat is tartalmazó munkafolyamat-modellek esetén, illetve abban az esetben lehet szükség, ha munkafolyamat-példányok futtatásának egymásra hatását (például a használt erőforrásokon, vagy megosztott adaton keresztül) kívánjuk vizsgálni. A tárgyalás egyszerűsítése érdekében azonban e nézőpontokat is elhanyagoljuk.

5.1.2. Munkafolyamat-vezérlési elemek és hibaterjedés

A munkafolyamat vezérlési elemei hibaterjedés szempontjából felfoghatók különleges komponensekként; azokat a dedikált munkafolyamat-végrehajtó megoldás hajtja végre, így belső hibamódjaik ezen megoldás különböző hibáihoz köthetők. Mivel egy munkafolyamat-végrehajtó keretrendszer szoftverminősége és megbízhatósága könnyen lehet sokkal magasabb, mint az általa összehangolt taszkoké, egyszerűsített elemzés során ezen elemeket tekinthetjük belső hiba és erőforráshiba-mentesnek.
A két, a gyakorlat szempontjából talán legfontosabb csomóponttípus, a feltételes elágazás és a párhuzamos végrehajtás hibaterjedési logikájának alapelveit érdemes koncepcionális szinten tárgyalnunk.
A feltételes elágazás esetén a végrehajtás valamely feltétel ellenőrzése alapján választ az adott munkafolyamat-példányban alkalmazandó végrehajtási ágról. Így egy ilyen csomópont esetén mind a kihagyás, mind a felesleges aktiváció pontosan egy ágra terjed tovább (a többi marad inaktív); helyes végrehajtás egy ágra továbbadása során pedig megőrzi az időzítési hibatulajdonságokat (a többi ág pedig ismét csak marad inaktív). A választás alapját adó feltételvizsgálat azonban vezethet helytelen eredményre, amennyiben a feltételvizsgálat során olvasott adat hibás. Ebben az esetben a specifikáció szerint „helyes” ágon kihagyás típusú hiba terjed tovább, míg a specifikáció szerint szándékoltan inaktív ágon felesleges végrehajtás. Ezt a feltételes választást összefűző csomópont megszünteti – de a helytelen végrehajtás hatása ettől még terjedhet tovább a rendszerben az adatobjektum-függőségeken keresztül.
A párhuzamos végrehajtás hibaterjesztési logikája természetesen más; itt a párhuzamosító csomópont bemeneti vezérlési hibáit „átmásolja” kimenetire, míg az összefűzés során csak akkor megy tovább – helyesen vagy feleslegesen – a vezérlés, ha a párhuzamos ágak végrehajtása során nem jelentkezik kihagyás típusú hiba.

5.1.3. A hiba terjedésének folyamata

A hibaterjedés dinamikus jelenség; a bemutatott elveket így érdemes idődiagrammal is megvilágítanunk, még ha a számítógépes elemzés maga nem is feltétlenül követi ezt a sorrendiséget a lehetséges hibaterjedések előállítása során.



7. ábra. Hibaterjedés idődiagramon.
Forrás: a szerzők saját szerkesztése

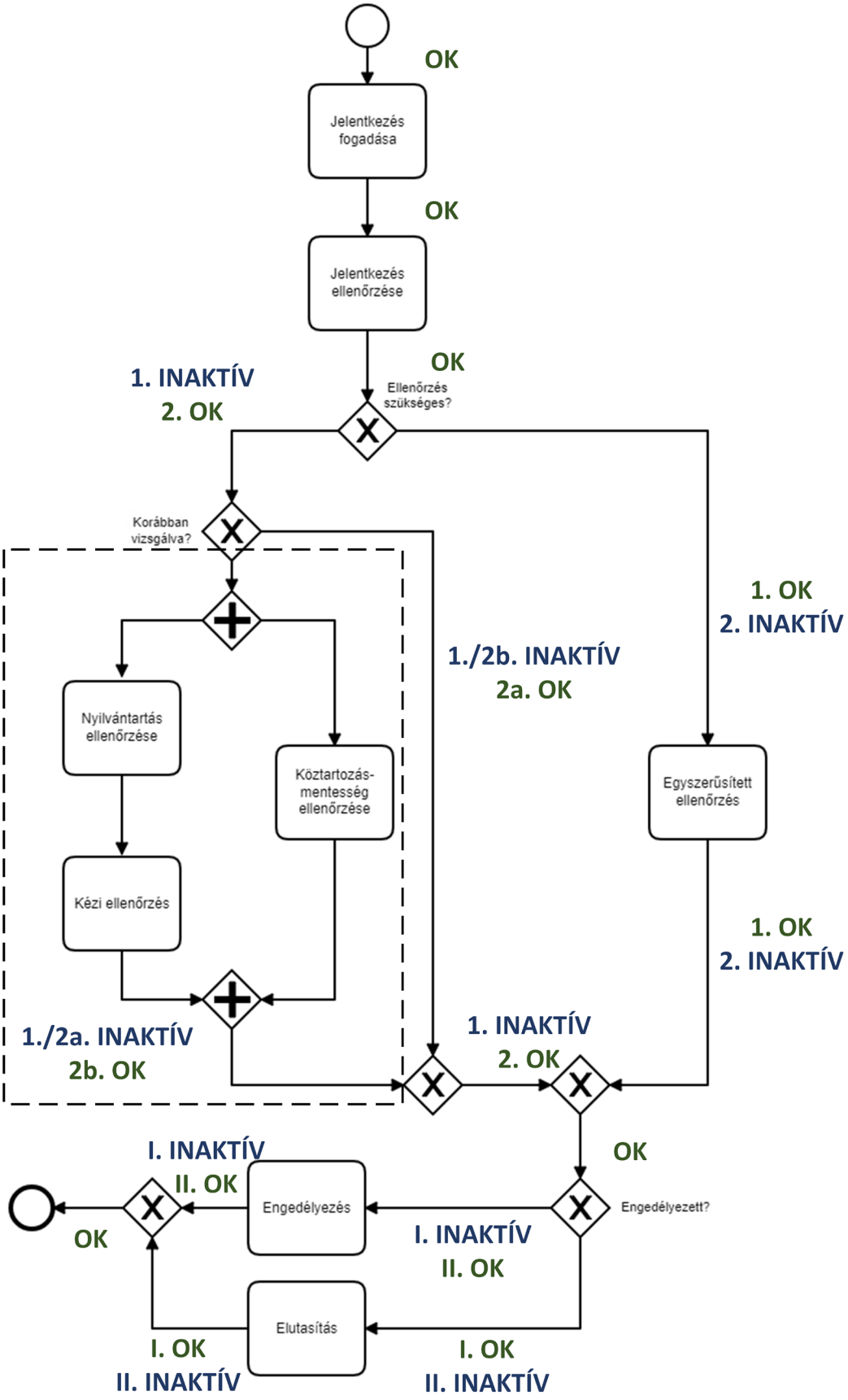
A 7. ábra olyan esetet mutat be, ahol a „Nyilvántartás ellenőrzését” megvalósító fizikai erőforrás (számítógép) elromlott, de a „Kézi ellenőrzés” ezt képes lesz felismerni és helyettesíteni. Az egyes logikai időpillanatokban a következők történnek:

- T1: (helyesen) beérkezik a végrehajtás az elágazáshoz. Az elágazási logika alapján a végrehajtást az alsó ágnak adjuk tovább, a felső végrehajtatlanul marad.
- T2: az alsó ágon helyesen beérkező végrehajtást a taszk „megfogja”; így rajta a kihagyás hiba terjed tovább. A felső ágon tovább terjed az inaktivitás.
- T3: a kihagyást a kézi ellenőrzés felismeri, és a tevékenység helyes végrehajtásával helyes továbbterjedő végrehajtássá alakítja.

Amint az látszik is, ebben a példában alapvetően logikai időt alkalmaztunk; a modellezett hibákhoz és a munkafolyamat-végrehajtás logikájához ez illeszkedik legalkalmasabban.

5.2. Egy hibaterjedési példa

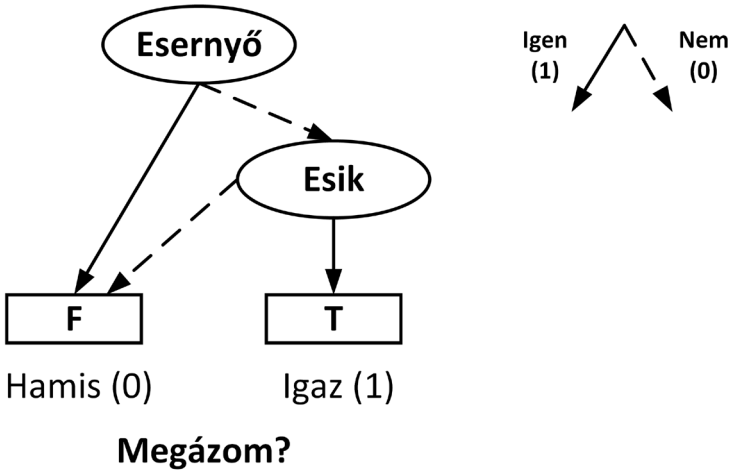
A hibaterjedés-elemzést a már korábban ismertetett munkafolyamat-példán szemléltetjük, élve azzal az egyszerűsítéssel, hogy adatfüggőségek jelenlétét nem feltételezzük. Törekedtünk arra, hogy a korábban körvonalazott logikai következtetési elvek segítségével az elemzés manuálisan is követhető legyen. Az elemzési módszer algoritmikus vetületeit Jakab–Kocsis–Gönczy 2018b szemléltette, Pataricza 2008a és Pataricza 2008b pedig matematikai precizitással tárgyalja.
A 8. ábra bemutatja a hibaterjedési elemzés által lehetségesnek vett hibaterjedési kombinációkat a hibamentes esetben. Érdemes megfigyelnünk, hogy az elágazásoknak köszönhetően a hibák jelen nem léte ellenére is több megoldás adódik – mindemellett a rendszerszintű hibahatásmód (failure mode) szintjén a megoldások egyenértékűnek, azonosan jónak tekinthetők. A többszörös megoldás az alkalmazott hibaterjedés-elemzési megközelítés egyik alaptulajdonságának tekinthető nemdeterminisztikus modellezésből adódik; a feltételes választás jellegű csomópontok esetén „számba vesszük” a vezérlés (helyes) átadását mindkét ágra.



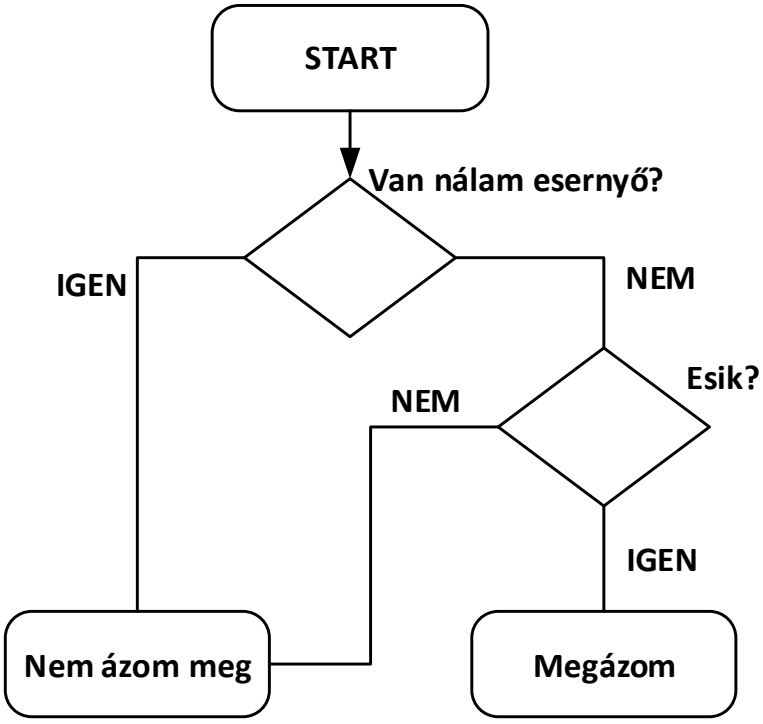
8. ábra. A munkafolyamat hibaterjedési megoldásai hibamentes esetben. Forrás: a szerzők saját szerkesztése

5.2.1. Megoldástér-leírás döntési diagramokkal

Míg ez a megközelítés kétségkívül szabatos és pontos, előrevetíti azt is, hogy a lehetséges megoldások száma a hibaaktivációs minták bevezetésével még tovább többszöröződik. (Ez egyben egy egyszerű, de remélhetőleg meggyőző szemléltető példa is arra, hogy a gyakorlatban szoftverrendszerek esetén miért reménytelen sokszor például az FMEA- és FTA-jellegű elemzések kézi végrehajtása.)
Ennek szemléltetésére tegyük fel, hogy – a kézi ellenőrzésen kívül – minden taszk végrehajtásához dedikált erőforrást rendelünk, valamint hogy feltételezzük, hogy a köztartozás-mentesség ellenőrzéséhez rendelt erőforrás elromolhat. Pusztán ezzel a feltételezéssel élve 16 megoldás adódik. Ennek munkafolyamat-ábrára annotálása már nehézkesen követhető; ehelyett a belső hibaok változókat (ideértve mind az erőforrások, mind a taszkmegvalósítások belső hibáit) és a kimeneti hatásokat foglaljuk össze egy úgynevezett többértékű döntési diagramon (multiple decision diagram, MDD; lásd 11. ábra). A döntési diagramok az informatika egyik alapvető relációreprezentációs és kezelési megközelítése. Koncepcionálisan legegyszerűbb változatuk, a bináris döntési diagramok (binary decision diagram, BDD [Akers 1978]) kompakt módon képesek egy kétértékű (0/1, igaz/hamis) változókészlet felett megengedett változóérték-kombinációkat reprezentálni oly módon, hogy egy rétegzett gráfban (a rétegeket a változóknak megfeleltetve) az éleket követve minden megengedett kombináció leolvasható. Ennek többértékű kiterjesztését adják az MDD-k.
A döntési diagramok területére kimerítő, matematikailag precíz bevezetést ad Pataricza 2006. Itt a döntési diagramokat egy egyszerű példával vezetjük be. A 9. ábra azt a logikai függvényt írja le BDD-ként, mely „Hamis”, ha nem ázunk meg, és „Igaz”, ha megázunk. Megázás egy esetben történhet: ha nincs nálunk esernyő, és esik is az eső. A 10. ábra a leolvasás, illetve kiértékelés folyamatát adja meg a BDD-re.

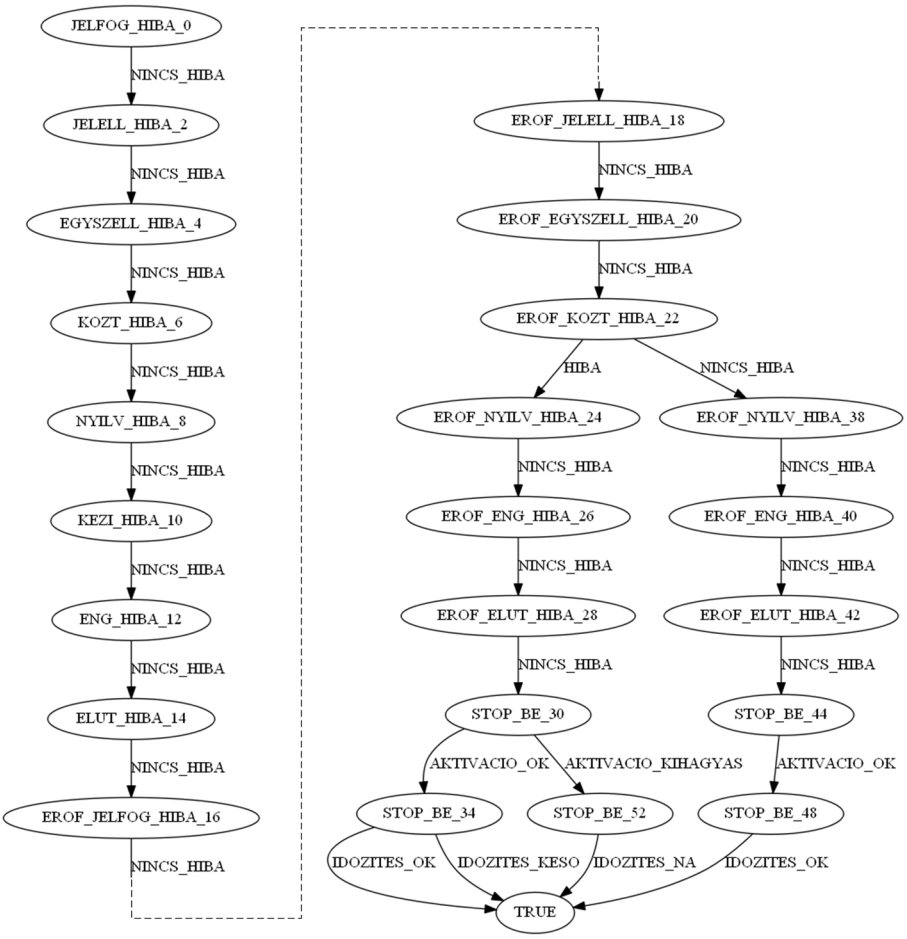


9. ábra. A „megázás” logikai függvénye BDD-ként.
Forrás: a szerzők saját szerkesztése



10. ábra. A BDD kiértékelésének folyamatábrája.
Forrás: a szerzők saját szerkesztése

A 10. ábrán bemutatott döntési diagramról – a jelen esetben szándékoltan „szerencsés” változó-sorrendezésnek köszönhetően – mind a megengedett hibaokok lehetséges szolgáltatási szintű hatásai, mind pedig az egyes hibahatásokhoz potenciálisan vezető hibaok-aktivációk leolvashatók. A teljes, az ok és végső okozati jelenségekre nem szűrt döntési diagram esetén a teljes lehetséges hibaterjedési utakra is igaz lenne ez; a teljes gráf azonban még a legegyszerűbb reprezentáció esetén is több tucatnyi szintet alkalmaz, így itt nem lenne hatékonyan bemutatható.



11. ábra. Hibaterjedési megoldáshalmaz többértékű döntési diagramként reprezentálva. (A csomópont-feliratokban a számok numerikus csomópont-azonosítók, az ábra leolvasása során elhanyagolhatók.) Forrás: a szerzők saját szerkesztése

Az ábra magas szintű interpretációja az, hogy a „Köztartozás-mentesség ellenőrzése” taszkot végrehajtó erőforrás hibái:

a) vezethetnek a munkafolyamat megakadásához;
b) vezethetnek a végrehajtás késéséhez;
c) de lehetségesek olyan lefutások, ahol nincs hatásuk.

Megfordítva vizsgálatunkat a következőket jelenthetjük ki.

a) Helyes lefutás lehetséges mind hibás, mind hibamentes esetben. (A hibaterjedési utak vizsgálata kimutatná, hogy a hibás esetben vagy az erőforráshiba lappangása miatt, vagy azért, mert a végrehajtás elkerüli a hibás erőforrás feletti munkavégzést.)
b) Késést csak a fent nevezett erőforrás hibája okozhat.
c) Elakadást csak a fent nevezett erőforrás hibája okozhat.

Egyszerű példánkon – ahol ráadásul elemzési példánkban ismerjük a megengedett hibaaktivációkat is – e kijelentések a következő módon olvashatóak le a diagramról.
A „helyes lefutás” munkafolyamat esetén nyilvánvalóan azt jelenti, hogy a folyamatot „záró”, utolsó csomóponthoz eljutó vezérlésihiba-kategóriák rendre „AKTIVÁCIÓ OK” és „IDŐZÍTÉS OK”. Mivel a kimenethez rendelt változókat – okkal – az utolsó szintekhez

rendeltük, a döntési diagram által megengedett lekötések az utolsó két változószinten azt mutatják, hogy mind a helyes lefutások, mind pedig a kihagyások és a késések lehetségesek. Követve a diagram szintjeit „felfelé" egyenes utakat járunk be – mivel egy erőforrás hibáit engedtük csak meg, a többi nyilvánvalóan nem rendelkezhet „hibásnak" lekötéssel egy helyes megoldásban.
A visszafele követés során eljutunk a „Köztartozás-mentesség ellenőrzése" taszkhoz rendelt végrehajtó erőforrás hibáját (vagy hibamentes állapotát) leíró változószintig. Itt látható, hogy a kiszámított megoldástér figyelembe veszi mind a hibás, mind pedig a hibamentes esetet. A jobb oldali ág (hibamentes eset) megnyugtató módon a munkafolyamat-példány hibátlan befejezésébe vezet. A baloldali, hibás eset „kiválasztása" azonban a diagram alján a fent említett hibás hibahatásmódokat és egyben a hibamentes lefutást is „engedélyezi", illetve lehetségesnek jelöli. A „Köztartozás-mentesség ellenőrzése" taszkhoz rendelt erőforrás hibája „feletti" hibamódok szintén egyetlen lekötéssel rendelkeznek.

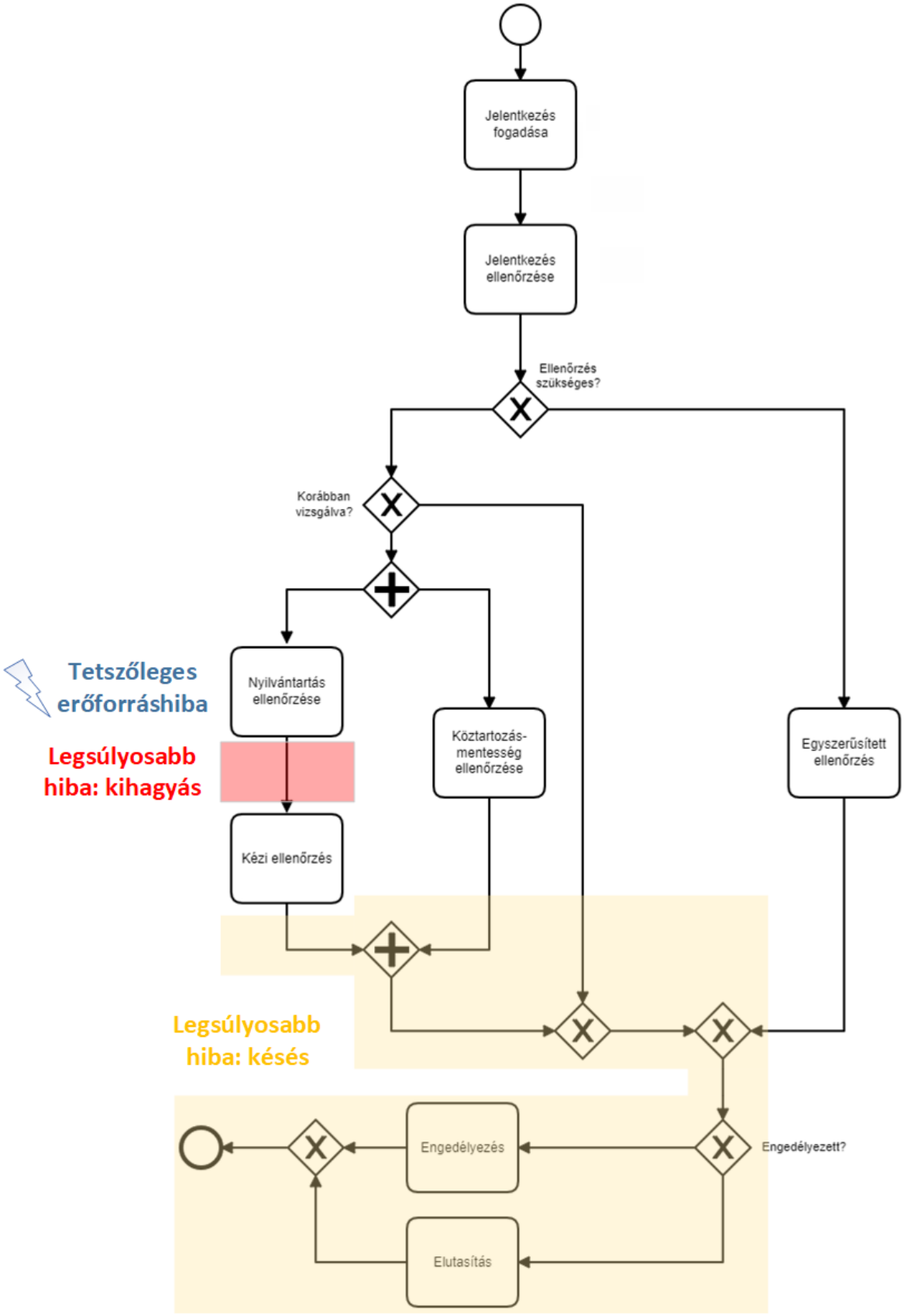
5.2.2. A változók sorrendezésének hatása

Megjegyzés: ez a pont mélyebb elméleti és megvalósítási aspektusokat taglal; feldolgozása opcionális, a következő részek megértéséhez nem szükséges.
Érdemes megfigyelni, hogy bár a „köztes", a taszkaktivációkon és taszk-nemaktivációkon keresztül történő hibaterjedési láncokat megragadó változókat „lehagytuk" a diagramról, még így is nagyban segíti a „leolvashatóságot" az, hogy a változó-sorrendezést a probléma – esetünkben: a munkafolyamat – szerkezetéből vezettük le.
Közismert tény, hogy a döntési diagramok méretét – azaz szélességét – alapvetően meghatározza a változók sorrendje. Egy szerencsétlenül megválasztott változó-sorrendezéshez tartozó diagramszélesség és az optimum között radikális különbségek lehetnek. Ez azokban az esetekben, amikor a döntési diagram különösen komplex, igen nagy változószámú relációkat kell hogy leírjon – például a hardver-, illetve szoftververifikáció és -validáció során – természetesen elsődleges fontosságúvá teszi a diagramszélességi értelemben minél jobb változó-sorrendezések alkalmazását. (Az optimális változó-sorrend előállítása számításelméleti értelemben nehéz probléma, így elterjedt a heurisztikák alkalmazása.)
Bár nem feltétlenül ad szigorúan vett matematikai értelemben optimális szélességet, az itt tárgyalt használati esetekben több szempontból is indokolt a változó-sorrendezés problémastruktúra – még pontosabban: komponenstopológia – alapján való kialakítása. Definíció szerint a hibaterjedésnek iránya van; a hibaforrásoktól mint gráfelméleti értelemben vett forrásoktól a rendszer valamilyen értelemben vett kimenete, amely felett a rendszerszintű hibahatást értelmezzük (gráfelméleti terminológiában: nyelő), „irányába" terjednek a hibák. Igaz ez még akkor is, ha a komponensek irányított összeköttetései nem körmentes gráfot (directed acyclic graph, DAG) feszítenek fel. (Megjegyzendő, hogy az itt alkalmazott hibaterjedés-elemzési megközelítés egyik nagy előnye, hogy mint azt a hivatkozott korábbi munkáinkban részleteztük, időbeli absztrakciót alkalmazó szindrómák segítségével az irányított köröket tartalmazó hálózatok is matematikailag pontosabban kezelhetők, mint más hibaterjedés-elemzési megközelítések alkalmazásával – az alacsony számítási igényt megtartva.) A hibaterjedés irányítottságának köszönhetően a mérnöki interpretációt, ami elsődleges szempont, leginkább a terjedési utak „lépéseit" követő struktúrájú döntési diagramok segítik.
Az algoritmikai részletek elhanyagolásával egyfajta struktúrakövető változó-sorrendezést nyerhetünk, például a következő durva algoritmusséma alkalmazásával. Először kialakítjuk a komponenstopológia egy olyan irányított feszítőfáját, amelyben

minden csomópontból vezet irányított út a rendszer „nyelőjébe” (vagy „nyelőibe”), ahol a hibahatást értelmezzük. Ezután sorrendezzük a csomópontokat a nyelő(k)től vett (inverz irányított út) távolság alapján; a legnagyobb távolságú csomópont(ok)ból alakítjuk ki a döntési diagram első szintjeit. Ezek feldolgozása után a következő, az előzőnél kisebb távolságú csomópontok kerülnek sorra, és így tovább, amíg elérünk a nyelőhöz. (Még könnyebben „leolvasható” struktúrákhoz jutunk, ha a „mélységi” tulajdonságokon túl figyelembe vesszük és egy változóblokkban vagy helyettesítő változóval reprezentáljuk az elágazás nélkül „sorba kapcsolt” komponenseket, de ennek tárgyalása jelen munka keretein túlmutatna.)
A matematikai optimalitás helyett a mérnöki interpretálhatóság előtérbe helyezését – amelyek esetlegesen persze egybe is eshetnek – az teszi lehetővé, hogy a hibaterjedési hipotézishalmaz jellemzően nem „túl nagy” – legalábbis a (bináris) döntési diagramokkal jellemzően támogatott verifikációs és validációs feladatokhoz képest. Amennyiben pedig mégis szükségessé válik szélességoptimalizáció alkalmazása, úgy továbbra is lehetőségünk van az optimalizált tárolás mellett kifejezetten az eredményinterpretációra szűrt és átrendezett diagramok alkalmazására – fenti MDD-nk erre is egy egyszerű példa (bár jelen esetben interaktív számítógépes megjelenítéssel még a teljes diagram is befogadhatóan „bejárható”).

5.2.3. A döntési diagramok konstruktív alkalmazása

Érdemes rámutatnunk arra, hogy a hibaterjedési hipotézishalmazokat reprezentáló MDD-k jellegükből adódóan közvetlenül alkalmasak mind diagnosztikai és futásidejű monitorozási szabályrendszerek kialakítására, mind pedig létező diagnosztikai és riasztási szabályrendszerek esetfedési és helyességi ellenőrzésére. Feltételezve, hogy minden taszk más fizikai erőforráson fut, az összes egyszeres fizikai hibához tartozó döntési diagram leírja az összes lehetségesnek vélt hibaok-hibahatás láncot. Ezen relációban a diagnosztikaelmélet ismert algoritmusait segítségül hívva megkereshetjük azt a minimális változóhalmazt, amelyek megfigyelésével egyértelműen különbséget tudunk tenni a lehetséges hibaokok között. Létező diagnosztikai és riasztási szabályrendszerek esetén pedig azt kell megvizsgálnunk, hogy helyes diagnosztikai/riasztási eredményt adnak-e a szabályok a hipotézishalmaz-MDD minden relációelemére.
Kisméretű példánkon talán nem tűnik indokoltnak sem az MDD-alapú hibaterjedés-reprezentáció, sem pedig az automatizált EPA alkalmazása. Nagyméretű, komplex rendszerekben azonban, ahol a taszkok és munkafolyamat-példányok osztoznak az erőforrásokon – és megfordítva: ahol többek között erőforrás-többszörözéssel védekezünk a hibák hatásának terjedésével szemben – az automatizált hibaterjedés-elemzés elengedhetetlen a logikailag lehetséges hibaterjedés-hipotézisek strukturált és kimerítő előállításához.
Jelen példánk előállítása során az MDD-ket csak mint reprezentációs eszközt hívtuk segítségül, az egyes megoldások létrehozása során az ún. korlátkielégítés-programozást (Constraint Satisfaction Programming, CSP – lásd például Szeredi–Benkő 2002) alkalmazva; aktívan folynak azonban kutatások, amelyek a modern, közvetlen MDD-alapú megoldáshalmaz-számító megközelítések a hibaterjedési problémára alkalmazását célozzák (lásd például Molnár – Majzik 2017).



12. ábra A „nyilvántartás ellenőrzése” taszk tetszőleges erőforráshibáinak kvalitatív hibabehatárolási tartományai a kézi ellenőrzés watchdogszerű működése esetén minden lefutási útra. Forrás: a szerzők saját szerkesztése

5.3. Teljesítményérzékenység-vizsgálat hibaterjedéssel

Példánk demonstrálta, hogy a különböző hibaokok által potenciálisan kiváltott hibahatások halmazának előállítására a modern rendszerszintű hibaterjedés-vizsgálat képes automatizált megoldást adni; míg a döntési diagramokkal való strukturált reprezentáció segít a megoldástér mechanizmusainak és nemdeterminizmusainak intuitív „megértésében". Kocsis 2018 ezt a megfigyelést oly módon viszi tovább, hogy a teljes megoldáshipotézis-halmazok felett vezeti be és formalizálja a kvalitatív hibabehatárolási tartományok fogalmát. A kvalitatív hibabehatárolási tartományok azt adják meg, hogy egy hibaaktivációs minta – a legegyszerűbb esetben: egy komponens valamely hibaokmódja vagy hibaokmódjai, de jellemzően egy alrendszer „egyszeres aktív hibaok" mintái (single fault assumption) – esetén a kiváltott legsúlyosabb hibák terjedésének hol vannak a határai a rendszerben. (A formalizmus ismertetését mellőzzük, azt Kocsis 2018 tárgyalja.)
Esetünkben hibabehatárolási tartományok szempontjából érdemes a „Nyilvántartás ellenőrzése" ágat vizsgálnunk. Mint azt a példa ismertetése során lefektettük, a „kézi ellenőrzés" a „nyilvántartás ellenőrzése" taszk viszonylatában egyszerre valósít meg egyfajta watchdog és végrehajtás-ismétlő funkcionalitást. Ha ugyanis észleljük, hogy a végrehajtás megakadt a nyilvántartás ellenőrzése ágon, a teljes ellenőrzést manuálisan végezzük el, amelynek ugyan a futásideje mindenképp jóval lassabb, mint az automatizált ellenőrzésé, ám másrészről a kihagyás vezérlési hibamódot késéssé alakítja át. Az így adódó, csökkenő súlyosságú hibabehatárolási tartományokat az 12. ábra szemlélteti. Figyeljük meg, hogy bár ismét lehetségesek hibamentes vezérlési utak, a hibabehatárolási tartomány kifejezetten a lehető legsúlyosabb esetek felmérése; így a kevésbé kritikus hibabehatárolási tartomány a folyamat végéig terjed. (Megjegyezzük, hogy a hibabehatárolási tartományok számításának kézenfekvő útja a döntési diagramok feldolgozása; az egyes lehetőségeket felsoroló megoldások, mint például a közvetlen korlátkielégítés-programozás, ebből a szempontból túlzottan korlátozott erejűek.)
A példa valójában nem teljesen triviális. A hibát „elkerülő" végrehajtási utak mellett ugyanis számba veszi a túlterhelt erőforrást is; ennek hatása azonban a kézi ellenőrzés után ekvivalens az összeomló erőforrásával, míg előtte a súlyosabbnak tekinthető kihagyás azt a hibaterjedési fedésvizsgálatban majorálja.
Korábbi munkáinkban demonstráltuk, hogy mind a szolgáltatásteljesítmény specifikációtól eltérése, mind ennek okai leírhatók és kezelhetők a hibatűrő számítástechnika hibaok-hibás állapot-hibahatás alapú hibaterjedési keretrendszerében. Ez sokszor nemcsak lehetőség, de egyben szükséges is: több példát is hoztunk arra, hogy a „kemény" hibák befolyásolhatják a teljesítményt – és fordítva: a teljesítményelégtelenség is vezethet közvetve „kemény" hibákhoz a rendszerszintű hibaterjedés során. Előbbire példánk is közvetlenül rávilágít.
A teljesítményfókuszú hibaérzékenység-vizsgálat egy lehetséges megközelítése így a teljes hibaterjedési hipotéziskészlet számítása, majd a szűrt és megfelelően rendezett, döntésigráf-alapú reprezentáció célvezérelt értelmezése, illetve a hibabehatárolási tartományok előállítása.
Míg az előbbi közvetlenül lehetővé teszi a védelmek szakértői tervezését, addig az utóbbi megteremti a rendszeren belüli többlépcsős kockázatmenedzsment lehetőségét. A kockázatkezeléshez természetesen szükséges ismerni mind a (szolgáltatási szintű) eseményhatás-súlyosságokat, mind pedig az esemény-valószínűségeket. Előbbi egy strukturált teljesítménymenedzsment-tervezési folyamatnak definíció szerint részét kell hogy képezze; az egyes hibaterjedési megoldáshalmazokhoz rendelhető valószínűségek becslése az elemi hibaesemények valószínűségeiből

pedig bevezethető a fenti hibaterjedés-elemzési megközelítésben (ennek ismertetése azonban túlmutat jelen mű keretein).

A hibaterjedéselemzés-alapú teljesítmény-érzékenységvizsgálat még egy lehetőségét szükséges kiemelnünk. A megközelítés lehetővé teszi, hogy mind az egyes komponensek különböző védelmekkel ellátását – ennek példája a fail-silent működés: bármely hiba esetén szigorúan „kihagyás" (omission) jellegű viselkedés, de a watchdog és hibajavító működések is –, mind pedig a tevékenységek erőforrásokra allokálását mint „szabad változót" a probléma-reprezentációba integráljuk. Így a teljes – MDD-vel reprezentált – megoldástér magában hordozza azt az információt, hogy a különböző védelmek alkalmazása és nem alkalmazása, illetve a különböző taszkparticionálások és redundanciasémák külön-külön milyen hibaterjedési megoldástereket eredményeznek. Következésképp nemcsak hogy összehasonlíthatóvá válnak az egyes lehetőségek, de az MDD-n megfogalmazott kényszerek és követelmények segítségével automatikusan kiválaszthatóvá válnak a közvetlen hibaterjedési és közvetett, hibabehatárolási tartományokra vonatkozó igények is.

A bemutatott módszer talán legnagyobb jelentősége az „okos" alkalmazások kontextusában pontosan ez. Mivel az okos alkalmazások szolgáltatásokat *integrálnak* „rendszerek rendszere" (System of Systems, SoS) jelleggel, szükséges, hogy hatékonyan tudjunk minőségi (kvalitatív) értelemben a felhasznált szolgáltatások lehetséges hibái és az általunk létrehozott szolgáltatás által vállalt viselkedés között kapcsolatot teremteni. Ez a számszerű, kvantitatív elemzést természetesen nem teszi okafogyottá az általános esetben; hanem a „tervezési tér bejárásával" megelőzi azt.

6. A FOLYAMATJAVÍTÁS MÓDSZEREI

A folyamat javításának több alapvető módszere van, tekintsük át ezeket a logikai javítástól a mennyiségi paraméterek javítását célzó módszerekig.
Folyamatok logikai helyessége alatt a folyamatnak azt a tulajdonságát értjük, hogy a specifikációnak megfelelően működik, és mentes a funkcionális hibáktól. Utóbbi hibák függetlenek a folyamat szakértői specifikációjától, inkább a folyamat „építőelemeinek" rossz használatából erednek. Fontos kiemelni, hogy a már ismertetett kvalitatív módszerek használata jelentősen egyszerűsít a logikai helyesség vizsgálatán, mert az egyes önálló adatértékek helyett úgynevezett ekvivalencia partíciókra bontva például a bemeneti változók értékét, a vizsgálandó esetek száma jelentősen csökkenthető.
Ekvivalenciapartíció alatt értjük azon elemek halmazát, amelyek egy adott tulajdonság szerint ekvivalenciarelációban vannak, leegyszerűsítve: egy adott tulajdonság vagy feltétel vizsgálata a halmazban lévő összes elemre ugyanolyan eredményre vezet. Például ha egy kérvény elfogadásakor az „ingyenes", „kis értékű" és „nagy értékű" lehet egy adott tulajdonság, akkor a 0-nál nagyobb, de a „kis érték" felső határánál nem nagyobb értékek ezen feltétel kiértékelése szempontjából egy ekvivalenciaosztályba (partícióba) esnek. Ezt a fogalmat a teszteléselméletben is használják, a kvalitatív modellek származtatásakor pedig értelemszerűen az lehet a szerepe, hogy a konkrét értékektől elvonatkoztatva megadja a lényeges értéktartományokat, illetve ha egy folyamat lehetséges végrehajtásait ténylegesen kipróbálni/tesztelni szeretnénk, akkor az ekvivalenciapartíciókból tudunk reprezentatív bemeneti értékeket (tesztadatokat) előállítani.
A folyamatot teljesen specifikáltnak nevezzük, ha minden pillanatban van értelmezett viselkedés az egyes változók lehetséges értékeire. Ha feltételezzük, hogy egy kérvény „Típus" jellemzője lehet „Egyszerű", „Kiemelt" és „Hibás", akkor a fenti folyamat nem tekinthető teljesen specifikáltnak, hiszen a kérvény elbírálásánál a „Hibás" kérvény kiértékelése nincsen lekezelve. A teljesen specifikáltság tehát megfelel annak, hogy minden pillanatban legyen „legalább 1" érvényes döntés.
A teljesen specifikált tulajdonság szisztematikusan ellenőrizhető, ha minden döntési ágnál vagy lehetséges bekövetkező eseménynél megvizsgáljuk, hogy az összes lehetséges kvalitatív tartományt lekezeltük-e (azaz az adott döntés szempontjából, egy változó teljes értékkészlete felosztható a döntés számára értelmezett ekvivalencia partíciókra). Itt kiemelendő, hogy a kvalitatív tartományok képzése adott esetben feladatfüggő is lehet, vagyis függhet attól, hogy egy adott folyamat milyen döntési feltételeket támaszt egy-egy változó értékével kapcsolatban. Ha például a kérvény kiértékelésénél szempont az, hogy milyen mennyiségű adatot érint, akkor a „kevés adat"-„nagy adat" particionálás célszerű, azonban ez változhat, ha a későbbiekben egy másik lépésnél a „sok adat" esetében például kötegelt lekérdezést biztosítunk adott adatmennyiség felett (például évenként), és a „kis adat"-„nagy adat"-„több részben kezelendő adatmennyiség" fogalmát emeljük be. Ebben az esetben vizsgálandó, hogy azok a döntési feltételek, melyek a „nagy adat" tartományra voltak értelmezve, minden esetben érvényesek lesznek-e a „több részben kezelendő adatmennyiség"-re.
Ha a modellfinomítás szokásos elvét követjük, és az egyes tulajdonságokat külön-külön finomítjuk, akkor a tulajdonságok értékeinek rendezettsége segít abban, hogy ezeket

a kérdéseket megválaszoljuk. Az előbbi példánál maradva, mivel az adatmennyiség tipikusan számmal írható le, amelyre érvényesek az alapvető aritmetikai műveletek (összeadás, kivonás) és relációk (kisebb, nagyobb stb.), az előbbi felbontás nem fogja azokat a döntési feltételeket érinteni, melyek a „kis adat”-„nagy adat” határát értékelik ki. A szakirodalomban predikátumabsztrakciónak nevezik azt a lépést, amelynek során egy-egy logikai IGAZ-HAMIS eredménnyel kiértékelhető kifejezés („X > 50”) igazságtartalma mentén információt hanyagolunk el annak érdekében, hogy a modell mérete, illetve a vizsgálandó esetek száma kezelhető méretű maradjon.
A folyamat logikai helyességének további feltétele a determinisztikus viselkedés, ami megfelel annak a követelménynek, hogy minden egyes pillanatban pontosan legyen eldönthető, hogy a folyamat egy adott input vagy esemény hatására milyen irányba lépjen tovább. Ez megfelel a „Legfeljebb 1 döntés lehetséges” feltételnek.
A teljesen specifikált ÉS determinisztikus folyamatok azok, amelyek logikai helyessége általánosságban igaz, de ebből nem következik az, hogy az adott folyamat valóban megfelel a szakterületi követelményeknek.

6.1. Lépések típusának finomítása, tulajdonságok definiálása

Annak érdekében, hogy a folyamat lépéseinek vizsgálatát támogassuk, r részben automatizáljuk, szükséges lehet az egyes lépések tulajdonságainak me ása. Egyfelől a lépések típuskészletét definiálja maga a BPMN-szabványt, másrészt a lépések típusára definiálható saját tulajdonságleíró nyelv is, amelyet technikailag is lehetséges az egyes folyamatelemekhez csatolni, azok „Property” (tulajdonság) készletét felhasználva. Ez nemcsak az elemzést támogathatja, de akár a folyamat megvalósításában is segíthet, mert különböző típusú/tulajdonságú elemekhez különböző szoftveres műveletek rendelhetők (a BPMN kiterjesztési lehetőségeinek használatával). Így például egy *felhasználói adatot olvasó lépésnél* kötelezően előírható a validálást elvégző programrészlet lefuttatása, vagy elemzési szempontból a folyamat funkcionális modelljéhez hozzárendelhető a hibamodell egy részlete is.
A programozási nyelvekhez hasonlóan a folyamatmodellezésben is ismert a kivételkezelés fogalma, amely voltaképp az elágazások egy speciális esete, amelyet előre definiált események aktiválhatnak (lásd alább). A kivételek ugyanakkor függhetnek attól, milyen lépés milyen típusú hibájára próbálunk reagálni, a bővíthető modell leírási mód ezt is támogathatja.
Az üzletifolyamat-modellező eszközök általában támogatják az egyes lépések típusának finomítását, illetve tulajdonságaik megadását. A típus itt lényegében megfelel az elemi folyamatlépés mint modellezési elem finomításának aszerint, milyen tevékenységet modellezünk az adott lépésben. A BPMN támogatja többek közt a típus fogalmának egyedi kiterjesztését, a modellezőeszközök alapból megkülönböztetik az ember által végrehajtott lépéseket (Human task), a helyben automatizált lépéseket (Script task) és a távoli szolgáltatás meghívását modellező lépéseket (Service task). Mivel az egyes lépések jellege nagyban befolyásolja a lépések végrehajtása során bekövetkező hibákat (és emiatt egyben támpontot is ad a hibatűrési tulajdonságok és mechanizmmusok modellezéséhez), az alábbiakban áttekintjük a BPMN lehetséges lépéstípusait és az azok végrehajtása során bekövetkező hibákat.

6.2. Folyamatlépések típusai és lehetséges hibáik

Az alábbiakban bemutatjuk a folyamatokban felhasználható lépéstípusokat és az azok tervezése és végrehajtása során lehetséges tipikus hibákat.

6.2.1. Üzleti szabályok végrehajtása folyamatokban

Az üzleti szabályok általánosságban olyan HA-AKKOR állításokat fogalmaznak meg, amelyek a folyamat által feldolgozott tokenek/adatok egyes tulajdonságain értelmezettek, és teljesülésük esetén vagy az adatok egyes tulajdonságait módosítják, akár új adatobjektumokat hoznak létre, akár más, az adatok köréből kilépő akciót hajtanak végre (Gönczy 2018).

Ezeknek a szabályoknak a leírására szolgál a Decision Modelling Notation (DMN) szabvány által definiált leírónyelv, amely üzleti szabályok csoportjainak leírását és folyamatba ágyazását támogatja. Üzleti szabályokkal tipikusan olyan döntéseket érdemes leírni, amelyekre az alábbi feltételek teljesülnek:

- A döntések adatvezéreltek, ismételhetők, a döntési logika sok esetben csak a paraméterekben tér el.
- A döntéseket az adott pillanatban rendelkezésre álló információk alapján meg lehet hozni (nincs szükség további felhasználói információra vagy távoli szolgáltatások igénybevételére).
- A döntéseket több helyen is felhasználjuk, azaz nem feltétlenül köthető egyetlen alkalmazáshoz, hanem ismételhető módon elérhetővé akarjuk tenni. Ennek tipikus példája lehet egy kedvezményszámítás, jogosultságkiértékelés, engedély formai ellenőrzése stb.

Azt a fajta modellezési megközelítést, amikor egy szakterület döntési mechanizmusait a modell szintjét megjelenítjük, externalizációnak nevezzük.
Maga a DMN-szabvány több felhasználási esetet definiál:

- Emberi (manuális) döntéshozatal támogatása. Ebben az esetben a szabályok és az egész modell dokumentációs célt szolgál, nem hajtható végre automatikusan, így bár a tervezést és a tervezői döntések meghozatalát és kommunikációját segíti, nem garantálható, hogy az elkészült rendszer a szabályoknak megfelelően működik. A szabályok ugyanakkor ebben az esetben is felhasználhatók a logikai hibák kivédésére, amennyiben például ellenőrző lista (*checklist*) készíthető, amely a manuális döntést segíti. Ezt a fajta ellenőrző listát a gyakorlatban is sok helyen alkalmazzák ügyfélkezelési folyamatokban. Ez nemcsak az adott döntéshozatalt segíti, hanem bármilyen utólagos panaszkezelés, ellenőrzés vagy teljesítményértékelés alapját képezheti.
- Lehet cél az automatizált döntéshozatal teljes specifikációja, továbbra is azt feltételezve, hogy rendszer maga nem szabályalapon működik, hanem tetszőleges programozási nyelven. A szabályok ebben az esetben már megfeleltethetők kell hogy legyenek az automatizált döntéshozatal egyes elemeinek vagy elemcsoportjainak (például egy programnyelven megírt ciklus egy feltétele vagy egy elágazást kiértékelő kód egy részlete), így az alábbiakban tárgyalt hibamódok végiggondolását és szisztematikus megelőzését a szabályalapú gondolkodás segítheti.
- Végül a teljesen automatizált, szoftverfejlesztői munkát nem igénylő, a folyamatba ágyazható szabályok, illetve szabályrendszerek is megfogalmazhatók a nyelven. Ebben az esetben a szabályok ellenőrzése – a végrehajtó környezet helyes működését feltételezve – egyben az alkalmazás, illetve az adott folyamatlépés ellenőrzését is jelenti.

Üzletiszabály-alapú döntéseknél az alábbi hibatípusokra érdemes felkészülni:
Vezérlési hiba jelentkezhet akkor, ha a szabályrendszer logikailag helytelen döntést hoz. Ennél a hibatípusnál a szabályrendszer előzetes ellenőrzésének teljessége döntő lehet, hiszen egy bonyolult feltételrendszernél előfordulhat, hogy a feltételek bizonyos kombinációinál lép fel egy olyan téves döntés, amely adott esetben nem a szabályrendszer szintjén, hanem a későbbi folyamatlépéseknél okoz hibát. Vezérlési hiba akkor jelentkezik, ha olyan értéket adunk meg a szabályok kiértékelésének hatására, amely egy vezérlési csomópont döntési feltételében is szerepel, és a döntés szempontjából lényeges eltérés jelentkezik. Itt visszautalva az ekvivalenciapartíció fogalmára, vezérlési hiba léphet fel akkor, ha nem azonos ekvivalencia osztályba tartozik a jó és a hibás érték. Ennek előzetes ellenőrzését a szabálydefiníció szintjén nehezítheti, ha egy adott szabálycsoportot több célra is felhasználunk, és a szabályok tervezésekor és fejlesztésekor az ekvivalenciapartíciók nem ismertek egyértelműen, így ellenőrizni sem lehet ezeket. Példa lehet erre, hogy ha egy kérvény kiértékelése során nem csak az elbírálási szempontokat értékeljük ki, hanem például a kérvényező valamilyen adata alapján a kérvényt különböző folyamatokhoz irányítjuk. Ebben az esetben a kérvényezőnél nemcsak az „engedélyezhető/tiltott” döntéshez használunk fel adatot, hanem a konkrét ügykezelőt is ez alapján jelöljük ki. Annak érdekében, hogy a hiba lehetőségét minimalizáljuk, javasolt a szabályrendszer fokozatos kiegészítése, a fenti esetben például azzal, hogy minden olyan információt, mely újabb döntések alapját képezi (akár a szabályrendszeren belül, akár későbbi döntéseknél), különböző változókra képezünk le, így a hiba behatárolása egyértelműbb, ami mind a futási idejű ellenőrzést, mind a tervezési/fejlesztési idejű tesztelést segítheti.
Végtelen ciklus: szélsőséges esetben akár az is előfordulhatna, hogy a szabályok kiértékelése végtelen ciklusba kerül (például egy adott feltétel hatására egy érték megváltozik, ami újra aktivál egy szabályt, például a „jóváhagyott” érték törlődik). A DMN-nyelv, bár elvben nem zárja ki ennek előfordulását, csökkentheti a végtelen ciklus bekövetkezésének valószínűségét azáltal, hogy a szabálycsoportok hierarchiába foglalásával rávezeti a tervezőt a strukturált gondolkodásra.
Teljesség: a szabályrendszer esetében is fontos a teljesség (teljesen specifikált tulajdonság), vagyis hogy a szabályok fel legyenek készítve az összes lehetséges bemeneti kombináció kezelésére. Automatizált végrehajtás esetén ennek egy lehetséges módja a kötelező „default” eredmény megadása, amely akkor lép érvénybe, ha semmilyen más kiértékelés nem vezetett eredményre.

6.3. Események feldolgozása

Mivel elosztott rendszerek esetén alapvető működési mód az aszinkron kommunikáció, amikor egy komponens (jelen esetben az üzleti folyamatot megvalósító elemek egyike) üzenetet küld egy másiknak, a BPMN támogatja ezt és az események definiálását is. Ez megfelel annak a működési módnak, amikor két folyamat (például egy engedélyeztetés két, egymástól független részlépése) egymástól az adatokon keresztül függ, de egyébként a két folyamat vezérlése független.
BPMN esetén az alapvető lépéstípusok közt megtaláljuk az üzenetküldés (Send) és fogadás (Receive) lépéseket. Ezeknél a hibamódok közt megkülönböztethetjük a lépés rossz definíciójából adódó tervezési hibákat és a futásidejű üzenetküldés hibáit. Tervezési hiba lehet, ha nem készültünk fel a teljes eseménytér kezelésére küldő vagy fogadó oldalon, azaz olyan üzenetet küld egy részfolyamat egy másiknak, amelyre az nincs felkészítve. Futásidejű hibák esetén megjelenik a már tárgyalt omission,

azaz elnyelés mint hibamód, amikor az üzenet nem érkezik meg, így a fogadó oldali folyamat nem tud továbblépni.
Üzenetküldés esetén megjelenhet a helytelen számú üzenettovábbítás problémája, azaz például egy ciklusban elküldött üzenetek esetén commission jellegű hiba léphet fel, ha a fogadó oldal nincs a megfelelő számú üzenet fogadására felkészítve, például már az első fogadott üzenet után továbblép a vezérlés.
A folyamatok közti aszinkronkommunikáció további problémákat okozhat akkor, ha a folyamatok időzítési problémák miatt nem adnak megfelelő választ. Ez esetben a kommunikáció aszinkron jellege miatt az is előfordulhat, hogy az egyik folyamat hibája a másik folyamatnál késleltetésként jelenik meg. A már ismertetett hibaterjedési módszereket célszerű kiterjeszteni annak érdekében, hogy több, egymással kommunikálható folyamat hibája is kezelhető legyen (Gönczy–Hegedüs–Varró 2011).

6.4. Funkcionális javítások

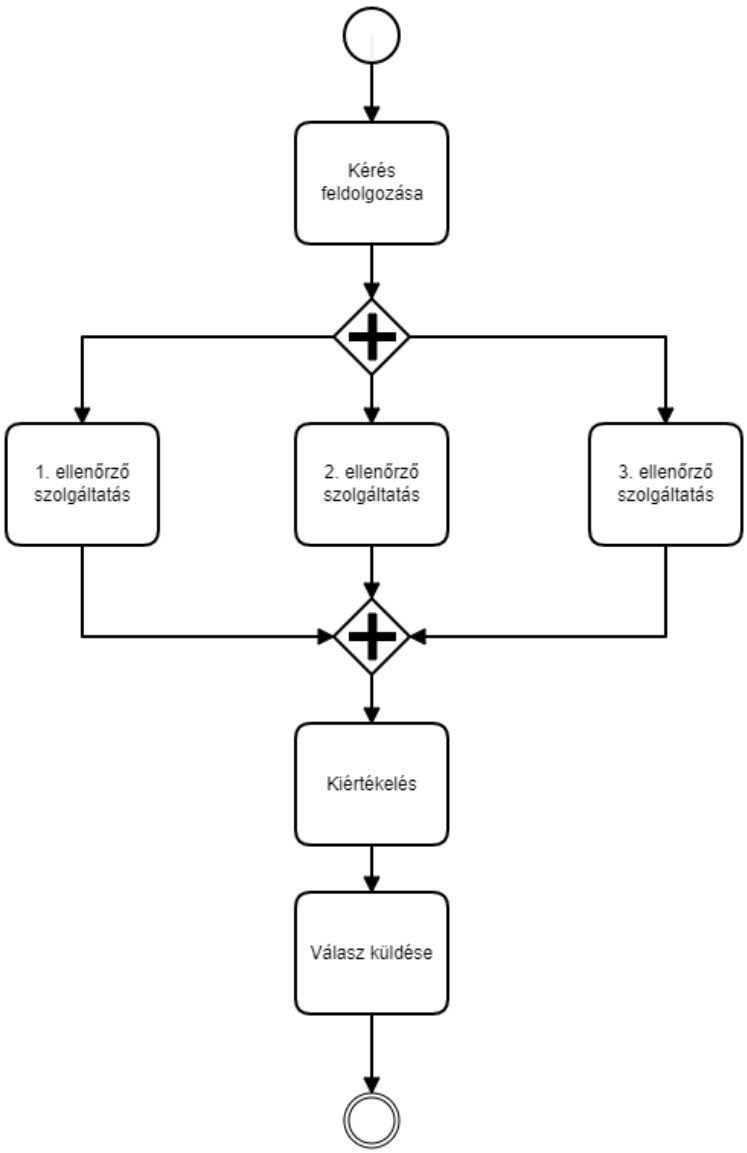
Az alábbiakban áttekintjük a klasszikus hibatűrő minták egy kiválasztott részhalmazának alkalmazását folyamatmodellekkel leírt rendszerekben.

6.4.1. Kivételkezelés

A folyamatmodellezési nyelvek támogatják a kivételek definícióját, hasonlóan a programozási nyelvekhez. A kivétel (Exception) megfelel egy speciális eseménynek, amely azt jelzi, hogy az adott lépés végrehajtása során valamilyen váratlan esemény következett be. Hasonlóan a programozási nyelvekhez, folyamatok esetén is lehetséges az adott kivételekhez kivételkezelési műveleteket definiálni. Modellezési szempontból ezek a BPMN esetében hasonlók ahhoz, mint amikor egy tetszőleges, aszinkron vagy időzített (azaz egy adott lépés logikai működéséből nem következő) eseményre reagál a rendszer azáltal, hogy az adott esemény után egy vezérlési ágat definiál, amely egy kivételkezelő lépést tartalmaz. Fontos kiemelni, hogy a modellezési megközelítés önmagában nem garantálja még azt sem, hogy az adott kivételkezelő esemény megfelelően működik, így természetesen, hasonlóan a rendszer normális működéséhez, a kivételkezelés mint működési ág ellenőrzése (akár teszteléssel, akár formális megközelítéssel) szükséges.

6.4.2. N-verziós programozás

Missziókritikus rendszerekben (például repülő vezérlése esetén) régóta alkalmazott alapelv az N-verziós programozás, amikor egy elemi szolgáltatás vagy funkció igénybevétele helyett több, egymástól függetlenül implementált, ugyanazt a funkcionalitást fizikailag és logikailag is elkülönített helyen megvalósító ún. variánst vagy verziót hívunk meg (Chen–Avizienis 1995). Folyamatmodellezés szempontjából ez megfelel annak, hogy egy elemi lépést kiegészítünk az alábbi sémát követő alfolyamattal:



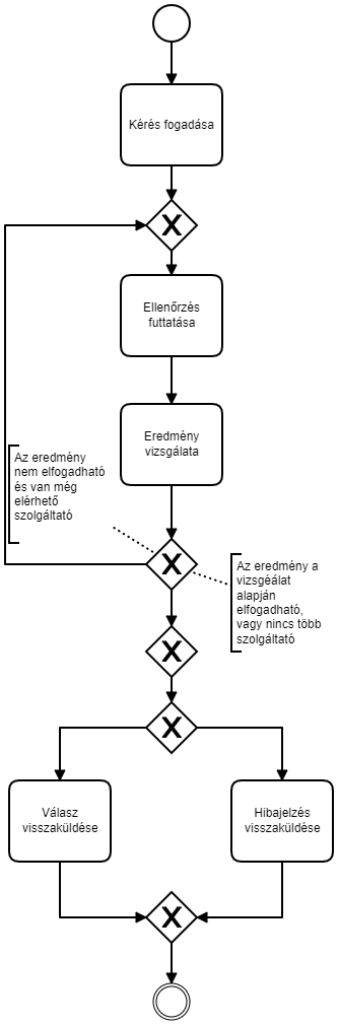
13. ábra. N-verziós programozás megvalósítása folyamatmodellben. Forrás: a szerzők saját szerkesztése

Hatása a folyamat megfigyelhető metrikáiban a következő:
A folyamat garantált, illetve elvárható/tervezhető **válaszideje** megnő, hiszen a leglassabb komponens/megvalósítás válaszát mindig ki kell várni. Emellett ugyanakkor javul a folyamat **rendelkezésre állása**, hiszen ez a minta képes kivédeni mind a tervezési hibák egy részét, mind a tranziens futási idejű hibák hatását. Így összességében a folyamat teljesítőképessége (az angol performability szakszóból) jobb lesz. Természetesen a minta alkalmazása nem minden esetben indokolt: példánkban a kérvényeket tipikusan nem érdemes minden esetben kézzel és automatikusan is ellenőrizni. Ugyanakkor például egy adatfeldolgozási folyamat esetén érdemes lehet egyes ellenőrzési lépéseket többféleképpen elvégezni (például különböző algoritmusokkal vizsgálni egy okosvárosi környezetben rögzített mérési adat hitelességét és hihetőségét).
Fontos kiemelni, hogy a módszer egyfelől feltételezi, hogy létezik egy olyan küszöbérték, amely alatt az eltérés elfogadható, illetve a szavazás minden esetben egyértelműen eldönthető (természetesen a modell kiegészíthető az időtúllépést mint hibakezelési módszert leíró ággal). Emellett, ha közös módusú hiba jelentkezik, vagyis

több komponenst is érint a hiba, vagy maga a specifikáció hiányos, akkor előfordulhat, hogy a módszer rossz választ ad.
Ezzel együtt előnye, hogy a döntés tipikusan gyors, és nincs szükség külső információforrásra (természetesen a szavazás eredménye képezheti újabb hihetőségvizsgálat tárgyát).

6.4.3. Recovery Block

A recovery block minta olyan, önálló funkcionalitást ellátó komponensre utal, amely a funkciót ellátó normál komponens elérése mellett rendelkezik egy tartalékmegoldással is, amely sok esetben lassabb, de robosztus eredményt ad (RANDELL–XU 1995). A kérések kiszolgálása során a vezérlő először a „normál” komponensnek adja át a kérést, majd ennek eredményét ellenőrzi. Ez az ellenőrzés lehet egy tartományok határát figyelembe vevő hihetőségvizsgálat, de a vezérlő komponens maga nem feltétlenül képes végrehajtani ugyanazt a számítást, mint az igénybe vett szolgáltatás. Ez megfelelhet például egy adatszolgáltatás ellenőrzésének: attól, mert egy azonosítót nem ismerünk, lehetnek olyan tulajdonságai (például a személyi szám esetén), amelyek legalább részleges ellenőrzést tesznek lehetővé.



14. ábra. Recovery block minta megvalósítása folyamatmodellben.
Forrás: a szerzők saját szerkesztése

A fenti ábra tetszőleges számú elérhető megvalósításra kiterjeszthető, a legtöbb tipikus esetben 2 vagy 3 az elérhető szolgáltatások száma.
Mit mondhatunk ezek alapján a GQM-módszerrel meghatározott metrikák teljesüléséről?
A folyamat **válaszideje** várhatóan megnő, azonban a hibamentes esetekre a válaszidő továbbra is az elsőnek választott (tipikusan gyors) komponens válaszidejével lesz egyenlő. Ugyanakkor. ha egynél több variáns hibázik, akkor a válaszidő az újrapróbálkozások miatt megnőhet, szélsőséges esetben nagyobb, mint az egyes szolgáltatások által nyújtott válaszidők összege.
A folyamat által adott eredmény megbízhatósága (hihetősége) annyiban jobb, hogy ha adott egy előre definiált ellenőrzési lehetőség, akkor függetlenül az egyes szolgáltatások hibáitól, akár rosszindulatú bemenetre is helyes választ (jelen esetben hibajelzést) ad a rendszer, hiszen az ellenőrzés a megvalósító szolgáltatástól független lehet.
Mindezek fényében ezt a megoldást akkor érdemes alkalmazni, hogyha a) arra számítunk, hogy a rendszer viszonylag ritkán fog hibázni, b) az eredmény külső ellenőrzése lehetséges, c) a megoldások közt teljesítmény/ár/… különbség van, így nem akarjuk mindig az összes szolgáltatást igénybe venni.
A recovery block minta jól illeszkedik a tipikus folyamattervezői döntésekhez (melyik szolgáltatás a preferált?), illetve amennyiben tranziens vagy kommunikációs hibák kivédésére szeretnénk használni, jól használható arra is, hogy a távoli, aktuális állapotot tükröző adatforrás és egy „elmaradt” állapotú, hiányos, de az esetek többségét lefedő lokális replika mint „backup” közti lekérdezéseket vezérelje.
A módszer használatának teljesítmény szempontjából történő optimalizálást vizsgálta Berman–Kumar 1999.

6.5. Adatfeldolgozási lépések és részfolyamatok hibái

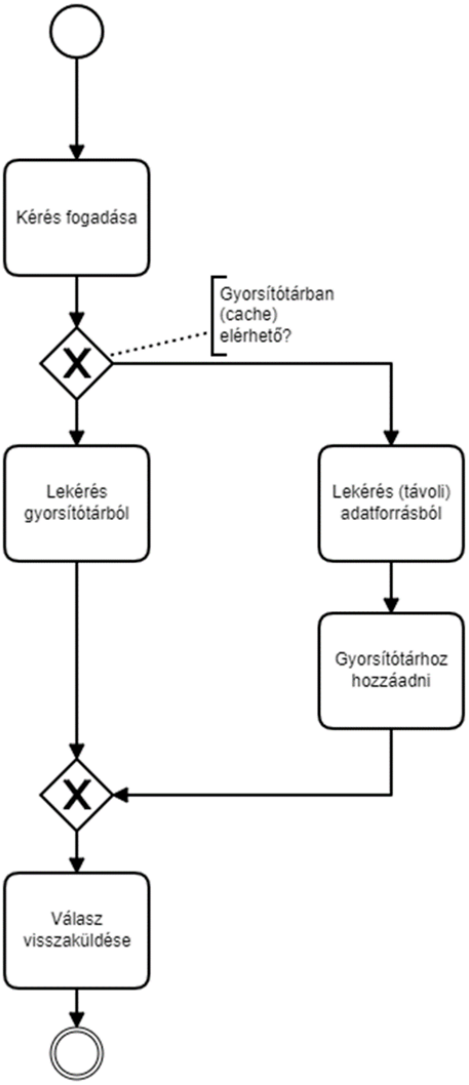
Bár a BPMN nem foglalkozik kiemelten az adatfeldolgozással mint folyamatlépéssel, számos munkafolyamat célja elsődlegesen adatokon végzett feldolgozási és kiértékelési lépések elvégzése.
Ilyen lépések lehetnek az alábbiak:
Adatok **standardizálása** (közös mérési skálára hozása), **normalizálása**, **hiányzó** értékek kiegészítése vagy törlése. A hiányzó értékekkel kiemelten fontos foglalkozni, mert számos algoritmus torz vagy nem megfelelő eredményt ad akár kis mennyiségű adathiba esetén, ugyanakkor ez a fajta „elnyelés” jellegű hiba kifejezetten gyakori olyan folyamatoknál, amelyek valós környezetben végzett adatgyűjtésből nyerik bemenetüket. Ha a folyamatot kiegészítjük a megfelelő lépésekkel, ennek a hibának a terjedését (a hiba aktiválását) kizárhatjuk, amennyiben az adatvesztés, illetve a hiány értéke az elfogadható mérték alatt marad.
Adatok **szűrése, aggregálása, származtatott adatok előállítása**. A nyers kiegészített adatokból szükséges lehet mind az adatmennyiség csökkentése, mind a több bemeneti adatból vagy egy adat időbeli változásából képzett származtatott adatok előállítása. Előbbi művelet támogatása lehet a szűrés (angolul filtering), amely során bizonyos értéktartományba tartozó adatokat megtartunk, a többi elhagyjuk. A szűrés érzékeny lehet nemcsak a bemeneti adatok hibájára, hanem arra is, hogy ha a szűrési feltételt nem megfelelően határoztuk meg. Különösen fontos lehet ennek vizsgálata akkor, ha nem szakértői vagy külső követelményből származó feltétellel dolgozunk (például azokat az adatokat vizsgáljuk, ahol egy érték magasabb, mint X), hanem olyan feltételrendszerrel dolgozunk, amelyet valamely előzetes elemzés vagy szoftver által adott javaslat eredményeként kaptunk. A manapság terjedőben lévő automatikus

analíziseszközök alkalmazása ugyanis amellett, hogy erős eszköztárat ad a felhasználó kezébe, azzal a potenciális veszéllyel jár, hogy egy kis mintán elvégzett következtetés alapján a teljes rendszerre nézve rossz feltételeket támasztanak, illetve a mért adatok jellegétől függően az adott szakterületre nézve téves következtetést vonnak le. Utóbbira példa lehet, ha olyan beépített szűrést használunk, amely az adatok „alsó” és „felső” X százalékát szűri ki. Amennyiben egy adott mintamérésben nincsenek olyan adatok, amelyek például felhasználói kérések tartalmát helyesen reprezentálnák, akkor a később ez alapján elvégzett szűrés például kizárhat olyan elemeket, amelyek előfordulása a teljes populációban akár jelentős mértékű is lehet.
Származtatott adatok képzésére példa lehet egy változó esetén a derivált használata, amely megfelel az adott metrika időegység alatti (adott esetben a legutolsó rendelkezésre álló értéktől számított) változásának. A kvalitatív modellezéssel foglalkozó tanulmányokban már leírtuk, hogy számtalan fizikai, gazdasági vagy akár humánfolyamat esetén az ún. *kvalitatív modellezés* jó közelítést ad a problémák leírására. Ennek egyik eszköze a változások vizsgálata, amelyet a tipikus adatfeldolgozó eszközök beépítetten támogatnak.

6.6. Folyamatok teljesítményének javítása

Az alábbiakban áttekintünk néhány módszert, amelyek, szemben az eddig bemutatott mintákkal, elsődlegesen a folyamat hibatűrésén próbáltak javítani, arra koncentrálnak, hogyan lehet a folyamat működését a felhasználó számára látható módon javítani a teljesítmény szempontjából, akár a folyamat válaszidejét, akár áteresztőképességét figyelembe véve.

6.6.1. Adatlekérdezési lépések gyorsítótárból történő kiszolgálása

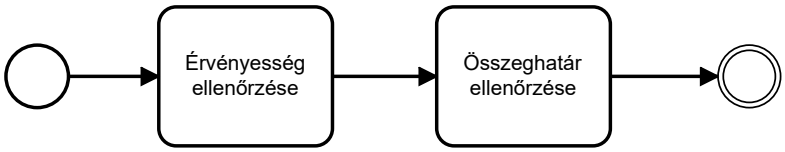


15. ábra. Folyamatlépés gyorsítótárból történő kiszolgálása.
Forrás: a szerzők saját szerkesztése

Itt fontos kiemelni, hogy a gyosítótár az informatikában számos fogalmat jelenthet, itt mi általánosságban azt feltételezzük, hogy folyamatok megvalósítása esetén tipikusan rendelkezése áll egy olyan adattárolási forma, amely korlátozott méretű, ugyanakkor akár nagyságrendekkel gyorsabb elérést tesz lehetővé.
A gyorsítótár méretezését illetően érdemes felidézni Zipf törvényét, amely kimondja, hogy egy adott adathalmaz elemeinek elérési gyakorisága fordított arányban áll az elemek népszerűségi rangsorban elfoglalt helyével. Példaként: a 2. legnépszerűbb elemet feleannyian kérdezik le, mint a legnépszerűbbet, stb.
Bár Zipf törvénye inkább tapasztalati ökölszabálynak fogható fel, és a 19. század folyamán angol természetes nyelvű szövegek elemzésekor született, a webes rendszerek dokumentumainak elérésére is sok esetben igaznak bizonyult.

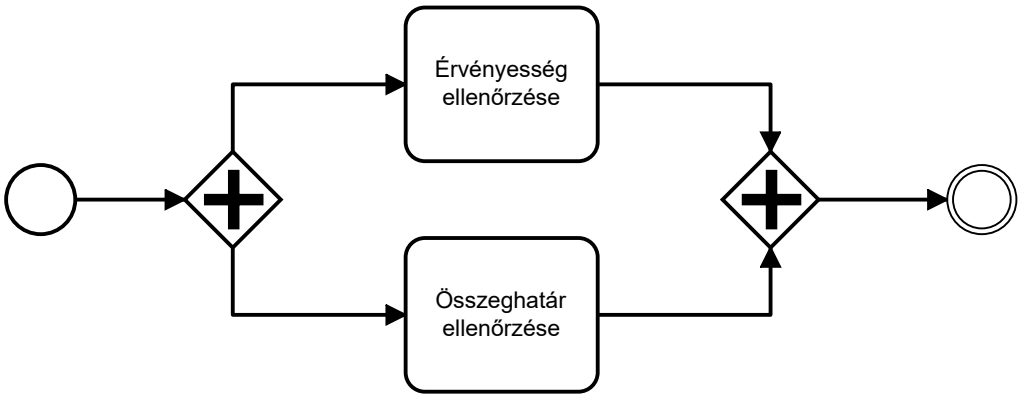
6.6.2. Logikailag független lépések párhuzamosítása

Folyamattervezés egyik alapkérdése az elemi lépések és sorrendi függőségek azonosítása után a lépések függetlenségének azonosítása (például a köztartozás-mentesség és a megfelelő előélet párhuzamos vizsgálata). Általában elmondható, hogy minden esetben érdemes azokat a lépéseket párhuzamos elágazásokba szervezni (Parallel Gateway vezérlőelem használatával), melyek közt sem direkt adatfüggőség nincsen (azaz egyik sem olvas olyan adatot, amelyet a másik módosít), sem vezérlésbeli kötöttség nem áll fenn (azaz a két feladat egymástól függetlenül is értelmezhető, tetszőleges sorrendben végrehajtható).
Az alábbi ábrákon példát mutatunk olyan folyamatrészletre, amely ugyanazokat a tevékenységeket először sorba szervezi, másodszor pedig párhuzamosítja. Megjegyzendő, hogy természetesen a párhuzamosítás haszna akkor jelentkezik, ha az egyes feladatok nem ugyanazon az erőforráspéldányon kerülnek elvégzésre (lásd következő minták).



16. ábra. Soros végrehatjásba szervezett lépések.
Forrás: a szerzők saját szerkesztése

A párhuzamos végrehajtás (amelynek programkód esetében kiterjedt szakirodalma van), természetesen igényelhet logikai tervezési megfontolásokat is, például az alábbi esetben gondoskodni kell arról, hogy az érvényesség és az összeghatár megállapítása egymástól függetlenül rögzíthető legyen. Ez technikailag megtehető többféleképpen is, vagy a folyamat által használt adatszerkezet és mentési technika módosításával, vagy akár a folyamatban értelmezett ún. lokális változók használatával, amelyek értékét csak az összes lépés lefutása után rögzítjük tartósan. Szintén fontos kiemelni, hogy a párhuzamosítással adott esetben veszthetünk is, például ha sok kérvényt adnak be érvénytelenül, akkor az alábbi vezérlési szerkezet eredményezheti azt is, hogy sok kérvény összeghatárnak való megfelelését feleslegesen fogjuk vizsgálni.



17. ábra. Párhuzamos végrehajtásba szervezett folyamatlépések.
Forrás: a szerzők saját szerkesztése

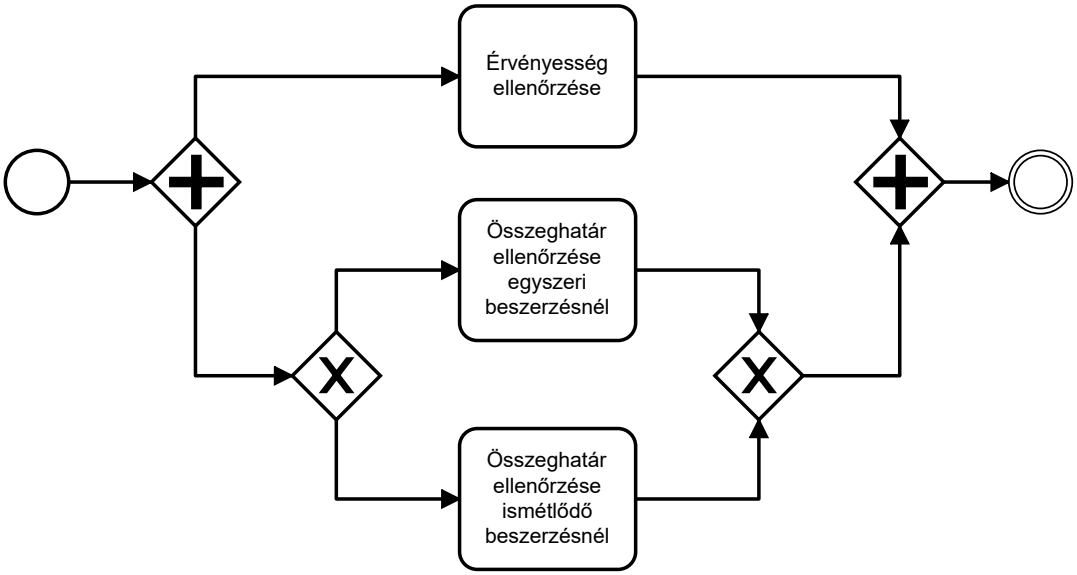
A párhuzamosítás ezek után azzal az előnnyel járhat, hogy csökken a felhasználó által érzékelt várható válaszidő.

6.6.3. Megosztott erőforrások használata

Az erőforrások tervezése során bevett egyik alapelv az általánosított erőforrások megosztott használata, amely voltaképp rokon az informatikában a felhőalapú számítási modellek eleterjedésével jelentkező megközelítéssel. Lényege, hogy specializált erőforrások helyett általános célúakat használunk, például a fenti példánál maradva, ha human erőforrások végzik az egyes feladatokat, akkor mindenkit felkészítünk mind az érvényesség, mind az összeghatár kezelésére, feltételezve, hogy a működést mindenki az összes lépés tekintetében ki tudja szolgálni. (Mivel a BPMN-diagramokon az erőforrások alapvetően nem látszanak, azért ez magán a folyamatmodellen nem okoz látható változást). Fontos megjegyezni, hogy ilyenkor is érvényesíthető az ún. négy szem elve, amikor egy adott példány esetében a két lépést két külön felhasználónak kell elvégeznie, így kerülhetők el a visszaélésből vagy tévedésből származó hibák.
Kapacitástervezés szempontjából ugyanakkor az általános erőforrások használata előnyös lehet, hiszen az egyes erőforrások kihasználtsága csökkenhet, a rendszer átbocsátóképessége megnő, illetve az egyes felhasználók várakozási ideje csökkenhet. Természetesen, mint minden tervezői döntésnél, itt sem csak pozitívumokkal jár a változás: a dedikált erőforrások adott esetben gyorsabban képesek elvégezni ugyanazt a feladatot, illetve így egy erőforráspéldány hibája akár több feladat elvégzése során vezethet hibás eredményhez.

6.6.4. Kritikus lépések erőforrás szintű leválasztása

A megosztott erőforrások használatához képest épp ellenkező eset, ha egy kritikus lépés végrehajtását erőforrás szinten is elválasztjuk a többitől. Ilyen lehet automatikusan végrehajtott lépések esetén például a riasztásgenerálás és az alacsony prioritású naplózási lépések külön erőforráson történő elvégzése akár annak az árán is, hogy ez a folyamat működése során lassulást okoz.
Ha azonban figyelembe vesszük a folyamat teljesítőképességét (performability) is, akkor a kritikus esetek számától függően az így tapasztalat kisebb hibaarány, kevesebb hibakezelési és ismétlési lépés miatt a várható végrehajtás (a folyamat válaszideje) akár rövidebb is lehet, mint ha nem különítjük el ezeket a feladatokat, tehát nemcsak a szogláltatásbiztonság, hanem a teljesítmény mérhető metrikái is javulnak. A feladatok elkülönítése történhet a már végrehajtott folyamatpéldányok adatainak elemzése alapján is, ez esetben azonosíthatók olyan lépések vagy lépésbemenet-párosítások, amelyek esetében érdemes a szétválasztást alkalmazni.
Az alábbi ábrán olyan kiegészítést mutatunk, amikor a párhuzamos végrehajtás bevezetése mellett a kiemelt feladatokat külön erőforráson kezeljük. Megjegyzés: léteznek olyan eszközök is, amelyeknél ez megtehető az adott lépéshez rendelt erőforrás hozzárendelési házirend (policy) definiálásával is, azonban szerencsésebb és karbantharthatóbb ezt már a modell szintjén jelezni, hiszen így mind a teljesítmény, mind a hibaterjedés szempontjbáól végzett vizsgálatok során explicit elkülöníthető az adott lépés.



18. ábra. Folyamatmodell-minta típus szerint különböző módon végrehajtott lépésekre. Forrás: a szerzők saját szerkesztése

7. ÖSSZEFOGLALÁS

Jelen tanulmány egy okos városi engedélyezési esettanulmány példáján keresztül bemutatta és áttekintette az alkalmazások minőségi követelményeinek kezelését azok definiálásán és mérhetővé tételén keresztül. Bemutattuk a teljesítmény és a hibatűrés mérési és kiértékelési szempontjait, valamint példát adtunk arra, hogy lehet már modell szintjén felkészíteni egy rendszert az ezeknek a szempontoknak történő megfelelésre. Annak érdekében, hogy a rendszer teljesítménye becsülhetővé váljon, elengedhetetlen a rendszer terhelésmodelljének felállítása. Példát adtunk arra, hogyan tehető ez meg mérési adatok alapján, illetve milyen főbb terhelésmodellek léteznek.

A rendszer szolgáltatásbiztonsági kiértékelésének egyik fontos eszköze a rendszerben fellépő esetleges hibák terjedésének elemzése, illetve annak vizsgálata, hogy milyen hibaokok vezethetnek rendszerszintű, a felhasználó által is észlelhető meghibásodáshoz. Fontos továbbá annak vizsgálata, mely komponens(ek) bővítendők ki a hibatűrő számítástechnikában ismert módszerek szerint annak érdekében, hogy a rendszer ellenállóbb legyen a hibákkal szemben (jelentse ez akár a hibák felismerését és jelzését, akár javítását).

Végül, amennyiben az elemzések azonosították a javítandó pontokat a rendszer működését leíró folyamatokban, akkor modellszinten is módosítható a rendszerterv a felhasználói követelményeknek történő jobb megfelelés érdekében, amit szintén példákkal illusztráltunk.

IRODALOMJEGYZÉK

Agrawala, Ashok K. – Mohr, Jeffrey M. – Bryant, Raymond M. (1976): An approach to the workload characterization problem. *Computer 9*, no. 6. p. 18–32.

Akers, Sheldon B. (1978): Binary decision diagrams. *IEEE Transactions on Computers 6*. p. 509–516.

Anscombe, Francis J. (1973): Graphs in Statistical Analysis. *The American Statistician*, no. 27:1. p. 17–21, DOI: 10.1080/00031305.1973.10478966

Arlitt, Martin – Jin, Tai (2000): A workload characterization study of the 1998 world cup web site. *IEEE Network 14*, no. 3. p. 30–37.

Barber, Scott (2004): Creating effective load models for performance testing with incomplete empirical data. In *Sixth IEEE International Workshop on Web Site Evolution, (WSE'04)*. p. 51–59. IEEE.

Barford, Paul – Nowak, Rob – Willett, Rebecca – Yegneswaran, Vinod (2006): Toward a model for source addresses of internet background radiation. In *Proc. of the 7th Passive and Active Measurement Conference*.

Basili, Victor R. – Rombach, H. Dieter (1988): The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering 14*, no. 6. p. 758–773.

Berman, Oded – Kumar, U. Dinesh (1999): Optimization models for recovery block schemes. *European Journal of Operational Research 115*, no. 2. p. 368–379.

Bondavalli, Andera – Simoncini, Luca (1990): Failure classification with respect to detection. *In Proceedings of 2nd IEEE Workshop on Future Trends of Distributed Computing Systems*. p. 47–53. IEEE.

Bonfiglio, Valentina – Brancati, Francesco – Rossi, Francesco – Bondavalli, Andrea – Montecchi, Leonardo – Pataricza, András – Kocsis, Imre – Molnár, Vince (2017): Composable Framework Support for Software-FMEA through Model Execution. In *Certifications of Critical Systems – The CECRIS Experience*, A. Bondavalli and Brancati, Francesco, Eds. River Publishers, Denmark.

Borgelt, Christian – Höppner, Frank – Klawonn, Frank (2010): *Guide to Intelligent Data Analysis*. Springer-Verlag London Ltd., London

Buck, Bryan – Hollingsworth, Jeffrey K. (2000): An API for runtime code patching. *The International Journal of High Performance Computing Applications 14*, no. 4. p. 317–329.

Burgess, Mark – Haugerud, Hårek – Straumsnes, Sigmund – Reitan, Trond (2002): Measuring system normality. *ACM Transactions on Computer Systems (TOCS) 20*, no. 2. p. 125–160.

Calabrese, Julieta – Muñoz, Rocío – Pasini, Ariel – Esponda, Silvia – Boracchia, Marcos – Pesado, Patricia (2017): Assistant for the Evaluation of Software Product Quality Characteristics Proposed by ISO/IEC 25010 Based on GQM-Defined Metrics. In *Argentine Congress of Computer Science*, pp. 164–175. Springer, Cham.

Casale, Giuliano – Mi, Ningfang – Cherkasova, Ludmila – Smirni, Evgenia (2012): Dealing with burstiness in multi-tier applications: Models and their parameterization. *IEEE Transactions on Software Engineering 38*, no. 5. p. 1040–1053.

Casale, Giuliano – Mi, Ningfang – Smirni, Evgenia (2010): Model-driven system capacity planning under workload burstiness. *IEEE Transactions on Computers 59*, no. 1. p. 66–80.

Chandola, Varun – Banerjee, Arindam – Kumar, Vipin (2009): Anomaly detection: A survey. *ACM Computing Surveys (CSUR) 41/3*. no. 15.

Chatfield, Chris (2002): Confessions of a pragmatic statistician. *Journal of the Royal Statistical Society: Series D (The Statistician) 51*, no. 1. p. 1–20.

Chen, J. Bradley – Endo, Yasuhiro – Chan, Kee – Mazieres, David – Dias, Antonio – Seltzer, Margo – Smith, Michael D. (1996): The measured performance of personal computer operating systems. *ACM Transactions on Computer Systems (TOCS) 14*, no. 1. p. 3–40.

Chen, Liming – Avizienis, Algirdas (1995): N-version programming: A fault-tolerance approach to reliability of software operatlon. In *25th International Symposium on Fault-Tolerant Computing*. p. 113. IEEE.

Cheng, Paul S. (1969): Trace-driven system modeling. *IBM Systems Journal 8*, no. 4. p. 280–289.

Cherkasova, Ludmila – Ozonat, Kivanc – Mi, Ningfang – Symons, Julie – Smirni, Evgenia (2008): Anomaly? Application change? Or workload change? Towards automated detection of application performance anomaly and change. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*. p. 452–461.

Chiang, Su-Hui – Vernon, Mary K. (2001): Characteristics of a large shared memory production workload. In *Workshop on Job Scheduling Strategies for Parallel Processing*. p. 159–187. Springer, Berlin–Heidelberg.

Cirne, Walfredo – Berman, Francine (2001): A comprehensive model of the supercomputer workload. In *IEEE International Workshop on Workload Characterization, 2001. WWC-4*. p. 140–148. IEEE.

Denning, Peter J. (2006): The locality principle. In *Communication Networks And Computer Systems: A Tribute to Professor Erol Gelenbe*. pp. 43–67.

Dixon, Ryan – Sherwood, Timothy (2008): Whiteboards that compute: A workload analysis. In *IEEE International Symposium on Workload Characterization, 2008. IISWC 2008*. p. 69–78. IEEE.

Downey, Allen B. – Feitelson, Dror G. (1999): The elusive goal of workload characterization. *ACM SIGMETRICS Performance Evaluation Review 26*, no. 4. p. 14–29.

Eeckhout, Lieven – De Bosschere, Koenraad – Neefs, Henk (2000): Performance analysis through synthetic trace generation. In *ISPASS*. p. 1–6. IEEE.

Eeckhout, Lieven – De Bosschere, Koenraad (2001): Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*. p. 25–34. IEEE.

Eeckhout, Lieven – Vandierendonck, Hans – De Bosschere, Koenraad (2003): Designing computer architecture research workloads. *Computer 36*, no. 2. p. 65–71.

Feitelson, Dror G. – Naaman, Michael (1999): Self-tuning systems*. IEEE Software 16*, no. 2. p. 52–60.

Feitelson, Dror G. – Tsafrir, Dan (2006): Workload sanitation for performance evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software*. p. 221–230. IEEE.

Feitelson, Dror G. (2002): Workload modeling for performance evaluation. In *IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation*. p. 114–141. Springer, Berlin–Heidelberg.

Feitelson, Dror G. (2015): *Workload modeling for computer systems performance evaluation*. Cambridge University Press, Cambridge.

Feldmann, Anja – Gilbert, Anna C. – Willinger, Walter – Kurtz, Tom G. (1998): The changing nature of network traffic: Scaling phenomena. *ACM SIGCOMM Computer Communication Review 28*, no. 2. p. 5–29.

Ferrari, Domenico (1972): Workload charaterization and selection in computer performance measurement. *Computer 5*, no. 4. p. 18–24.

Ferrari, Domenico (1984): On the foundations of artificial workload design. Vol. 12, no. 3. p. 8–14. *ACM*.

Finamore, Alessandro – Mellia, Marco – Meo, Michela – Munafo, Maurizio M. – Di Torino, Politecnico – Rossi, Dario (2011): Experiences of internet traffic monitoring with Tstat. *IEEE Network 25*, no. 3. p. 8–14.

Fonseca, Rodrigo – Almeida, Virgilio – Crovella, Mark – Abrahao, Bruno (2002): On the intrinsic locality properties of web reference streams. *In IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies* (IEEE Cat. No. 03CH37428) Vol. 1., p. 448-458., IEEE.

Fonseca, Rodrigo – Almeida, Virgílio – Crovella, Mark (2005): Locality in a web of streams. *Communications of the ACM 48*, no. 1. p. 82–88.

Gamma, Erich (1995): Design patterns: elements of reusable object-oriented software. Pearson Education, India

Ganapathi, Archana – Chen, Yanpei – Fox, Armando – Katz, Randy – Patterson, David (2010): Statistics-driven workload modeling for the cloud. *In IEEE 26th International Conference on Data Engineering Workshops, ICDEW*. p. 87–92.

García-Dorado, José Luis – Finamore, Alessandro – Mellia, Marco – Meo, Michela – Munafo, Maurizio (2012): Characterization of ISP traffic: Trends, user habits, and access technology impact. *IEEE Transactions on Network and Service Management* 9, no. 2. p. 142–155.

Gil-Garcia, J. Ramon – Pardo, Theresa A. – Nam, Taewoo (2015): What makes a city smart? Identifying core components and proposing an integrative and comprehensive conceptualization. *Information Polity 20*, no. 1. p. 61–87.

Gönczy László (2018): Kritikus infrastruktúrák és szolgáltatások modellezése. *BME-NKE Okosváros kutatóműhely*, Kismonográfia. Megjelenés alatt.

Gönczy, László – Hegedüs, Ábel – Varró, Dániel (2011): Methodologies for model-driven development and deployment: An overview. In *Rigorous software engineering for service-oriented systems*, p. 541–560. Springer, Berlin–Heidelberg.

Gribble, Steven D. – Brewer, Eric A. (1997): System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *USENIX Symposium on Internet Technologies and Systems*.

Hanmer, Robert (2013): Patterns for fault tolerant software. John Wiley & Sons, Chichester

Henderson, Tristan – Kotz, David – Abyzov, Ilya (2008): The changing usage of a mature campus-wide wireless network. *Computer Networks 52*, no. 14. p. 2690–2712.

Hernández-Campos, Félix – Jeffay, Kevin – Smith, F. Donelson (2003): Tracking the evolution of web traffic: 1995-2003. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems. MASCOTS* 2003. p. 16–25. IEEE.

Hollingsworth, Jeffrey K. – Miller, Barton Paul – Cargille, Jon (1994): Dynamic program instrumentation for scalable performance tools. In *Proceedings of the Scalable High-Performance Computing Conference*. p. 841–850. IEEE.

Hotovy, Steven (1996): Workload evolution on the Cornell Theory Center IBM SP2. In *Workshop on Job Scheduling Strategies for Parallel Processing*. p. 27–40. Springer, Berlin–Heidelberg.

Huppler, Karl (2009): The art of building a good benchmark. In Technology Conference on Performance Evaluation and Benchmarking. pp. 18–30. Springer, Berlin–Heidelberg.

IBM (2006): *An architectural blueprint for autonomic computing*. IBM White Paper 31., IBM Corporation, Hawthorne, NY

IEEE Computer Society. Software Engineering Technical Committee (1993): *IEEE Standard for a Software Quality Metrics Methodolog*y. Institute of Electrical and Electronics Engineering.

ISO/IEC 25012:2008 (2008): *Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Data quality model*. ISO.

ISO-IEC 25010:2011 (2011): *Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models*. ISO.

ISO-IEC TS 25011:2017 (2017): *Information Technology - Systems and Software Quality Requirements and Evaluation (SQuaRE) - Service Quality Models*. ISO.

Jakab László – Kocsis Imre – Gönczy László (2018a): *Kvalitatív módszerek és elemzési megközelítések felhasználása kritikus infrastruktúrák védelmében és teljesítménymenedzsmentjében I. rész.* BME-NKE Okosváros kutatóműhely. Kismonográfia. Megjelenés alatt.

Jakab László – Kocsis Imre – Gönczy László (2018b): *Kvalitatív módszerek és elemzési megközelítések felhasználása kritikus infrastruktúrák védelmében és teljesítménymenedzsmentjében II. rész.* BME-NKE Okosváros kutatóműhely. Kismonográfia. Megjelenés alatt.

Kessler, Richard E. – Hill, Mark D. – Wood, David A. (1994): A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers 43*, no. 6. p. 664–675.

Kocsis, Imre (2018): Design for Dependability Through Error Propagation Space Exploration. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. p. 172–178. IEEE.

Koldinger, Eric J. – Eggers, Susan J. – Levy, Henry M. (1991): On the validity of trace-driven simulation for multiprocessors. In *ACM SIGARCH Computer Architecture News*, vol. 19, no. 3, pp. 244–253. ACM.

Kotsis, Gabriele (1997): A systematic approach for workload modeling for parallel processing systems. *Parallel Computing 22*, no. 13. p. 1771–1787.

Kurmas, Zachary – Keeton, Kimberly – Mackenzie, Kenneth (2003): Synthesizing representative I/O workloads using iterative distillation. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems. MASCOTS 2003*. p. 6–15. IEEE.

Law, Averill M. – Kelton, W. David (1991): *Simulation modeling and analysis*. Vol. 2., McGraw-Hill, New York

Leland, Will E. – Willinger, Walter – Taqqu, Murad S. – Wilson, Daniel V. (1995): On the self-similar nature of Ethernet traffic. *ACM SIGCOMM Computer Communication Review 25*, no. 1. p. 202–213.

Maier, Gregor – Schneider, Fabian – Feldmann, Anja (2011): NAT usage in residential broadband networks. In *International Conference on Passive and Active Network Measurement*. pp. 32-41. Springer, Berlin–Heidelberg.

Malony, Allen D. – Reed, Daniel A. – Wijshoff, Harry A. G. (1992): Performance measurement intrusion and perturbation analysis. *IEEE Transactions on Parallel & Distributed Systems 4*. p. 433–450.

Menasce, Daniel A. – Almeida, Virgilio A.F. – Dowdy, Lawrence W. – Dowdy, Larry (2004): *Performance by design: computer capacity planning by example*. Prentice Hall Professional, Upper Saddle River, New Jersey

Molnár, Vince – Majzik, István (2017): Constraint programming with multi-valued decision diagrams: a saturation approach. In *Proceedings of the 24th PhD Mini-Symposium*, B. Pataki, Ed. Budapest: BME MIT. p. 54–57.

Nieuwejaar, Nils – Kotz, David – Purakayastha, Apratim – Ellis, C. Sclatter – Best, Michael L. (1996): File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems 7*, no. 10. p. 1075–1089.

Nurmi, Daniel – Brevik, John – Wolski, Rich (2007): QBETS: queue bounds estimation from time series. In *Workshop on Job Scheduling Strategies for Parallel Processing*. p. 76–101. Springer, Berlin–Heidelberg.

Pataricza András szerk. (2006): *Formális módszerek az informatikában*. Typotex, Budapest

Pataricza, András (2008a): *Model-based dependability analysis*. DSc Thesis. Hungarian Academy of Sciences.

Pataricza, András (2008b): Systematic generation of dependability cases from functional models. In *Proc. of Formal Methods for Automation and Safety in Railway and Automotive Systems. Symposium FORMS/FORMAT*, October. pp. 9–10.

Rajkumar, Ragunathan – Lee, Insup – Sha, Lui – Stankovic, John (2010): Cyber-physical systems: the next computing revolution. In *47th ACM/IEEE Design Automation Conference (DAC)*. p. 731–736. IEEE.

Randell, Brian – Xu, Jie (1995): The evolution of the recovery block concept. *Software Fault Tolerance 3*. p. 1–22.

Rosenstein, Mark (2000): What is actually taking place on web sites: e-commerce lessons from web server logs. In *Proceedings of the 2nd ACM conference on Electronic commerce*. pp. 38–43. ACM.

Schneider, Fabian – Ager, Bernhard – Maier, Gregor – Feldmann, Anja – Uhlig, Steve (2012): Pitfalls in HTTP traffic measurements and analysis. In *Passive and Active Measurement.* p. 242-251. Springer, Berlin–Heidelberg.

Sedgewick, Robert – Wayne, Kevin (2011): *Algorithms*. Addison-Wesley Professional, Boston

Sherman, Stephen – Baskett, Forest – Browne, James C. (1972): Trace-driven modeling and analysis of CPU scheduling in a multiprogramming system. *Communications of the ACM 15*, no. 12. p. 1063–1069.

Shmueli, Edi – Feitelson, Dror G. (2006): Using site-level modeling to evaluate the performance of parallel system schedulers. In *14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2006*. p. 167–178. IEEE.

Singh, Ajay – Segall, Zary (1982): Synthetic Workload Generation for Experimentation with Multiprocessors. In *ICDCS*. p. 778–785.

Smith, Alan Jay (1982): Cache memories. *ACM Computing Surveys (CSUR) 14*, no. 3. p. 473–530.

Smith, Keith A. – Seltzer, Margo I. (1997): File system aging—increasing the relevance of file system benchmarks. In *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, no. 1. pp. 203–213. ACM.

Sprunt, Brinkley (2002a): The basics of performance-monitoring hardware. *IEEE Micro 4.* p. 64–71.

Sprunt, Brinkley (2002b): Pentium 4 performance-monitoring features. *IEEE Micro 4.* p. 72–82.

Sreenivasan, Katepalli R. – Kleinman, A. J. (1974): On the construction of a representative synthetic workload. *Communications of the ACM 17*, no. 3. p. 127–133.

Stolfo, Salvatore J. – Hershkop, Shlomo – Hu, Chia-Wei – Li, Wei-Jen – Nimeskern, Olivier – Wang, Ke (2006): Behavior-based modeling and its application to email analysis. *ACM Transactions on Internet Technology (TOIT) 6*, no. 2. p. 187–221.

Szeredi Péter – Benkő Tamás (2002): *Nagyhatékonyságú logikai programozás*. Kézirat, Budapest, Magyarország.

Talby, David – Feitelson, Dror G. (2005): Improving and stabilizing parallel computer performance using adaptive backfilling. In *International Parallel and Distributed Processing Symposium*, p. 84a. IEEE.

Trivedi, Kishor S. – Vaidyanathan, Kalyanaraman (2002): Software reliability and rejuvenation: Modeling and analysis. In *IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation*. pp. 318–345. Springer, Berlin–Heidelberg.

Tsai, Jeffrey J. P. – Fang, K-Y. – Chen, H-Y. (1990): A noninvasive architecture to monitor real-time distributed systems. *Computer 23*, no. 3. p. 11–23.

Tufte, Edward R. – Goeler, Nora Hillman – Benson, Richard (1990): *Envisioning information*. Vol. 126., Graphics Press, Cheshire, CT, USA.

Tufte, Edward R. – McKay, Susan R. – Christian, Wolfgang – Matey, James R. (1998): *Visual explanations: images and quantities, evidence and narrative*. Graphics Press, Cheshire, CT, USA.

Tufte, Edward R. (2001): *The visual display of quantitative information*. Vol. 2., Graphics Press, Cheshire, CT, USA.

Tukey, John W. (1977): *Exploratory data analysis*. Vol. 2. Addison-Wesley Publishing Company, Boston

Uhlig, Richard A. – Mudge, Trevor N. (1997): Trace-driven memory simulation: A survey. *ACM Computing Surveys (CSUR) 29*, no. 2. p. 128–170.

Urbanics, Gábor – Gönczy, László – Urbán, Balázs – Hartwig, János – Kocsis, Imre (2014): Combined Error Propagation Analysis and Runtime Event Detection in Process-Driven Systems. In *International Workshop on Software Engineering for Resilient Systems*. p. 169–183. Springer, Cham.

Vieira, Kleber – Schulter, Alexandre – Westphall, Carlos – Westphall, Carla (2010): Intrusion detection techniques in grid and cloud computing environment. *IT Professional, IEEE Computer Society 12*, no. 4. p. 38–43.

Walter, Edward S. – Wallace, Victor L. (1967): Further analysis of a computing center environment. *Communications of the ACM 10*, no. 5. p. 266–272.

Weicker, Reinhold P. (1991): A detailed look at some popular benchmarks. *Parallel Computing 17*, no. 10–11. p. 1153–1172.

Weinreich, Harald – Obendorf, Hartmut – Herder, Eelco (2006): Data cleaning methods for client and proxy logs. In *Proceedings of WWW'06 Workshop on Logging Traces of Web Activity: The Mechanics of Data Collection*.

Yeh, Tse-Yu – Patt, Yale N. (1991): Two-level adaptive training branch prediction. In *Proceedings of the 24th annual international symposium on Microarchitecture*, pp. 51–61. ACM.

Zhang, Li – Liu, Zhen – Riabov, Anton – Schulman, Monty – Xia, Cathy – Zhang, Fan (2003): A comprehensive toolset for workload characterization, performance modeling, and online control. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. p. 63–77. Springer, Berlin–Heidelberg.

Zhang, Qi – Riska, Alma – Sun, Wei – Smirni, Evgenia – Ciardo, Gianfranco (2005): Workload-aware load balancing for clustered web servers. *IEEE Transactions on Parallel and Distributed Systems 16*, no. 3. p. 219–233.

Zhao, Yi – Gong, Jiayu – Hu, Yun – Liu, Zhenyu – Cai, Lizhi (2017): Analysis of quality evaluation based on ISO/IEC SQuaRE series standards and its considerations. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*. pp. 245–250. IEEE.

Zilber, Julia – Amit, Ofer – Talby, David (2005): What is worth learning from parallel workloads? A user and session based analysis. In *Proceedings of the 19th Annual International Conference on Supercomputing*. p. 377–386. ACM.

A Nemzeti Közszolgálati Egyetem kiadványa

Kiadó:

Nemzeti Közszolgálati Egyetem
Közigazgatási Továbbképzési Intézet

www.uni-nke.hu

Felelős kiadó:
Prof. Dr. Kis Norbert rektorhelyettes
Címe: 1083 Budapest, Üllői út 82.

Kiadói szerkesztő:
Dorogi Katalin

Tördelőszerkesztő:
Friebert Máté

ISBN 978-963-498-398-9 (PDF)