Miskolci Egyetem

GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

Irányított gráfokkal leírt elosztott alkalmazások helyességének biztosítása csoportmunka környezetekben

Ph.D. értekezés

Készítette:
Sipos Gergely

Hatvany József Informatikai Tudományok Doktori Iskola

Magyar Tudományos Akadémia
Számítástechnikai és Automatizálási Kutató Intézet
(MTA SZTAKI)

Doktori iskola vezetője:
Dr. Tóth Tibor

Tudományos vezető:
Dr. Kacsuk Péter

2010

Köszönetnyilvánítás

Elsődlegesen köszönettel tartozom Dr. Kacsuk Péternek, aki témavezetőként és kollégaként bevezetett a kutatói világba. Motivált és támogatott új ötleteim kidolgozásában, segített annak az iránynak a megtalálásában, amely végül a disszertációig vezetett. Számtalan projekt létrehozásával lehetővé tette számomra a hazai és külföldi közegben való fejlődést és megmérettetést, informatikai tudásom bővítése mellett a kutatásmenedzsmentben és a csoportvezetésben való kiteljesedést.

Köszönettel tartozom az MTA SZTAKI „Párhuzamos és Elosztott Rendszerek Laboratórium" és a University of Reading „Centre for Advanced Computing and Emerging Technologies" dolgozóinak, akik munkájukkal, javaslataikkal, kérdéseikkel hozzájárultak eredményeim, cikkeim és a disszertációm javításához. Külön köszönet Németh Zsoltnak és Kovács Józsefnek, akik számtalan gyakorlati tanáccsal láttak el a dolgozat írása közben.

A családom támogatása nélkül nem érhettem volna el idáig. Szívből köszönöm Édesanyámnak és Édesapámnak, hogy lehetővé tették számomra a tanulást, biztos hátteret nyújtva annyi éven át. Hálával tartozom menyasszonyomnak, Erikának és öcsémnek, Ádámnak, akik bíztattak és bíztak bennem. Segítségükkel megtanultam, hogy egyszer minden befejezhető.

Amszterdam, 2010. július 18.

Sipos Gergely

Összefoglalás

A számítógépes csoportmunka támogatása az 1980-as évek közepén az asztali számítógépek és számítógépes hálózatok elterjedésével egyidőben kezdődött. Az első csoportmunka-alkalmazások elektronikus naptárak voltak, melyekkel könnyebben lehetett közös találkozókhoz mindenkinek megfelelő időpontot választani. A csoportmunka támogatás ma a fejlesztő- és szerkesztőrendszerekre a legjellemzőbb. Ezekkel az eszközökkel egyazon fájlt (pl. rajzot, táblázatot, dokumentumot) tud egyidőben több személy módosítani.

Az üzleti-ipari, valamint tudományos folyamatok leírásának és automatizálásának egyik legszélesebb körben használt eszköze a munkafolyam, angolul workflow. A workflow egy irányított gráfként felfogható folyamatleírás, melyben a gráf csomópontjai elemi feladatokat, az élek pedig a lépések közötti információs és időbeli függőségeket definiálnak. Egy workflow leírás létrehozása idő és munkaigényes folyamat, különösen ha több ember bevonására van szükség. Bonyolult workflow alkalmazások összeállítása csoportmunkát támogató környezetekkel lenne hatékony.

A dolgozat módszereket ismertet irányított gráfokként felfogható workflow alkalmazások több személy általi szerkesztésének lehetőségére. A megoldásokkal folyamat-fejlesztő csoportok képesek közösen, kis erőfeszítéssel új leírásokat létrehozni, majd végrehajtani. Az ismertetésre kerülő módszerek zárolással biztosítják a hozzáférést több ember számára ugyanazon workflow leírás különböző részeihez úgy, hogy eközben a gráf tartalmi és formai helyessége garantált. A helyesség megőrzése kritikus fontosságú ahhoz, hogy a gráf alapján az üzleti-ipari, vagy tudományos folyamat valóban végrehajtható legyen.

Tartalomjegyzék

Gyakran előforduló jelölések listája

A	Egy gráf csomópont. A csomópontok nagybetűvel jelöltek.
ab	Egy gráf *A* csomópontjából a *B* csomópontjába futó irányított *él*. Az élek kis betűvel jelöltek és a csomópontok betűire utalnak.
[*ab*]	Egy gráf *A* csomópontjából a *B* csomópontjába futó irányított *út*. Az út egy vagy több irányított él egymásutánja.
m(*A*) m(*ab*)	Egy gráf *A* csomópontjának, illetve *ab* élének módosítását végző utasítás. Az utasítás hatására a csomópont/él egy vagy több paraméterének értéke megváltozik.
d(*A*) d(*ab*)	Egy gráf *A* csomópontjának, illetve *ab* élének törlését végző utasítás. Az utasítás hatására a csomópont/él törlődik a felhasználó saját gráf nézetéből.
a(*A*) a(*ab*)	Egy gráf *A* csomópontjának, illetve *ab* élének törlését végző utasítás. Az utasítás hatására a csomópont/él törlődik a felhasználó saját gráf nézetéből.
T_i	Egy *i* felhasználó workflow módosítására irányuló szerkesztési tranzakciója. Egy szerkesztési tranzakció gráf komponens módosítási, törlési és hozzáadási utasítások sorozata.
R_i	Egy *i* felhasználó által szerkeszteni kívánt gráf komponensek (csomópontok és élek) halmaza.
G_i *vagy* U_i	Egy *i* felhasználó számára kizárólagos zárolással (U lock) lefoglalt részgráf. Ahol szükséges hangsúlyozni a „User lock” elnevezést, ott az U_i jelölést használom.
S_i	Egy *i* felhasználó számára *nem* kizárólagos zárolással (System lock) lefoglalt részgráf.
A*x*	Gráf szerkesztési igényt elbírálni, zárolandó részgráfot meghatározni képes algoritmus *x*. verziója. (A dolgozat 6 különböző algoritmust tárgyal A1-A6 néven.)
P1	A 3.1 fejezetben bevezetett, szabad élek kialakulása ellen védelmet nyújtó protokoll.
T^S	Egyazon gráfot ugyanazon szerkesztési szakasz során szerkeszteni kívánó tranzakciók sorozata, az ún. *tranzakciósor*.

A dolgozat új tudományos eredményei

1. Tézis Workflow alkalmazások kollaboratív szerkesztése:
Kapcsolódó fejezetek: 2. és 3.

1.1. Altézis Összegyűjtöttem a workflow alapú alkalmazások valós idejű, kollaboratív fejlesztésével szemben támasztott felhasználói követelményeket és megmutattam, hogy azok zár alapú konkurenciakezelést használó, replikált architektúrájú rendszerrel mind kielégíthetők.

1.2. Altézis Megalkottam három olyan algoritmust, melye k zárak segítségével képesek biztosítani workflow alkalmazások szerkesztési céllal történő, több felhasználó közötti egyidejű felosztását. Bebizonyítottam, hogy mindhárom algoritmus képes garantálni a workflow gráfok konzisztenciáját a fejlesztők munkájának eldobása vagy utólagos kompenzációja nélkül.

2. Tézis Workflow-n alapuló csoportmunka hatékonysága:
Kapcsolódó fejezet: 4.

2.1. Altézis Kidolgoztam egy mérési módszert és kapcsolódó metrikákat a kollaboratív workflow alkalmazások feldarabolását végző algoritmusok hatékonyságának vizsgálatára. Megmutattam, hogy a mérési módszerrel bármilyen gráfon végbemenő tetszőleges kollaboratív szerkesztési eset kiértékelhető. A módszer segítségével megállapítható, hogy több zároló algoritmus közül melyik enged egyidőben nagyobb számú felhasználót dolgozni egy workflow-n, illetve hogy az algoritmusok mennyire részesítik egyenlő elbánásban az egymás után érkező fejlesztőket.

2.2. Altézis A 2.1. Altézisben bevezetett mérési módszer segítségével megállapítottam az 1.2. Altézisben kidolgozott három algoritmusról, hogy melyik milyen szerkesztési szituációban eredményezi a leghatékonyabb csoportmunkát. A különböző szerkesztési esetekre meghatároztam, hogy azokban melyik algoritmussal érhető el a legnagyobb szerkesztésbeli párhuzamosság.

3. Tézis Maximális hatékonyságú csoportmunka elérése:
Kapcsolódó fejezet: 5.

3.1. Altézis A 2.2. Altézis megállapításai alapján kidolgoztam az 1.2. Altézis három algoritmusánál hatékonyabb további két olyan algoritmust, amelyek szintén csak egyféle zárat tesznek a felhasználók számára láthatóvá. A második algoritmusról beláttam, hogy hatékonysága csak további zár típusok bevezetése árán növelhető, ez a lépés viszont bonyolultabbá tenné a kollaboratív fejlesztőrendszer használatát.

3.2. Altézis Definiáltam egy zárolási módszert, amely képes akkor is részgráfot allokálni, ha a felhasználó által fejlesztésre kért részgráf konkurens munka miatt nem foglalható le. A módszer képes a lehető legnagyobb ilyen alternatív részgráfot kiválasztani, így biztosítva, hogy a felhasználó a gráfon tervezett módosításainak többségét el tudja végezni.

1. Bevezetés és motivációk

1.1. Workflow alkalmazások és rendszerek

A mai gyorsan fejlődő, gyorsan változó üzleti és ipari világban egy vállalat sikerének legfontosabb záloga, ha minél alacsonyabb költséggel képes rutinszerű feladatait ellátni, illetve ha a lehető leggyorsabban és leghatékonyabban tud a piac új kihívásaira reagálni, új szolgáltatásokkal és termékekkel előállni. Ennek érdekében a vállalatoknak folyamatosan felülbírálni, javítani, optimalizálni kell a saját belső, illetve őket a környezetbe ágyazó külső folyamataikat (Georgakopoulos et al, 1995). Az ipari-üzleti folyamatok formális leírása, optimalizálása egyidős az iparosodással. A kezdetekben kizárólag az anyagi, majd később egyre inkább az információs folyamatok leírása, vállalatokon belüli, és vállalatokon átívelő egységesítése vált fontossá (Medina-Mora et al, 1993). Ezeknek a leírásoknak a célja először a minőségjavítás, a megismételhetőség, az egységesítés volt. A gyakran felmerülő folyamatok beazonosítása, feladatokra, majd részfeladatokra bontása lehetővé tette a rutinszerű lépések azonos minőségű, egyre alacsonyabb költség melletti, egyre hatékonyabb megvalósítását.

A számítástechnika az üzleti-ipari folyamatok (business process) nagymértékű automatizálását tette lehetővé. A számítógéppel segített üzleti folyamatokhoz kapcsolódó kutatások és fejlesztések az 1980-as években kezdődtek. Kezdetekben a folyamatok lépéseit adó tevékenységek elsősorban emberi erőforrásokat használtak, azaz a folyamatokat alkotó „munkák” (task-ok) emberek által elvégzett aktivitások voltak. A számítógépes alkalmazások és sztenderdizált szolgáltatások palettájának szélesedésével a 90-es évektől kezdődően azonban egyre elterjedtebbé vált számítógépes programok üzleti folyamatokba való bekapcsolása. Ekkor, az 1980-as évek vége, 1990-es évek eleje körül kezdték *a számítógéppel segített és automatizált ipari-üzleti folyamatokat workflow-knak*[1] *hívni* (Georgakopoulos et al, 1995)(Workflow, 2010). Kezdetekben a vállalatok saját fejlesztésű, csak házon belül alkalmazható alkalmazásokat és ezeket használó workflow-kat alkottak. Később azonban természetes igényként merült fel a vállalatok között átívelő munkafolyamatok leírása és automatizálása. Ez szabványok kidolgozását, a workflow-kat leíró és kezelő rendszerek egységesítését kívánta. A szabványok kidolgozására jött létre 1993-ban a mára több mint 300 vállalatot tömörítő Workflow Management Coalition (WfMC)(WfMC, 1993).

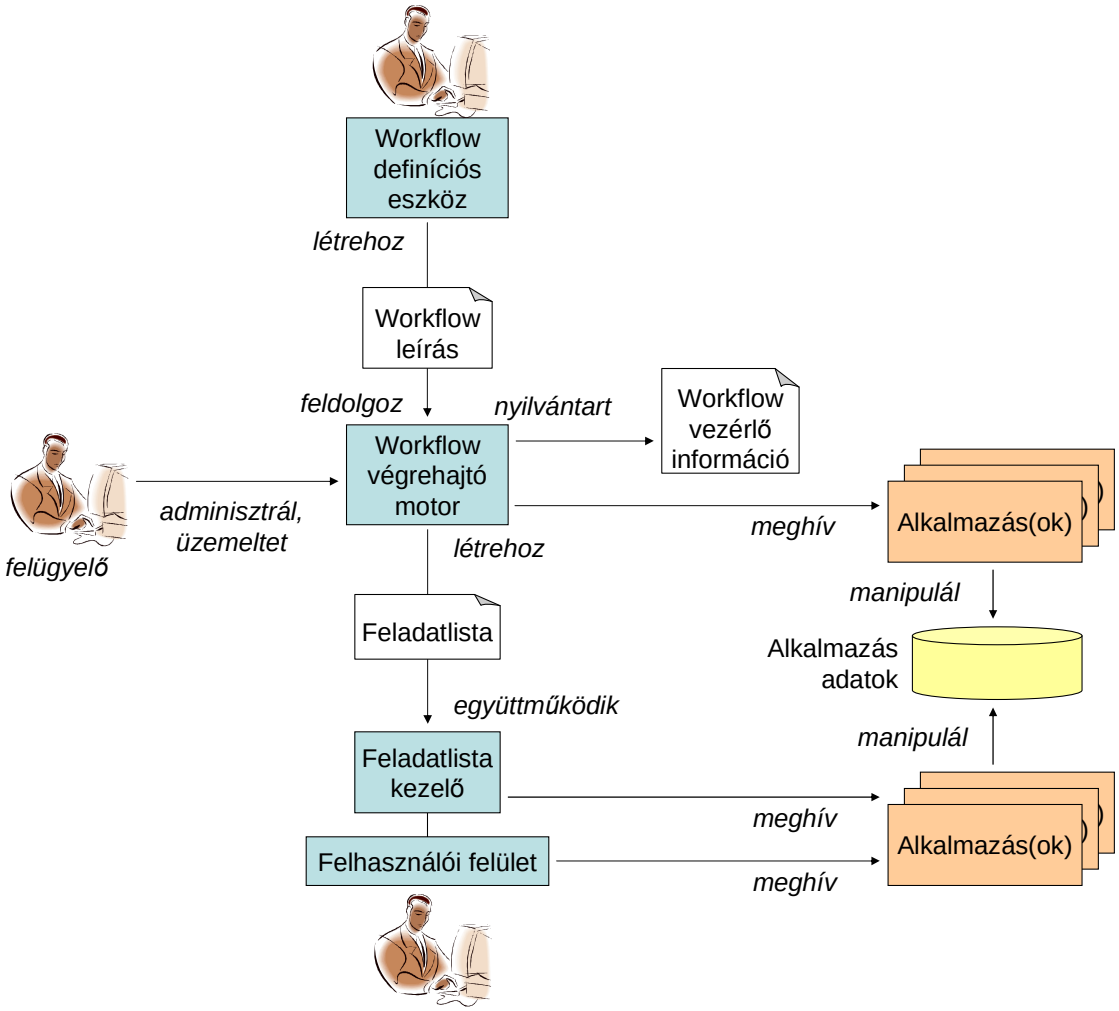
A WfMC 1995-ben adta ki a Workflow Referencia Modellt (WfRM, 1995), azt a dokumentumot, amely részletezi egy workflow rendszert felépítő általános elemeket, ezáltal segítve elő az egymással összekapcsolható, együttműködésre képes workflow megoldások kiépítését. A referencia modellben szereplő egyik magas szintű specifikáció szerint (ld. még 1-1. ábra) egy workflow rendszer valamilyen workflow definíciós eszközből (workflow definition tool), workflow végrehajtó motorból (workflow management engine), továbbá olyan interfészekből és komponensekből áll, amelyekkel a workflow lépéseit elvégző felhasználók és számítógépes programok a rendszerhez kapcsolódhatnak.

A workflow maga egy irányított gráfként fogható fel, melyben a csomópontok üzleti-ipari workflow-k esetén elvégzendő feladatokat reprezentálnak, az élek pedig valamilyen feladatok közötti függőségeket jelentenek – például dokumentumokon alapuló információs függőség, vagy egyszerű előidejűség. A workflow leírások többnyire grafikus szerkesztőrendszerrel készülnek.

Egy workflow leírás alapján maga az üzleti folyamatot egy végrehajtó motor valósítja meg. A motor feladata a workflow által tartalmazott elemi feladatokhoz megtalálni a megfelelő

[1] Ugyan egyes magyar munkákban a „workflow” angol kifejezés „munkafolyam” módon van fordítva, a dolgozat marad az eredeti angol elnevezésnél, annak szélesebb körben való ismertsége és elfogadottsága miatt.

végrehajtót – a megfelelő személyt, vagy a megfelelő szoftverszolgáltatást – biztosítani a munka elvégzéséhez szükséges adatokat, figyelni a munka alakulását, a munkák közötti függőségeket, és egy-egy munka elvégzése után az eredményt továbbítani a következő munkafázisok dolgozóinak.



1-1. ábra: Egy workflow rendszer főbb elemei a Workflow Management Coalition referencia modellje szerint. (WfRM, 1995) alapján.

A 90-es évek óta eltelt közel 20 év alatt a workflow alkalmazások az üzleti-ipari világ mellett más körökben is elterjedté váltak, és emiatt a *maga a workflow fogalom erősen túlterhelt lett. Különböző alkalmazói csoportok más-más fogalmat értenek workflow alatt.*

Például ma számos tudományos terület használ workflow-kat a kutatási folyamatok automatizálására (Taylor et al, 2006). Különösen jellemző ez a tudomány nagy adathalmazokkal dolgozó, számítógépes szimulációs megoldásokra erőteljesen építő ágaiban, az úgynevezett e-tudományokban (e-Science). A részecskefizika (Deelman et al, 2003), bioinformatika (Oinn et al, 2004), számítógépes kémia (Sudholt et al, 2006) vagy a földtudományok (Yilmaz et al, 2001) mind-mind használnak workflow alkalmazásokat. A tudományos workflow-k számítógépekkel segített tudományos folyamatokat automatizálnak. Egy tudományos workflow általában valamilyen adatgyűjtő vagy adatgeneráló lépések, adatfeldolgozó lépések és eredmény kiértékelő lépések egymásutánjából áll. A tudományos workflow-k döntő többsége automatizált, vagyis

számítógépes programok által ellátott munkaelemeket tartalmaz. A manuális lépések és emiatt az emberi erőforrások használata a tudományos workflow-kban elenyésző. Tudományos workflow-kat támogató technológiák a Web Szolgáltatások és a Grid (Fox, Gannon, 2006). A Web Szolgáltatások meglévő algoritmusok és alkalmazások szabványosított szolgáltatásként való közzétételét és workflow munkafolyamatokba való integrálását teszik lehetővé. A Grid különböző intézmények, és felhasználók által birtokolt számítógépes erőforrások (elsősorban processzorok, tárhelyek) megosztását, illetve workflow-kban való felhasználását teszi lehetővé (Foster, Kesselman, 1998). A Web Szolgáltatások és a Grid együtt olyan teljesen automatizált workflow-k létrehozását teszik lehetővé, amelyek különböző intézmények, kutatóhelyek által üzemeltetett szoftverszolgáltatásokat használnak, és amelyek futás közben képesek nagyszámú erőforráson szétterülni, ezáltal skálázható megoldást adni a felhasználók számának, és a feldolgozandó adatok mennyiségének fluktuációja esetén is.

A számítógéppel segített munka témakörében szintén gyakran használt fogalom a workflow. Itt azonban elsősorban több személy vagy csoport munkájának koordinálására szolgáló eszközt, egy *számítógéppel segített munkaszervezési módszert* értenek alatta (Winograd, 2004). Ilyen kontextusban elsősorban a munkát elvégző személyek, és nem az elvégzendő munka van a középpontban, és emiatt a workflow csomópontok döntő többsége az együttműködő személyeket vagy csoportokat reprezentálja.

Értekezésemben a workflow fogalma alatt irányított gráf segítségével leírt munkafolyamatot értek. Az, hogy a munkafolyamat üzleti-ipari, tudományos, munkaszervezési vagy bármilyen egyéb célból születik-e, a dolgozat szempontjából mellékes. A workflow számomra egy irányított gráf, melyet létre kell hozni, majd később egy workflow végrehajtó motor segítségével „le kell futtatni”. A 3-3. ábra mutat néhány ilyen irányított gráfként ábrázolt workflow-t. A workflow fejlesztés ebben a megközelítésben egy „model-driven engineering” módszer, amely folyamatok absztrakciókkal való leírását segíti (Schmidt, 2006). Ezek az absztrakciók közelebb állnak a létrehozóhoz (a workflow fejlesztőhöz), mint a belőlük generált és ténylegesen futtatásra kerülő algoritmikus kódokhoz. A futásra kerülő kódok a workflow definíció alapján a workflow futtató motorban, illetve az általa meghívott, elemi lépéseket elvégző szolgáltatásokban és alkalmazásokban jönnek létre. *A workflow-k emberközelibb absztrakciók által képesek a felhasználók széles körét bevonni, különböző tudású csoportok együttműködésében közös nyelvként funkcionálni.*

Bármilyen alkalmazási területet is tekintünk, egységesen igaz, hogy a workflow-k egyre bonyolultabb felépítésűek, egyre több tudást akkumulálnak, egyre összetettebb gráfokkal fogalmazhatók meg. Ez egyrészt a workflow alkalmazások méretén, másrészt a minőségén is látszik (Kamath et al, 1996)(Cummings et al, 2008). Az ilyen alkalmazások létrehozása azonban hagyományos, egyetlen felhasználót támogató workflow definíciós eszközökkel rendkívül időigényes és sok hibalehetőséget ad. *Workflow-k hatékony fejlesztéshez csoportmunka támogatásra van szükség.*

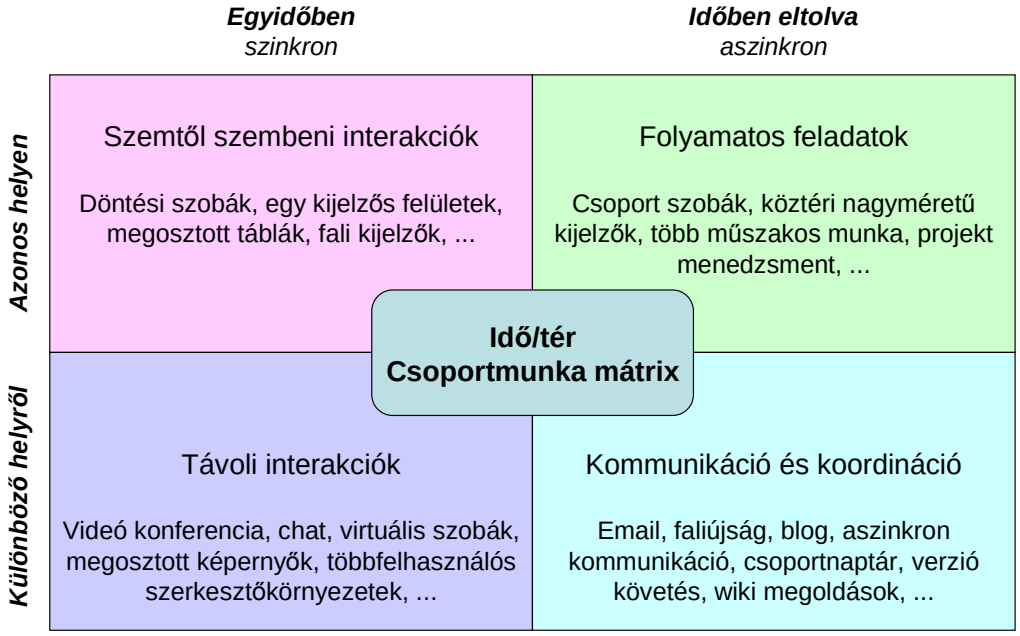
1.2. Csoportmunka támogatás

A számítógépes csoportmunka, vagy más néven kollaboratív munka támogatása az 1980-as évek közepén az asztali számítógépek és számítógép-hálózatok elterjedésével egyidőben kezdődött. Az első kollaboratív alkalmazások elektronikus naptárak voltak, melyekkel könnyebben lehetett közös találkozókhoz mindenkinek megfelelő időpontot választani (Ehrlich, 1987). A kollaboratív rendszerek kutatása az 1990-es években felgyorsult. Köszönhető volt ez a mára szinte minden irodában és otthonban elérhető Internet terjedésének, a hálózati

sávszélességek növekedésének, a globalizációból fakadó, földrajzilag elosztott munkacsoportok iránti igénynek.

A kollaboratív rendszerek kutatásával foglalkozó tudománynak egy 1984-es workshopon lett a "*Computer Supported Cooperative Work*" (röviden CSCW) a neve (Grudin, 1994). CSCW az informatika azon ága, amely a számítógéppel támogatott csoportmunka lehetőségeit, motivációit, hardver és szoftvereszközeit, szolgáltatásait tanulmányozza (Wilson, 1991). A CSCW egy erősen interdiszciplináris terület, melyben az informatikusok mellett pszichológusok és szociológusok is tevékenykednek. Míg utóbbiak főleg a kollaboráló csoporttagok céljainak, módszereinek és elégedettségének analízisével foglalkoznak, az informatikusok elsősorban csoportmunkát támogatni képes eszközöket, úgynevezett *csoportmunka* rendszereket (angolul *groupware*) terveznek, implementálnak és üzemeltetnek.

A csoportmunka rendszerek legáltalánosabban elfogadott osztályozása a felhasználásuk módja szerint, az ún. CSCW mátrix segítségével történik (Baecker, 1995) (ld. 1-2. ábra).



1-2. ábra: CSCW mátrix a csoportmunka együttműködések típusainak és a támogató környezeteknek a kategorizálására. (Baecker, 1995) alapján.

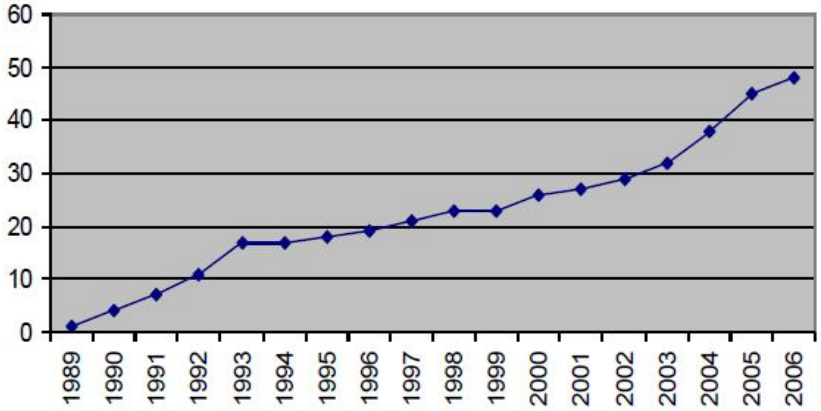
A mátrix első oszlopa a szinkronban (egyidőben) történő kollaborációkat tartalmazza. Ennek felső cellájába tartoznak az egy helyiségben tartózkodó személyek együttműködését segítő eszközök, úgy mint különböző kijelzők, táblák. Alsó cellába a távoli helyről valós időben zajló együttműködések: ide sorolhatók a video- és audiókonferenciák, a képernyőmegosztás, vagy a többfelhasználós szerkesztőkörnyezetek.

A mátrix második oszlopa az időben aszinkron együttműködéseket tartalmazza. Ennek első cellájába a többszörös interakciót igénylő, de egyazon helyen történő tevékenységek tartoznak, úgy mint projekt menedzsment, időben eltolt műszakokban való munka, üzenőfalakon keresztüli munkakoordináció. A jobb alsó cellába az aszinkron és számítógépeken keresztüli interakciók tartoznak, úgy mint email, blogok, csoport naptár, wiki és más hasonló típusú weblapok.

Értekezésemben workflow alkalmazások (irányított gráfok) több személy általi, egyidejű szerkeszthetőségére adok megoldásokat. Eredményeim a mátrix bal alsó cellájába, „workflow gráfokra specializált valós idejű kollaboratív szerkesztőkörnyezet" címmel kerülhetnek be. Ebbe

a cellába tartozik minden olyan rendszer, amellyel több felhasználó tud egyidőben *egyazon számítógépes fájlt szerkeszteni* (real time collaborative editor - RTCE) (Chen, 2006).

RTCE eszközökhöz kapcsolódó kutatások és fejlesztések adják a CSCW témakör legnagyobb szeletét. Ezen belül a legaktívabb két részterület a dokumentumok kollaboratív szerkesztése, és a programkódok kollaboratív szerkesztése (Dewan, Riedl, 1993). Az 1989 és 2006 időszakot vizsgálva készített Chen egy felmérést *dokumentumok* kollaboratív szerkesztésére alkalmas valós idejű rendszerekről (Chen, 2006). A tanulmány alapján megállapítható, hogy az RTCE rendszerek száma évről évre növekszik, mind az akadémiai, kutatási célból létrehozott megoldások, mind az ipari, piaci értékesítésre szánt megoldások tekintetében (ld. 1-3. ábra). Ha ehhez hozzávesszük az RTCE további alkalmazási területeit, úgy mint kollaboratív irodai alkalmazások (pl. (Sun et al, 2006)), kollaboratív médiaszerkesztő és dizájn eszközöket (pl. (Mauve, 2000)), kollaboratív Computer Aided Design (CAD) (pl. (Fuh, Li, 2004)) – akkor egyértelműen kijelenthető, hogy egy *kritikus felhasználói tömeg elérése után az egyfelhasználós szerkesztőrendszerek többfelhasználós környezetekké transzformálása elkerülhetetlen lépés*.



1-3. ábra: Valós idejű, kollaboratív dokumentumszerkesztő rendszerek száma. (Akadémiai és ipari megoldások együttesen) Forrás: (Chen, 2006)

1.3. Az értekezés céljai

Úgy vélem, hogy a workflow technológiák mostanára jutottak el arra a szintre, hogy a felhasználók igényeljék a kollaboratív workflow létrehozás és szerkesztés lehetőségét. Értekezésemben olyan megoldást kívánok adni a workflow szerkesztő környezetek fejlesztőinek, melynek segítségével képesek a jelenleg elterjedt, egyfelhasználós eszközöket többfelhasználós, valós idejű kollaboratív workflow fejlesztő környezetekké alakítani. Az új eszközök lehetőséget adnak a workflow fejlesztőknek ahhoz, hogy több számítógépen keresztül akár egymástól fizikailag távolról együttesen végezzék el munkafolyamatok workflow-val való leírását, fejlesztését, szerkesztését.

A workflow rendszerek utóbbi időben való erőteljes terjedése, azok sokszor különböző irányba mutató fejlesztései azt mutatják, hogy nem lehet számítani a workflow nyelvek harmonizálására, a workflow szerkesztő környezetek számának csökkenésére (Curcin, Ghanem, 2008). *Munkámban éppen ezért általános, rendszerfüggetlen megoldást kívánok adni a kollaboratív workflow fejlesztés problémájára. Dolgozatom a workflow fejlesztés folyamatának analízise, a workflow fejlesztés csoportmunkára vonatkozó igényeinek vizsgálatán keresztül tesz ajánlást a legalkalmasabb kollaboratív workflow szerkesztőkörnyezet eszköztárára. Mivel azonban az elmúlt közel hét évben Grid infrastruktúrákat használó tudományos workflow*

alkalmazások támogatásával és fejlesztésével foglalkoztam, ezért az értekezésben használt gyakorlati alkalmazások és környezetek ebből a témakörből kerülnek ki. Ez azonban nem jelenti, hogy az értekezésben foglalt eredmények csak Grid workflow alkalmazásokra alkalmazhatók.

Workflow alkalmazások fejlesztésének csoportmunka környezetekkel való támogatása egy új, és eddig kevéssé kutatott terület. Annak ellenére, hogy a workflow-k, mint elosztott erőforrásokat használó folyamatok kapcsán sokszor előkerül a „kollaboratív" jelző, ezt a szakirodalomban többnyire arra értik, hogy a workflow futása több személynek – egy kollaboratív csoportnak – a munkáját koordinálja, *magának a workflow-nak a definiálási folyamatát viszont nem érinti, az továbbra is „egyfelhasználós"* (Winograd, 2004).

Más módszerek a workflow-k központi tárolóban való publikálásával és letölthetőségével biztosítanak együttműködési lehetőséget a fejlesztők között. Például a Myexperiment.org egy olyan közösségi weblap, ahova tudományos workflow alkalmazások tölthetők fel, majd láthatók el mindenféle leírással, címkével (Goble, De Roure, 2007). Az ilyen rendszerekkel körülményes és hosszú ideig tart összetett workflow-k megalkotása.

Értekezésem célja a workflow fejlesztést többfelhasználós folyamattá bővíteni, a workflow fejlesztési szakaszban támogatást adni irányított gráfok több felhasználó általi, hatékony szerkesztésére. A javasolt megoldás felosztja a workflow gráfot a résztvevő szerkesztők között oly módon, hogy a részgráfok száma – és ennek következtében az egyidőben szerkesztést végezni képes felhasználók száma – maximális lesz, viszont a teljes gráf helyessége megmarad anélkül, hogy bárkinek a változtatásait kompenzálni, vagy mellőzni kellene.

A javasolt koncepciók a tipikus workflow fejlesztő személyek – a hagyományos értelemben vett programozásban járatlan üzleti-ipari alkalmazottak, vagy e-tudományos kutatók – tudásszintjéhez és elvárásaihoz idomulnak, emiatt a gyakorlatban jól használható megoldást eredményezhetnek. A dolgozat mind a workflow, mind a CSCW közösség számára új eredményeket hoz, erősíti a kettő kapcsolatát, elősegíti közöttük a tudástranszfert. A dolgozat konkrét hozzájárulásai ezen témakörökhöz:

1. A workflow fejlesztés folyamatának kollaboratív csoportmunka szempontjából történő vizsgálata, a workflow fejlesztés kollaboratív munkával szemben támasztott követelményeinek összegyűjtése és analízise.
2. A valós idejű csoportmunka alkalmazásokban használt konkurenciakezelési módszerek összevetése a workflow fejlesztéssel szemben támasztott követelményekkel, a legcélszerűbben használható megoldás kiválasztása és workflow fejlesztő rendszerhez való adaptálása.
3. Egy olyan kollaboratív fejlesztőrendszer megalkotása, amely képes bármilyen irányított gráfként ábrázolt workflow (vagy akár más ilyen gráfként ábrázolható adathalmaz) több felhasználó közötti felosztására, a személyek által végzett konkurens változtatások egyetlen gráfba történő integrálására.
4. Olyan gráf felosztó algoritmusok, amelyek használata egy kollaboratív workflow fejlesztő rendszeren belül biztosítja, hogy a workflow-k szemantikai konzisztenciája több felhasználó egyidejű munkája esetén is megmarad, és így a workflow futása közben elkerülhetőek az inkonzisztenciából adódó hibák, és az utólagos módosítással járó plusz munka.
5. Egy módszer, amellyel kollaboratív szerkesztési célból workflow gráfokat felosztó algoritmusok hatékonysága összehasonlítható. Az összehasonlítás lehetőséget ad ismert követelményrendszer esetén a legmegfelelőbb gráf felosztó algoritmus kiválasztására.

6. Az összehasonlítási módszer segítségével megalkotom azt a workflow gráf felosztó algoritmust, amely kollaboratív szerkesztés közben a lehető legtöbb felhasználó egyidejű munkáját teszi lehetővé, ezáltal biztosítva a leghatékonyabb együttműködést.

Az értekezés 2. fejezete ismerteti a workflow fejlesztés ma általános, egyfelhasználós folyamatát, majd annak kiterjesztését több konkurens felhasználóra. Leírásra kerülnek a valós idejű csoportmunka alkalmazásokban használt konkurenciakezelési megoldások, majd ezek workflow fejlesztők elvárásaival való összevetése alapján egy irányított gráfok szerkesztésére használható szerkesztőkörnyezet szolgáltatásai.

A 3. fejezet a workflow gráfok kollaboratív szerkesztés közbeni konzisztenciájának biztosítására fókuszál. Ismertetésre kerülnek a konzisztencia csoportmunka témakörben használt jelentései, illetve azok workflow gráfokra való adaptációja. A 2. fejezetben kidolgozott rendszer továbbfejlesztésével olyan új megoldásokat ismertetek, melyekkel egy valós idejű kollaboratív szerkesztőrendszer biztosítani tudja a gráfok konzisztenciáját még több felhasználós, konkurens szerkesztés esetén is. A módszerek nemcsak a konzisztenciát, de azt is garantálják, hogy a felhasználók gráfon végzett módosításai mind bekerülnek a workflow végleges változtatásába, utólagos kompenzálásra, javításra nincs szükség.

A 4. fejezetben a 3. fejezet módszereinek értékelésére és a felhasználók szempontjából leghatékonyabb módszer kiválasztására kerül sor. A hatékonyságot az egyidőben egyazon gráfon dolgozni képes felhasználók számával, illetve a felhasználók igényeinek igazságos kiszolgálásával fogom jellemezni.

Az 5. fejezetben a 4. fejezetben használt metrikák és vizsgálati módszerek alapján kidolgozom a korábbiaknál hatékonyabb, továbbá az adott feltételrendszer mellett elérhető leghatékonyabb gráf felosztási módszerek.

A 6. fejezet az értekezés témájához kapcsolható további kutatási lehetőségeket ismerteti, a 7. fejezet a munka összegzését adja. A 8. fejezetben felsorolom a kutatás témakörében publikált saját cikkeimet. A 9. fejezet a hivatkozott irodalmak jegyzéke.

2. Workflow alkalmazások kollaboratív szerkesztése

Az Internet alapú kollaboratív alkalmazások az elosztott rendszerek egy olyan részterülete, amelyek a világhálón keresztül *ember-ember interakciót* tesznek lehetővé. Kutatásom célja olyan valós idejű kollaboratív workflow fejlesztőrendszer meghatározása, amely felhasználói érzet tekintetében a lehető legkisebb eltérést adja az egyfelhasználós workflow fejlesztő rendszerekhez képest, és amely a lehető leghatékonyabban és „legcélszerűbben" használható workflow fejlesztésen keresztül zajló csoportmunkára.

2.1. A workflow fejlesztési folyamat

Bármilyen workflow felhasználási területet is vizsgálunk, aránylag nagyszámú fejlesztőrendszert lehet beazonosítani, akár akadémiai, akár ipari megoldásokat tekintünk. Annak ellenére, hogy ezen rendszerek egymástól jelentősen eltérő workflow nyelveket és felhasználói felületeket nyújthatnak, önálló desktop alkalmazásként vagy kliens-szerver architektúrában működhetnek, a workflow fejlesztési folyamat mindegyiknél azonos.

Egy workflow életciklusa két részre bontható: szerkesztési fázisra és végrehajtási fázisra (Ld. 2-1. ábra). A szerkesztési fázis kiindulópontja lehet egy teljesen üres gráf – ekkor a workflow fejlesztését „nulláról" kezdi el a felhasználó – vagy lehet egy már meglévő workflow – ekkor annak továbbfejlesztéseként áll majd elő az új alkalmazás. (Ez utóbbi esetben a kiindulópontként használt workflow származhat a felhasználó gépéről, vagy egy központi tárolóból.)

A szerkesztési fázis során a felhasználó egy vagy több *szerkesztési szakaszban* (editing session) fejleszti a gráfot (Yang et al, 2000). Egy-egy ilyen szerkesztési szakasz a fejlesztés bonyolultságától és a felhasználó munkamódszerétől függően percekig, órákig, de akár napokig is eltarthat. Egy ilyen szerkesztési szakasz során a felhasználói szerkesztőkörnyezetben végig meg van nyitva a gráf, a szakaszok közötti időszakokban viszont a felhasználó nem dolgozik a workflow-n.

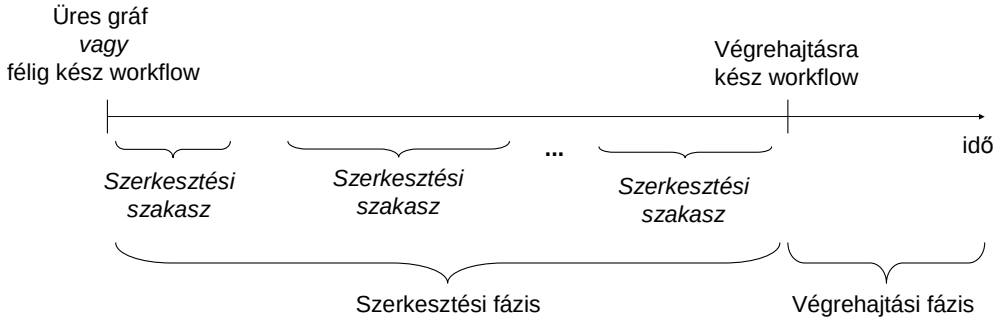
A szerkesztési fázist követő végrehajtási fázisban történik a workflow futtatása[2]. Ez a workflow tartalmától, a workflow futtató rendszertől függően lehet lokális gép, klaszter, Grid, szolgáltatás-hálózat, speciális üzleti vagy ipari környezet. Munkám a workflow-k kollaboratív szerkesztésére fókuszál, ezért a továbbiakban ezt a fázist fogom kollaborativitás szempontjából elemezni.

A jelenlegi workflow fejlesztőkörnyezetek szinte kivétel nélkül mind egyfelhasználósak. Egyfelhasználós rendszerekben egy-egy szerkesztési szakasz során csak egyetlen felhasználó szerkesztőrendszerében van megnyitva a workflow, és csak ebben az egy szerkesztőben végbemenő módosítások hatására változik meg annak állapota. Munkámban nem egy konkrét megvalósításhoz köthető workflow típussal foglalkozom, ezért a workflow fogalmát a lehető legáltalánosabb módon a következőképpen használom:

Def. 1. A ***workflow*** egy irányított gráf: W = (V, E), ahol V a csúcsok, E az irányított élek halmaza.

[2] Ugyan egyes workflow rendszerekben vannak próbálkozások a szerkesztési és végrehajtási fázis átlapolódásának támogatására, ilyen eszközökkel a futás közbeni szerkesztés csak a *még nem futó és még nem lefuttatott* gráf-részeken lehetséges. Ezen részgráfok fejlesztése viszont így egyenértékű a hagyományos workflow-k esetén a végrehajtási fázis előtti szerkesztéssel, kollaborativitás szempontjából tehát nem támaszt további követelményeket.

Ez a definíció a workflow szakirodalomban széles körben használt, és lényegében *bármilyen workflow nyelvre használható* (Aalst et al, 2003)(Barker, Hemert, 2008)(Yu, Buyya, 2005)(Meilin et al, 1998).



2-1. ábra: Workflow életciklusa: szakaszokból álló szerkesztési fázis, amelyet végrehajtási fázis követ.

A workflow egy csomópontja lehet bármilyen aktivitás, amely adatokat olvas, transzformál, termel. Irodai környezetekben akár szoftver, akár ember által ellátott feladat is megjelenhet csomópontként. Tudományos, kutatói, ipari workflow-kban csomópont lehet például Web szolgáltatás, futtatásra szánt számítási feladat (job), valamilyen számítási adatmozgatási, vagy tároló szolgáltatás.

Két csomópont között futó él a csomópontok közötti függőséget reprezentálja. A függőség lehet adatfüggőség (data dependency), amikor is az él forrás csomópontja által termelt adatot a cél csomópont bemenő adatként használ. A függőség lehet előidejűség (control dependency) amikor is a forrás csomópont lefutása (befejeződése) után kezdődhet csak el a cél csomópont végrehajtása.

Egy-egy csomópont, vagy egy-egy él *N* db paraméterrel van a workflow-ban leírva. A paraméterek az adott entitás tulajdonságait adják meg. Ezen tulajdonságok – és emiatt a paraméterek száma és típusa – a csomópont vagy él jellegétől függ. Egy Web szolgáltatás csomópont például rendelkezhet egy WSDL dokumentumra mutató URL paraméterrel, a szolgáltatás bemenő adatait tartalmazó szövegmezőkkel és XML dokumentumokat tároló paraméterekkel. Egy Grid job típusú csomópont megadható a futtatandó kód helyével, annak parancssori paramétereivel, környezeti változóival, stb. Egy irányított él paraméterei meghatározhatják például az adattovábbítás módját, (pl. preferált útvonal, tömörítési mód, titkosítás szintje, dokumentum fejléce, stb.)

Egyfelhasználós rendszerek esetén egy workflow-hoz csak egy felhasználó fér hozzá, ezért egy szerkesztési szakasz során az csak egyetlen szerkesztőrendszerben lehet a gráf megnyitva. A szerkesztőrendszer bármilyen felhasználói felületet nyújt a workflow szerkesztéshez, abban teljesen általános esetben a következő műveletek végrehajtását kell biztosítania a felhasználó számára:

Workflow *csomópontokra* vonatkozó műveletek:

- `Add(X)`: hozzáad egy új, *X* névvel ellátott csomópontot a workflow gráfhoz. A csomóponthoz létrehozásakor nem kapcsolódik semmilyen él, paramétereinek értéke alapértelmezés szerint definiált. (üres, vagy valamilyen alapértelmezett értékekkel bír. Pl. a csomópont a rendszertől kap egy egyedi azonosítót, így ez az azonosító paraméter nem lesz üres.) A gráf csomópontokat nagy betűkkel jelölöm.

- `Delete(X)`: eltávolítja az *X* csomópontot a workflow-ból. Ha az *X* csomóponthoz élek is kapcsolódnak, akkor azokat ez az utasítás nem érinti.
- `Modify(X, pId, p)`: az *X* csomópont *pId*-vel azonosított paraméterének értékét *p*-re változtatja. Az új értékkel felülíródik a paraméter korábbi értéke.
- `Extend(X, pId)`: az *X* csomóponthoz hozzárendel egy *pId*-vel azonosított új paramétert. A paraméter egyúttal alapértelmezett értékét is megkapja.
- `Remove(X, pId)`: az *X* csomóponttól leválasztja annak *pId*-vel azonosított paraméterét.

Workflow *élekre* vonatkozó műveletek:

- `Add(X, Y)`: hozzáad a workflow-hoz egy új élt, amely az *X* csomópontból az *Y* csomópontba fut. Az él paramétereinek értéke alapértelmezés szerint definiált. (üres, vagy valamilyen alapértelmezett értékekkel bír.). A későbbiekben egy *X* csomópontból egy *Y* csomópontba futó élre az *xy* kisbetűs jelölést használom.
- `Delete(xy)`: eltávolítja az *xy* élt a workflow-ból. Ha az *xy* él végpontjai még léteznek, akkor azokat ez az utasítás nem érinti.
- `Modify(xy, rId, r)`: az *xy* él *rId*-vel azonosított paraméterének értékét *r*-re változtatja. Az új értékkel felülíródik a korábbi érték.
- `Extend(xy, rId)`: az *xy* élhez hozzárendel egy *rId*-vel azonosított új paramétert. A paraméter egyúttal alapértelmezett értékét is megkapja.
- `Remove(xy, rId)`: az *xy* éltről leválasztja annak *rId*-vel azonosított paraméterét.

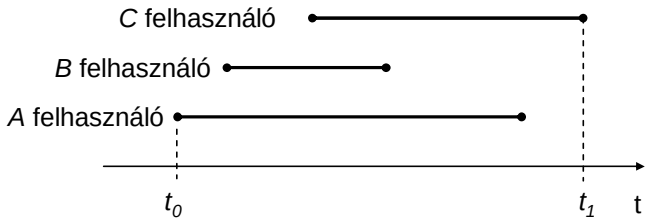
A műveletlista a teljesen általános esetet mutatja. A valóságban jónéhány rendszer a műveletek csak egy részét támogatja – például úgy, hogy a csomópontok és élek paraméterlistáját a felhasználók nem változtathatják. (Vagyis nem támogatottak az Extend és Remore műveletek.)

Amit fontos kiemelni, hogy a listában nincs olvasás (read) művelet sem csomópontokra, sem élekre: minden csomópont és él eleve beolvasásra kerül a workflow megnyitásakor, azok további olvasására nincs szükség.

2.2. Workflow fejlesztés kiterjesztése több felhasználóra

Olyan valós idejű kollaboratív szerkesztőrendszert kívánok megalkotni, amellyel *több felhasználó egyidőben tud egyazon workflow alkalmazást szerkeszteni*. Ez a workflow életciklusa tekintetében az egyfelhasználós felhasználási módhoz képest annyi változást jelent, hogy egy-egy *szerkesztési szakasz* során több felhasználó is módosíthat az alkalmazáson. Egy ilyen szerkesztési szakasz (továbbiakban kollaboratív szerkesztési szakasz) kezdete az a pillanat, amikor az első felhasználó megnyitja szerkesztésre a gráfot, és akkor ér véget, amikor az utolsó felhasználó is bezárta a workflow-t a szerkesztőrendszerében[3] (Ld. 2-2. ábra).

[3] Valójában egyfelhasználós esetben is az első felhasználó megnyitásával kezdődik a szerkesztési szakasz, és az utolsó felhasználó bezárásával ér véget, viszont ott ugyanaz a két személy. Kollaboratív esetben a két személy nem feltétlenül ugyanaz.



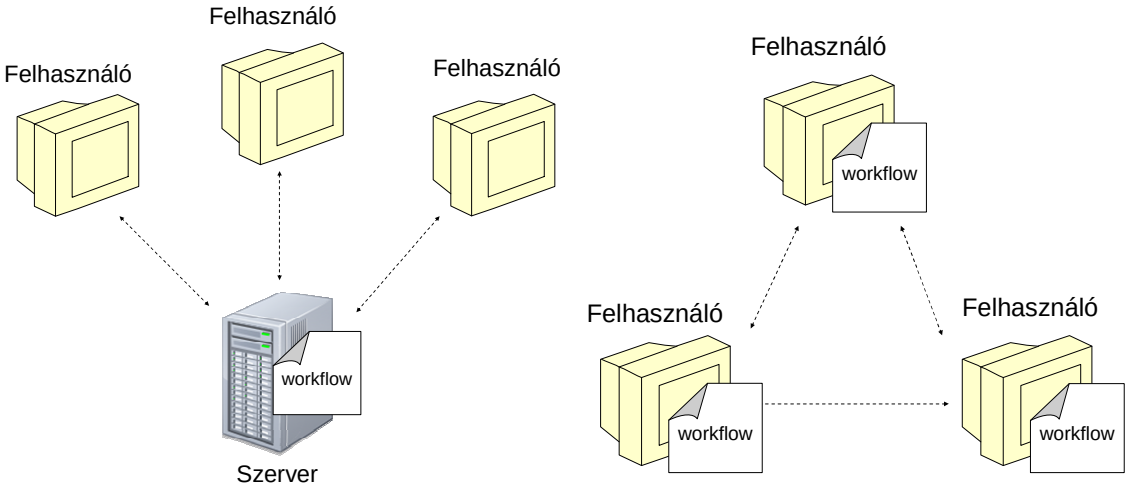
2-2. ábra: Három felhasználó részvétele egy kollaboratív szerkesztési szakaszban. t_0 a szerkesztési szakasz kezdete (A felhasználó megnyitja a gráfot), t_1 a szerkesztési szakasz vége (C felhasználó bezárja a gráfot).

Kollaboratív workflow-k életpályája az egyfelhasználós workflow-kéhoz hasonlóan szintén szerkesztési fázisra és futtatási fázisra tagolódik, a szerkesztési fázis pedig több szerkesztési szakaszból áll. Ebben a tekintetben nincs különbség. Hogy egy-egy kollaboratív szerkesztési szakasz során hány személy dolgozik a gráfon, milyen munkamegosztás szerint, milyen hosszú ideig, és hogy a kollaboratív szakaszok milyen gyakorisággal követik egymást, az nem adható meg általánosságban. Ez mind a megosztott gráftól, a csoport jellegétől, a felhasználók munkamódszerétől függ.

Interaktív szoftverek kollaboratív környezetté alakítása során a felhasználói élmény megtartása az egyik legfontosabb követelmény. Ez a következő kritériumokat támasztja (Sun et al, 1998):

1. Alacsony válaszidő megtartása: a kollaboratív fejlesztőkörnyezet a helyi felhasználó akcióira gyorsan adjon választ, a kollaboratív architektúrává alakítás ne járjon megnövekedett késleltetéssel.
2. Magas fokú párhuzamosság: egyidőben több felhasználó tudja a megosztott entitást módosítani.
3. Kommunikációs késleltetés elrejtése: a rendszernek magas és kiszámíthatatlan késleltetéssel rendelkező hálózatokon, pl. az Interneten is működnie kell.

Ez a követelményrendszer a csoportmunka környezetekben replikált architektúrát eredményez, melyben a megosztott entitás – esetünkben a workflow – a kollaboratív csoport tagjainak gépein egy-egy példányban tárolódik, valamilyen központi szerveren való tárolás helyett. (Preston, 2007). A 2-3. ábrán látható a két megközelítési mód közötti különbség. A replikált architektúra lehetőséget ad a felhasználóknak a megosztott entitás saját példányának módosítására, ezen változtatások saját felhasználói interfészükön való azonnali megjelenítésére.



2-3. ábra: Központi és replikált adattárolási módszer kollaboratív szerkesztőkörnyezetben. Az interaktivitás megköveteli a replikált architektúra használatát. A résztvevők közötti kommunikációs mód mindkét esetben akár kliens-szerver, akár peer-to-peer lehet. (Preston, 2007) nyomán.

A replikált architektúrában a kollaboráló felhasználók számítógépeinek (kollaboráló gépeknek) az együttműködése a megosztott entitást módosító utasítások kezeléséből áll. Egy-egy ilyen utasítás a következő állapotsoron megy keresztül (Ellis, Gibbs, 1989): *létrehozás*, *helyi gépen való végrehajtás*, *továbbítás a többi felhasználó gépének*, *megérkezés távoli gépekhez*, *távoli gépeken való végrehajtás*. Konkurenciakezelés nélkül a helyi utasítások létrehozásuk után azonnal lefutnak a helyi gépen, egy távoli géphez való megérkezésük után pedig ott szintén azonnal végrehajtódnak. A probléma ezzel, hogy a hálózati átvitel ideje nem nulla, az átviteli idő a különböző géppárok között, illetve akár egy-egy géppárra de különböző utasításokra is más-más lehet. Ennek következében *az üzenetek a kollaboráló gépeken más-más sorrendben érkeznek meg, különböző sorrendben kerülhetnek végrehajtásra és ez a megosztott entitás integritását sértheti*.

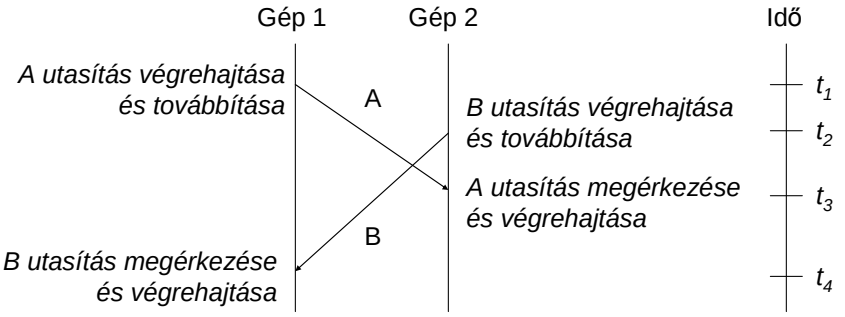
Több felhasználó megosztott entitáson való együttműködése – legyen az az entitás egy dokumentum, egy naptár, egy rajz, programkód vagy egy workflow – konfliktust okozhat. Az ebből fakadó problémák ellen a kollaboratív rendszereknek *konkurenciakezelő megoldással* kell védekezniük (Ellis, Gibbs, 1989).

2.2.1. Konkurenciakezelés módszerei

Elosztott rendszerekben való konkurenciakezelés kutatása legelőször az elosztott adatbázisok és a párhuzamos szimulációk területén belül történt (Bernstein, Goodman, 1987)(Fujimoto, 1990). Az itt született megoldások két nagy körbe sorolhatók: *szerializációs módszerek* és *zárolási módszerek* (Bernstein, Goodman, 1987)(Cellary et al, 1988).

A 2-4. ábra egy példát mutat két kollaboráló gép aszinkronitására (Gép 1 és Gép2). Gép 1 a t_1 időpillanatban létrehozza, az entitás saját példányán végrehajtja, majd Gép 2-nek továbbítja az A utasítást. Gép 2 ugyanezt teszi t_2 időpontban a B utasítással. Ezután a t_3 időpontban Gép 2 megkapja és végrehajtja a saját példányán az A utasítást, t_4 időpontban pedig Gép 1 megkapja, és saját példányán végrehajtja a B utasítást. Mivel konkurencia nélkül a két gép más sorrendben futtatja a saját példányán az A és B utasításokat, ezért az entitás egyik, másik, vagy akár mindkét gépen kieshet a valós állapotból, inkonzisztens állapotba kerülhet. Ez akkor okoz problémát, ha az A és B utasítások *nem kommutatívak*, vagy ha valamely gép által generált utasítások *nem*

atomi módon való futtatása (vagyis más gépről származó utasításokkal való keverése) veszélyezteti azok helyességét. (Pl. Egy A, A_2, A_3 utasítássor nem keverhető a B utasítással.)



2-4. ábra: Konfliktusban álló adatmanipulációs utasítások átlapolódása veszélyeztetheti az adatok integritását. Forrás: (Greenberg, Marwood, 1994)

2.2.1.1. Szerializációs módszerek

A CSCW környezetekben alkalmazott szerializációs módszerek a hagyományos adatbázis-kezelőknél alkalmazott szerializációs módszerek replikált architektúrákra adaptált változatai. Ezek a módszerek az átfedésben levő, objektumokat manipuláló tranzakciók utasításainak olyan rendezését állítják elő, melyben a konfliktusban álló utasítások egyidőben nem, kizárólag csak egymás után futhatnak (Bernstein, Goodman, 1987). A szerializációs módszerek gyakran a tranzakciók utasításainak teljes rendezését adják. Szerializáción belül pesszimista és optimista módot lehet megkülönböztetni[4].

A pesszimista szerializáció azt biztosítja, hogy egy kollaboráló gép addig ne futtathasson helyi utasítást, amíg a többi gép által már továbbított utasítások oda nem értek hozzá, és le nem futottak nála. Pesszimista szerializáció megvalósítható koordinációs üzenetek, globális óra, multi-verzió készítés segítségével és garantálja, hogy *több gépen kezdeményezett tranzakciók utasításai minden gépen ugyanabban a sorrendben kerülnek végrehajtásra*. A 2-4. ábrán látható példa esetében a pesszimista szerializáció biztosítja, hogy a 2. gép addig nem hajtja végre a saját objektumpéldányán, illetve küldi el az 1. gépnek a B utasítást, amíg az A utasítás meg nem kapta és végre nem hajtotta.

Az optimista szerializációs módszerek a pesszimistával szemben nem blokkolnak egyetlen utasítás-végrehajtást sem, a konfliktusok megelőzése helyett utólagos konfliktusfeloldást biztosítanak. Ezek a módszerek abban bíznak, hogy ritkán áll fenn konfliktushelyzet, és amikor fellép, akkor egyszerűbb azok utólagos javítása, mint a folytonos megelőzés.

Az optimista szerializációban alkalmazott konfliktusfeloldás egyik módja az „visszavon-ismétel” (undo-redo) módszer, amikor is konfliktus esetén a rendszer állapotát a legutóbb végrehajtott utasítások visszagörgetésével a konfliktushelyzet előtti helyes állapotba kell visszaállítani (undo), majd onnan egy olyan új állapotba, amelybe az utasítások megfelelő sorrendben történő végrehajtásával jutunk (redo). Optimista szerializációval a 2-4. ábrán látható példában Gép 2 a t_3 időpillanatban, A utasítás kézhezvételekor veszi észre, hogy B-t túl hamar futtatta (mivel A globális időbélyege alacsonyabb lesz, mint B időbélyege), emiatt visszagörgeti B-t, majd végrehajtja először A-t, azután B-t. Az eredmény, hogy mind az 1., mind a 2. gépen A, B sorrendben kerülnek az utasítások végrehajtásra. Ennél a konfliktusfeloldási módszernél a már végrehajtott utasítások nyilvántartása, a rendszer állapotainak mentése, a visszapörgetés, majd

[4] Az „optimista” kifejezés helyett ebben a kontextusban az „aggresszív”, „pesszimista” helyett a „konzervatív” vagy „nem optimista” kifejezések is használatosak.

újrafuttatás különböző protokollokkal is megvalósítható (Jefferson, 1985)(Karsenty, Beaudouin-Lafon, 1993).

Optimista szerializációban alkalmazott konfliktusfeloldás másik módja a „művelet transzformáció” (angolul *operation transformation*), amely konfliktus felismerésekor nem görget vissza egyetlen már végrehajtott utasítást sem, viszont az új utasítást nem változatlan, hanem egy olyan transzformált formában hajtja végre, mely pontosan azt a végeredményt adja, amelyet a konfliktust okozó utasítások helyes sorrendben történő végrehajtása adott volna. A gyakorlatban számos undo-redo módszert alkalmazó rendszer is végez művelet transzformációt, ezzel ugyanis több utasítás összevonható, a visszalépés és előregörgetés hatékonysága javítható (Karsenty, Beaudouin-Lafon, 1993)(Prakash, Knister, 1992). A fenti példában művelet transzformáció használata esetén a 2. gépnek az A utasítás kézhezvétele után nem azt, hanem egy olyan A’ utasítást kéne végrehajtania, amely ugyanazt az állapotot fogja eredményezni mintha az A és B utasítások az A, B sorrendben lettek volna végrehajtva.

A szerializációs módszereknek mind megvannak a jellegzetes problémáik, egyik sem használható általánosan: a pesszimista módszer lassú futást, sok várakozást okoz és így nem hatékony megoldás abban az esetben, amikor nincs sok konfliktushelyzet egy rendszerben. Az optimista módszerek ilyenkor a blokkolás nélküli utasítás-végrehajtás miatt általában hatékonyabbak. Az optimista megközelítés viszont olyan korrekciós mechanizmusokkal jár, melyek leprogramozása és üzemeltetése jelentős költség. Ami ennél is fontosabb, hogy sok helyzetben azért nem használható az optimista módszer, mert *számos utasítás nem görgethető vissza* (különösen ha az külső rendszerekben, a való világban való folyamat elindítása volt), és *sok utasításhoz nem lehet transzformált megfelelőt találni*.

2.2.1.2. Zárolási módszerek

Zár (angolul *lock*) alapú konkurenciakezelési módszerek privilegizált hozzáférést biztosítanak a résztvevőknek a megosztott entitás bizonyos részéhez. Zárolás segítségével vagy a teljes rendszerben globális szerializálhatóság biztosítható, vagy az egyes résztvevők utasításaiból olyan atomi utasítássorozatok képezhetők (lokális szerializáltság), melyek egymás után tetszőleges sorrendben futhatnak, viszont át nem lapolódhatnak (Greenberg, Marwood, 1994).

Zár alapú rendszerben a kollaboráló résztvevő a megosztott entitást alkotó egy (vagy több) komponensre vonatkozó lefoglalási kérést küld, és amennyiben a komponenseket még senki sem zárolta, a kérése jóváhagyásra kerül, a komponens lefoglalódik. Amennyiben a komponens már zárolt, a kérés elutasítódik. Zárolt komponenseket egyedül a zár tulajdonosa szerkesztheti, amennyiben azokra már nincs szüksége lemondhatja a foglalást, ezzel elérhetővé téve a komponenst a többi résztvevő számára. A szerializációhoz hasonlóan a zárolás alapú módszerek is néhány nagy kategóriába sorolhatók: *pesszimista*, *fél-optimista* (semi-optimistic), *optimista*. Ezeket az 1. táblázat foglalja össze.

1. táblázat: Optimizmus-változatok zárolás alapú konkurenciakezelésre.

Zárolás optimizmusának szintje	Módosíthatja a zárolást igénylő a kért komponenst, amíg a zárolásra vár?	Elengedhet a zárolást igénylő egy lefoglalásra kért, és módosított komponenst, amíg a zárolásra vár?
Pesszimista zár használat	Nem	Nem
Fél-optimista zár használat	Igen	Nem
Optimista zár használat	Igen	Igen

Pesszimista zár használatról beszélünk amennyiben egy gépnek várnia kell a zárolás jóváhagyásáig mielőtt a kért entitást módosíthatná. Amennyiben a zárolási kérés blokkoló, a gép nem is hajthat végre más utasítást a kérés eredményének kézhezvételéig. Amennyiben nem blokkoló a kérés, a gép tovább folytathatja a munkáját más komponensekkel. *Pesszimista zár használat esetén nem fordulhat elő, hogy több gép azonos komponenst egyidőben módosít*.

Az *optimista zár használati módok* (fél-optimista is ide tartozik) alapelvüket tekintve hasonlóak az optimista szerializációhoz: feltételezik, hogy kevés konfliktushelyzet lesz a rendszerben, vagyis kevés zárolási kérés kerül majd elutasításra. Ennek fényében a módszerek a zárolási kérésekkor úgynevezett „kísérleti zár" (tentative lock) azonnal adnak a felhasználóknak, és ez már elegendő a komponens szerkesztéséhez, vagyis nem kell megvárni a zárolási igény elbírálását. Ha később a zár megadásra kerül, akkor a kísérleti zár teljes értékű zárolásra változik. Ha a zár megtagadásra kerül, akkor a komponensen addig elvégzett változtatások eldobásra kerülnek, a komponens eredeti állapota kerül visszaállításra. Ha valóban ritkán van zárolási elutasítás, és valóban sokáig kell várni egy-egy zárolási kérés elbírálására, akkor a már elvégzett módosítások eldobásának ára valóban alacsonyabb lehet, mint a várakozás költsége.

A különböző zár típusok közötti másik különbség, hogy optimista zár használat esetén (fél-optimista esetén nem) a kísérleti zárral lefoglalt és módosított komponensek elengedhetők, vagyis az ilyen zár tulajdonosa a kísérleti zár teljes értékű zárrá válása, és annak elengedése előtt már elkezdhet más komponenseken zárolást kérni. Ez nem is okoz problémát, ha a kísérleti zár végül valóban teljes értékűvé válik, mert ekkor a módosított komponens új állapota véglegesítődik. Ha azonban a kísérleti zár nem válhatott teljes értékűvé, és ennek ismerete nélkül a kollaboráló gép további komponenseket úgy kezdd el szerkeszteni, hogy azok állapota feltételezi a korábbi módosítások végleges voltát, akkor az elutasítás hatására kaszkád-visszaállításra kerülhet sor. Ennek során az elutasított zárral módosított komponens és minden további, annak módosított állapotára építő komponens eredeti állapotba kerül vissza. A fél-optimista zárolás nem engedi meg a kísérleti zárral foglalt komponensek elengedését, emiatt ott kaszkád-visszaállításra nem kerülhet sor. A pesszimista rendszerek pedig nem is használnak kísérleti zárat, tehát ott sincs erre lehetőség.

2.2.2. Konkurenciakezelés valós idejű kollaboratív rendszerekben – szakirodalmi áttekintés

Ahogy azt többen is kimutatták (Munson, Dewan, 1996)(Ellis, Gibbs, 1989), az elosztott adatbázisoknál és szimulációs rendszereknél alkalmazott konkurenciakezelési megoldások nem minden esetben alkalmazhatók kollaboratív szerkesztőkörnyezetekre. Ennek okai:

1. A kollaboratív környezetben megosztott entitások szemantikája jóval bonyolultabb lehet, mint az adatbázisok írás-olvasás műveletei. Emiatt az adatbázisoknál alkalmazott konkurenciakezelési módok túl nagy megkötéseket jelenthetnek egy kollaboratív környezetben.
2. Adatbázis környezetekben a konkurens tranzakciók nem láthatják egymás munkáját, az adatbázisok azt az érzetet adják a tranzakcióknak, hogy egyedüliként futnak (izoláció). Kollaboratív környezetekben gyakran hasznos, sőt szükséges hogy a felhasználók egymás munkáját nyomon követhessék, és ezáltal a saját tevékenységüket átértékeljék, a csoport érdekeihez igazítsák.
3. Bizonyos kollaboratív rendszerekben a felhasználók hosszabb-rövidebb ideig elviselik a megosztott entitás inkonzisztens állapotát, és így nincs szükség folyamatos

konkurenciakezelésre. Adatbázisok esetén a konzisztencia folyamatosan fenntartandó feltétel.

4. Konfliktus esetén egy hagyományos adatbázis-kezelő a konfliktusban részt vevő valamelyik tranzakciót abortálja, addig végzett változtatásait visszaállítja. Kollaboratív rendszerekben a tranzakciók változtatásai időigényes, nehezen megismételhető emberi munka eredményei. Azok megsemmisítése értékes eredmények eldobását jelentené.

Csoportmunka rendszerek konkurenciakezelési módszerei az adatbázisoknál alkalmazott megoldásokból indultak ki. Míg elosztott adatbázisokat leggyakrabban előre programozott tranzakciók, addig csoportmunka rendszereket emberek használnak. Az emberek a pesszimista módszereknél tapasztalható várakozásokkal és az optimista módszereknél tapasztalható adatvesztéssel szemben néha jobban, néha viszont kevésbé toleránsak, mint a programok.

A legelső kollaboratív rendszerek a munkák időbeni eltolásának módszerét (angolul *turn taking* vagy *floor control* használták, amely egyidőben csak egy személynek ad hozzáférést a megosztott entitáshoz (Sarin, Greif, 1985). Ez tulajdonképpen a zár alapú konkurenciakezelés azon változata, amikor a teljes megosztott entitás egyetlen komponensként csak egyben foglalható le. Ez a kikötés számos alkalmazási területen nyilvánvalóan túl nagy megszorítást jelent, és a kollaboratív rendszerek fejlesztésének olyan irányát generálta, amelynek végén egyesek teljesen szakítottak a konkurenciakezeléssel, azt teljes egészében a felhasználókra bízva (Greenberg, Bohnet, 1991)(Ellis et al, 1991).

A felhasználókra bízott konfliktuskezelést használó rendszerek szükségszerűen nagy hangsúlyt kell hogy fektessenek a felhasználók tudatosságának (awareness) biztosítására, vagyis minél több olyan információ megjelenítésére, amely alapján a felhasználók egymás munkáját nyomon követhetik, szándékukról képet alkothatnak, ami alapján az esetleges konfliktushelyzeteket felismerhetik és feloldhatják (Stetik et al, 1987). *Az ilyen módszerek az optimista szerializációhoz és zárolási módszerhez hasonlóan azt feltételezik, hogy ritkán lesz konfliktushelyzet, és még ha konfliktus lép is fel több felhasználó között, az könnyen feloldható.* A teljesen felhasználókra bízott konkurenciakezelés sem nyújt azonban minden esetben célszerű megoldást:

- A tudatosság jelentős többletkommunikációval jár, melyet bizonyos felhasználók nem tolerálnak, mások nem tudnak hatékonyan kihasználni.
- A megosztott entitás méretének és a felhasználók számának növekedésével a felhasználói tudatosság mértéke romlik, emiatt egyre több konfliktus keletkezik, és egyre később veszik észre azokat a felhasználók.
- Konfliktuskezelés hiányában a felhasználók rá vannak kényszerítve, hogy konfliktusfigyeléssel, felismerés esetén konfliktusfeloldással foglalkozzanak. Ez egyrészt a tényleges munka kárára időkiesést jelent, másrészt a kompenzáció során a korábbi módosítások visszaállítása, módosítások eldobása mind-mind további elvesztegetett idő és energia. *Kollaboratív rendszerek felhasználói sokszor előnyben részesítik a konfliktusok megelőzését az utólagos kezeléssel szemben* (Munson, Dewan, 1996).

Az utóbbi körülbelül egy évtizedben a két extrém konkurenciakezelési megközelítés helyett jóval árnyaltabb, egy-egy konkrét csoport igényeire jobban illeszkedő megoldások születtek. *Ezek a csoportmunka megoldások azonban mindig kiindulási alapként, építőkövekként használják az elosztott adatbázisokra kidolgozott konkurenciakezelési módszereket.*

Replikált architektúrát használó kollaboratív rendszerek megalkotásában rejlő egyik legnagyobb kihívás a lokális másolatok konzisztenciájának biztosítása (Ignat, Norrie, 2008). *Ez a fajta konzisztencia az úgynevezett szintaktikai konzisztencia, mely garantálja, hogy a megosztott*

entitásnak minden kollaboráló ugyanazt a képét látja (Sun, 2002). Kollaboratív rendszerek kutatásában a szintaktikai konzisztencia biztosítása jelenős irodalommal rendelkezik. Sun et al. széles körben elfogadott definíciója szerint a szintaktikai konzisztenciát biztosító módszerek az alábbi, így együtt kizárólag csoportmunka szoftverekben jelentkező, és együttesen CCI problémaként hívott jelenség ellen nyújtanak megoldást (Sun et al, 1998).

1. *Konvergencia probléma* (convergence problem): A megosztott entitást szerkesztő műveletek lokális másolatokon történő eltérő végrehajtási sorrendje a másolatok között tartalmi divergenciát okoz, azok tartalmilag egyre jobban eltérnek egymástól.
2. *Oksági viszony megsértésének problémája* (causality violation problem): előfordulhat, hogy valamely gépen a távoli felhasználó(k) módosításai nem ok-okozati sorrendben érkeznek meg és kerülnek lefuttatásra. Emiatt például egy művelet az entitás olyan állapotára vonatkozik, amely jelenleg nem létezik, és amely miatt az nem hajtható végre.
3. *Szándék sérülés problémája* (intention violation problem): A megosztott entitást módosító műveletek konkurens generálása miatt előfordulhat, hogy egy szerkesztési lépés távoli gépen való végrehajtásakor nem azt a változást hozza létre, amely a létrehozójának a generáláskor szándékában állt. Ennek oka, hogy az művelet generálásakor a generáló környezete még más állapotban mutatta az entitást, mint amiben kellett volna.

Sun et al. munkája összegzi (Sun et al, 1998), hogy a CCI problémákra a következő konkurenciakezelési módszerek jelentenek megoldást: munkák időbeni eltolása, zárolás alkalmazása, szerializáció. Ezeket bővebben ismertetik az alábbi alfejezetek.

2.2.2.1. Munkák időbeni eltolása (turn-taking)

A fentebb már ismertetett „turn-taking” módszer a zárolás lehető legnagyobb granularitással alkalmazott változata, és mindhárom CCI problémára megoldást ad. Használata ma elsősorban a csoportos szoftverfejlesztésben az úgynevezett verziókezelő rendszerekre jellemző (Version Control Systems - VCS).

A VCS-ek tetszőleges típusú fájlok verziókövetésére és több felhasználó által konkurens módon történő szerkesztésére használható. VCS rendszereknek számos implementációja létezik, ma a produkciós szinten legelterjedtebbek (Smashingmagazine, 2008) a CVS, SVN, Git, Mercurial, Bazaar, LibreSource, Monotone. VCS-ek zár alapú használata közben egy felhasználó zárat használva kizárólagos hozzáférést kap *egy vagy több fájlhoz, fájlrészlethez*. A lefoglalt fájl(oka)t/fájlrész(eke)t a saját munkaterületében szabadon módosíthatja, a munka befejezése után azok új verzióját visszatölti a központi VCS tárolóba, ezzel feloldja a hozzájuk kapcsolt zárakat.

A VCS rendszerek első generációja a fájlt tekinti a kollaboráció alapelemének és fájl szinten nyújt verziókezelést (Ignat, Norrie, 2008). Zárolási módban emiatt velük fájl szintű lefoglalást lehet elérni, ennél alacsonyabb szemcsézettségre nincs mód. Az ilyen VCS rendszerekkel lehetetlen olyan entitásokon valós idejű kollaborációt végezni, amelyek egyetlen fájlban tárolódnak. Mivel a workflow leírások döntő többsége egyetlen fájlban tárolódik, ezért ez a módszer csak körülményes módon, a workflow több fájlba való szétbontásával, majd szerkesztés utáni egyetlen fájlba való összeillesztésével lehetne megoldható.

Néhány új verziókezelő rendszer (például CVS) ugyan tud már fájlon belüli tartalmat is szinkronizálni, ezek azonban közvetlenül továbbra sem használhatóak kollaboratív workflow fejlesztésre. Az ilyen rendszerek a szinkronizációt *szöveges* fájlok soraira képesek elvégezni, *nem ismerik az irányított aciklikus gráfot, mint adatstruktúrát*. Ennek hiányában maguktól nem tudnak gráfot leíró fájltartalmakat csomópontok és élek halmazának tekinteni és szinkronizálni.

A dolgozat későbbi részeiben ismertetésre kerülő rendszer azonban ráépülhet fájlrészeket lefoglalni képes verziókezelő rendszerre.

2.2.2.2. Zárolás (locking)

Zárolás segítségével a megosztott entitás alacsonyabb szemcsézettségű felosztása érhető el. Ezzel biztosítható az egyidejű szerkesztés, amennyiben több felhasználó egyidőben más-más részt foglal le. Általában zár alapú konkurenciakezelést alkalmaznak a grafikus rajzolóprogramok (Newman-Wolfe et al, 1992)(Okada, Tanaka, 1995) és a CAD szerkesztőrendszerek (Fuh, Li, 2004)(Galli, Lou, 2000) kollaboratív változatai, és ugyan kevésbé domináns, de szintén erősen elterjedt a zárolás használata kollaboratív dokumentumszerkesztő környezetekben is (Haake, Wilson, 1992)(Noel, Robert, 2003). Dokumentumszerkesztő alkalmazásoknál jellemzően a kezdeti, piszkozatkészítés fázisában alkalmaznak inkább zárolást fejezet, bekezdés, szó, vagy akár betű szinten (Ignat, Norrie, 2008).

2.2.2.3. Szerializáció (serialization)

Akár a pesszimista, akár az optimista szerializáció kerül is használatra, segítségével minden gépen a konkurens utasításoknak azonos végrehajtási sorrendje jön létre. A módszer ezáltal biztosítja a példányok konvergenciáját és ok-okozati sorrendjét (1. és 2. probléma a CCI hármasból). Mindazonáltal *szerializációval nem garantált, hogy az időrendi sorrendbe szervezett utasítások valóban azt a hatást adják, amelyet a generáló felhasználók akartak.* Az egyidőben generált utasítások ugyanis az entitások olyan példányaira vonatkoznak, amelyek még nem tartalmazták a másik gépen generált utasítás hatását. Például a 2-4. ábrán a Gép 2-n generált B utasítás az entitás A által még nem módosított állapotára vonatkozik. Így, ha ezután valamilyen szerializációs protokoll segítségével az A, B utasítások ebben a sorrendben futnak le, akkor az entitásnak olyan állapota jön létre, melyet a Gép 2 felhasználójának nem állt szándékában létrehozni. Ez olyan esetekben okoz valóban problémát, amikor az entitáson nemkommutatív műveletek is definiáltak. *Mivel a workflow szerkesztés műveletei nem minden esetben kommutatívak (pl. egy csomópont megváltoztatása, a csomópont törlése), ezért workflow-k szintaktikai konzisztenciája kollaboratív fejlesztés közben szerializáció segítségével nem biztosítható. A pesszimista szerializációs módok ráadásul a felhasználók számára előre nem látható késleltetéseket is okozhatnak, és emiatt interaktív környezetekben ezek használata szintén nem szerencsés* (Munson, Dewan, 1996).

Szöveges dokumentumok kollaboratív szerkesztésére ugyan nemkommutatív „karakter beszúrás” és „karaktertörlés” műveletekből áll, ennek ellenére a szerializáció elterjedt az ilyen rendszerekben. Ezek általában a dokumentum véglegesítése során az intenzív csoportmunkához nyújtanak előnyöket (Ignat, Norrie, 2008). Velük ugyanis zárolás nélküli, gyorsan kivitelezhető, kötetlen együttműködés valósítható meg, ráadásul a szándék sérülésének problémája sem okoz nagy kellemetlenséget: egy-egy feleslegesen módosított karakter vagy szó gyorsan javítható.

Mivel a pesszimista szerializáció a felhasználó számára előre nem látható késleltetéseket okozhat, ezért interaktív használatra elsősorban az optimista módszerek preferáltak. Számos elméleti cikk és gyakorlatban is elérhető szövegszerkesztő csoportmunka környezet tartalmaz ilyen megoldásokat (Ellis, Gibbs, 1989)(Ressel et al, 1996)(Nichols et al, 1995)(Zaffer et al. 2001) (Sun, Ellis, 1998)(Suleiman et al, 1997)(Vidot et al, 2000)(Li, Li 2005)(Sun, Sun, 2006). Ezek döntő többségében művelet transzformációt is alkalmaznak, ugyanis a karakterbeszúrás és karaktertörlés műveletei egyetlen paraméterrel, a célkarakter sorszámát megadó értékkel dolgoznak. *Ennek a paraméterértéknek az eltolásával ezek az utasítások könnyen és gyorsan transzformálhatók.*

2.2.3. Kollaboratív workflow fejlesztés követelményrendszere

Az, hogy egy adott célra szánt csoportmunka alkalmazásban milyen konkurenciakezelési módszert érdemes használni a leendő felhasználók igényei, jellemzői, munkájuk során használt tipikus viselkedésformái alapján dönthető el. Az előzőekből kiderült, hogy akár a zárolás (alacsony, nem teljes workflow szinten), akár a szerializáció (elsősorban az optimista) képes biztosítani workflow alkalmazások kollaboratív szerkesztése közben a replikált példányok szintaktikai konzisztenciáját.

Mivel egy-két prototípustól eltekintve kollaboratív workflow fejlesztő rendszerek nem állnak rendelkezésre, emiatt egyértelműen nem definiálható, hogy hogyan használnák azokat a fejlesztői csoportok, melyik módszert találnák jobbnak. Ennek ellenére számos publikáció említ elképzelt kollaboratív workflow fejlesztést leíró szituációkat, forgatókönyveket (Khetawat et al, 1997)(Baker et al, 1999)(Filho, Hirata, 2002)(Held, Blochinger, 2009). Ezen forgatókönyvek, továbbá grid workflow alkalmazások fejlesztéséhez kapcsolódó eddigi tapasztalataim (Skouteris et al, 2008)(Krajcek, Kiss, Sipos, 2006)(Sipos et al, 2005)(Kacsuk et al, 2008) alapján összegyűjtöttem a kollaboratív workflow fejlesztéssel szemben általában támasztott igényeket, és ezek alapján fogok a továbbiakban javaslatot tenni egy valós idejű workflow fejlesztőrendszer felépítésére. Az itt felsorolt igények a legjellemzőbb felhasználói elvárásokat adják. Egy-egy speciális esetben lehetnek ezekhez képest eltérések, de az általam összegyűjtött igénylista a használat leggyakoribb eseteit lefedik:

1. *Csoport tagjainak szakmai háttere*: Általában, és egyre inkább tudományterületek képviselői, ipari, irodai vagy kutatási folyamatok szekértői és nem informatikai szakemberek. Ezen személyek klasszikus értelemben vett programozói ismeretekkel egyáltalán nem, vagy alig rendelkeznek, informatikai képesítésük sokszor egyáltalán nincs.
2. *Csoport tudásának homogenitása*: Egy csoporton belül nagyon változó lehet az egyes tagok ismerete. Elképzelhető, hogy különböző tudományágak, tudományterületek szakértői dolgoznak együtt azért, hogy egy interdiszciplináris folyamatot modellezzenek. Jelentős eltérés lehet a személyek informatikai ismereteiben és szemléletében is.
3. *Csoport mérete*: A csoportok mérete jól tippelhető, várhatóan kicsi, szinte kizárt hogy 10-15 fő fölé nőjön. Ezt számos, publikációkban előrevetített esettanulmány támasztja alá.
4. *Csoport tagjai közötti kapcsolat*: A csoport tagjai ismerik egymást (általában személyesen is), és erős a bizalom közöttük. Képesek egymást többféle kommunikációs csatornán át elérni (email, telefon, Skype, stb.).
5. *Munkamegosztás és a tagok munkájának fókuszáltsága*: Workflow fejlesztés során egy-egy felhasználó általában (de nem kizárólag) a gráf egy-egy részének kidolgozásáért felelős. Ez a rész lehet egyetlen csomópont, néhány egymáshoz kapcsolt csomópont, de akár a gráf egy teljes ága is. Az egyes részeken végzett módosítások nem, vagy csak minimális hatással vannak a workflow más részeire.
6. *Felhasználói bizalom*: A workflow-k és különösen a Web szolgáltatásokat, Grid szolgáltatásokat használó workflow technológiák viszonylag újkeletűek, egyelőre alacsony irántuk a bizalom. Workflow rendszerek elterjedését elősegíti a könnyen érthető szabályok szerint működő kollaboratív környezet, illetve ha a rendszer

megakadályozza, hogy a felhasználók egymás munkáját felülírják, egymásnak adatvesztést okozzanak.

7. *Kollaborációk hossza*: Egy-egy felhasználói szakasz (user session) hossza általában órákban, mint percekben mérhető. A felhasználói szakaszokból összeálló szerkesztési szakaszok szintén pár órás, esetleg napos nagyságrendbe esnek. Különösen hosszúra nyúlhat egy szakasz, ha az valamely olyan gráf csomópont hozzáadását tartalmazza, amely új szolgáltatás kifejlesztését igényli (pl. új Grid job leprogramozása; a workflow-ba illeszthető Web szolgáltatás beüzemelése).
8. *Módosítások értéke*: Egy-egy felhasználói szakasz során végzett fejlesztési tevékenység minden esetben kézzel végzett munka, ezért megismétlése jelentős többletmunkával jár, *szűk határidő esetén ismétlésre, korrekcióra esetleg egyáltalán nincs lehetőség*. Egy felhasználó munkájának eldobása, elvesztése, utólagos kompenzációja a rendszer, vagy egy másik konkurens felhasználó változtatásai miatt nagy értékű kárt jelent.
9. *Valós idejű kollaborációra való igény*: Kollaboratív szerkesztés esetén több felhasználó egyidőben szeretne szerkeszteni egy gráfot. Vannak helyzetek, amikor a felhasználók szeretnék egymás munkáját valós időben látni (a még nem elmentett változtatásokat is), és saját még nem elmentett munkájukat megmutatni, de akadnak olyan szituációk is, amikor erre nincs igény, sőt akár zavaró lehet a nem véglegesített változtatások megjelenítése.
10. *Megosztott entitás integritása*: Mivel a workflow alkalmazások futtatási fázissal is rendelkeznek, ezért bizonyos konzisztencia feltételeket a szerkesztés végén mindenképp teljesíteniük kell. A konzisztencia feltételek megőrzése kulcsfontosságú kollaboratív workflow fejlesztés során is. (A konzisztencia kezelés témakörével külön, az 3. fejezetben foglalkozom majd részletesen.)

Ezen jellemzők és kritériumok alapján a *zár alapú konkurenciakezelést* tartom a legalkalmasabbnak valós idejű kollaboratív workflow fejlesztőrendszerek számára. Mivel egy workflow gráfban a csomópontok és az élek egymástól függetlenül szerkeszthető komponensek, ezért természetesen adódik *a csomópont és él szintű zár granularitás*, amely képes egyazon gráfon több felhasználónak egyidejű szerkesztést biztosítani. Ez a szint a felhasználók számára könnyen érthető és nagy vizuális információtartalmú.

Mivel a workflow fejlesztés erősen tudásintenzív folyamat, ezért egy-egy workflow-n végzett változtatás értéke nagy. A zárolás alapú módszerek közül csak a pesszimista mód képes garantálni, hogy egyetlen workflow fejlesztési folyamat sem kerül más, konkurens felhasználók munkája miatt abortálásra. Pesszimista zárolással minden felhasználó számára garantálható, hogy a változtatásai a végleges workflow-ba kerülnek, azokat nem kell majd más, konkurens felhasználó változtatásai miatt utólagosan módosítani, vagy eldobni[5].

A zárak igénylése és feladása teljesen elosztott módon történhet úgy, hogy minden felhasználó a saját szerkesztőkörnyezetén keresztül tud komponenseken zárat igényelni illetve lemondani. A zárolási kérések elbírálását végezheti egy központi szolgáltatás, de akár a csoport egy vagy több kitüntetett szerepkörben lévő tagja is. Előbbi eset demokratikus csoportfelépítést jelent, ahol mindenki azonos eséllyel pályázik zárolásra, utóbbi esetben a csoport „vezetői” személyes preferenciáik alapján adhatnak és tagadhatnak meg lefoglalásokat.

[5] Ezt valójában a pesszimista lockolás is csak abban az esetben garantálja, ha a felhasználó módosításai nem okozzák a workflow *szemantikai inkonzisztenciáját*. A szemantikai konzisztencia témakörét az értekezés 3. fejezetében részletesen tárgyalom.

Mivel a csoport tagjai ismerik, ráadásul jól ismerik egymást, bíznak egymásban, ezért nem kell rosszindulatú résztvevőktől tartani, általában nincs esélye annak, hogy valaki mások kiéheztetése miatt hosszú ideig tart zárolva részeket.

A következő, 2.3 részben ismertetem a szorosan vett kollaboratív workflow és gráf fejlesztéshez kapcsolódó szakirodalmakat, azután a 2.4 fejezetben részletezem az általam javasolt pesszimista zárolási módszert használó valós idejű kollaboratív workflow fejlesztőrendszer működését.

2.3. Kollaboratív workflow fejlesztő környezetek – szakirodalmi áttekintés

A workflow alkalmazások az utóbbi években váltak nagyszámú felhasználók számára is elérhetővé. Amíg csak néhány, ráadásul informatikai képesítéssel rendelkező személy volt workflow fejlesztői státuszban, addig ritkán volt szükség csoportmunka-támogatásra. Ha pedig szükség volt rá, akkor az megoldható volt VCS rendszerekkel úgy, hogy a fejlesztők egymás után, egyre újabb és jobb verzióit állították elő a workflow-knak. Az utóbbi körülbelül tíz évben azonban a workflow technológiák felhasználó szélesebb rétege számára váltak elérhetővé, ezzel párhuzamosan egyre nagyobb igény generálódott kollaboratív workflow-k iránt.

A szakirodalomban az utóbbi években megjelent néhány olyan publikáció, amely kifejezetten kollaboratív workflow-khoz, illetve csoportos gráf-fejlesztéshez kapcsolódik. A GroupGraph rendszer gráfok valós idejű, egyidőben több felhasználó általi szerkesztését teszi lehetővé (Filho, Hirata, 2002). Ezek a gráfok azonban rendszermodellezési célokból, csak mint matematikai struktúrák jönnek létre, nincs futtatási fázisuk, mint a workflow gráfoknak. Mivel a gráfok tetszőleges rendszert modellezhetnek, a GroupGraph rendszer csak a gráf szintaktikai konzisztenciájával törődik, magasabb szintű, szemantikai konzisztencia védelmet nem tartalmaz. A szintaktikai konzisztenciát a rendszer az általam is javasolt szintén pesszimista zár használattal biztosítja, viszont zárakat csak csomópontokra enged rátenni, és egyidőben egy felhasználó csak egy csomópontot zárolhat. Ez ugyan megelőzi a 2.5.1 részben tárgyalásra kerülő holtpont (deadlock) helyzetek kialakulását, viszont nem tesz lehetővé hatékony munkát akkor, amikor egyidőben több csomópontnak kell hasonló adatokat beállítani, vagy amikor az élek is paraméterezhetők és azok paramétereinek módosítására van szükség. A GroupGraph rendszer egy prototípussal rendelkezett 2002-ben, azóta az sem elérhető.

A „kollaboratív workflow” kifejezést 2005-ös cikkemben használtam először olyan workflow alkalmazásokra, amelyek több ember fejlesztése nyomán jöttek létre (Sipos et al, 2005). Az akkori cikk már javasolta a komponens szintű, pesszimista lefoglalást a konfliktushelyzetek kezelésére. 2005 óta született erre a cikkemre több olyan hivatkozás, amely a módszer használhatóságát igazolja különböző esettanulmányokon keresztül.

(Friese et al, 2006) munkája az anyagtudomány és anyagmegmunkálás területén belül ismerteti a kollaboratív workflow módszert olyan számításigényes szimulációk és minőségbiztosítási szempontból fontos kalkulációkra, melyek BPEL workflow-k segítségével írhatók le.

Held és Blockinger Hobbes rendszere (Held, Blochinger, 2008)(Held, Blochinger, 2009) alapvetően szintén ennek a koncepciónak egy BPEL nyelvet használó implementációja. A Hobbes az egyfelhasználós, BPEL nyelven épített workflow-k fejlesztési folyamatának időnkénti többfelhasználósra való kiterjesztését valósítja meg úgy, hogy a workflow tulajdonosa a gráfot bizonyos időtartamra több felhasználó között szétoszthatja. Ezen szétosztás az általam javasolt megoldáshoz hasonlóan komponens szintű zárak segítségével történik, viszont központosított módon, vagyis kizárólag a workflow eredeti tulajdonosa foglalhat le komponenseket. Ez sajnos a

vezető elérhetetlensége esetén hosszú várakozásokat okozhat. A zár lemondást viszont a gráf tulajdonos mellett a zárat birtokló fejlesztők is elvégezhetik. Az általam javasolt, automatikus zár kezelő módszer jobb lehet heterogén tudással rendelkező fejlesztői csoportok esetén, ahol a csoport tagjai közül senki sem rendelkezik elég információval ahhoz, hogy a gráf felosztást egymaga elvégezze. BPEL-ben ráadásul csak fagráfok készíthetők, ilyenkor a lefoglalás ugyanúgy működik, mint a szintén fagráfként ábrázolható XML vagy szöveges dokumentumokon.

Phung et al. cikke (Phung et al, 2009) kollaboratív workflow-k orvosi célokra való használatáról ír. Az esettanulmány szerint több szakterület medikusa dolgozhat együtt vizsgálatok és diagnózisok elkészítésén. A rendszer Grid szolgáltatásokat használna a workflow ütemezésére és a futáshoz szükséges erőforrások összegyűjtésére. Ugyan ő is zárolást említi, mint a kollaboratív fejlesztés során legcélszerűbben használható konkurenciakezelési módot, a döntését nem indokolja, és a módszer használatának részleteit sem tárgyalja. A cikk csak egy architektúra komponenseinek nagyvonalú leírása, mint kutatási eredmények ismertetése.

Baker et al. munkája (Baker et al, 1999) felvázol egy olyan infrastruktúrát (Collaboration Management Infrastructure), melyben – elsősorban krízishelyzetek kezelésére – különböző területek szakértői tudnak futtatható workflow alkalmazásokat létrehozni. A megoldás azonban már előre elkészített workflow-k bemenő adatainak a megadását jelenti, és nem teszi lehetővé a valós idejű gráf módosítást. A rögzített gráfokból kifolyólag az azokhoz való hozzáférés is fix, előre tudható hogy kinek melyik gráfrész számára kell bemenő adatokat definiálnia, emiatt nincs is szükség konkurenciakezelésre.

Számos olyan publikáció található, amelyben a „kollaboratív workflow” kifejezést egyszerűen olyan workflow alkalmazókra használják, amelyek futás közben több intézethez köthető szoftver és/vagy emberi szolgáltatást vesznek igénybe, vagyis futás közben „több intézményen átívelnek”, futásukkal intézmények közötti együttműködést valósítanak meg. A „kollaboratív” jelző ilyen módú használatának oka, hogy eredeti értelmezésben a workflow egyetlen intézményen belüli munkafolyamatot jelentett, és ekkor még a „kollaboratív” jelző a több intézménynél elvégzett feladatok egyetlen munkafolyamba vonására utalt (Mecella et al, 2001). Például Khetawat et al. munkája (Khetawat et al, 1997) olyan workflow-kat ismertet kollaboratívként, melyekben egy-egy csomópont végrehajtása egy adott intézményben dolgozó csoport szolgáltatásának igénybevételét jelenti, így a több csomópontos workflow vállalatok közötti együttműködést koordinálnak. Zhao et al munkája (Zhao et al, 2006) ezzel a módszerrel több intézmény lokális rendszerében definiált workflow alkalmazását integrálja egyetlen, intézményeken átívelő workflow-vá. Müller et al cikke (Müller et al, 2006) hasonló megoldást mutat be Web szolgáltatás alapon nagyméretű városfejlesztési projektek ütemezésére. Egy magyar vonatkozású publikáció, hasonló integrációs módszerrel grides workflow-k egymásba ágyazását ismerteti (Kukla et al, 2008). Mivel egy grides workflow eleve több intézmény szolgáltatását használja, ezért ennek a munkának elsősorban a különböző technológiák közötti kompatibilitás megteremtése a célja.

Ryall et al. munkájában (Ryall et al, 1997) a számítógépet tekinti a gráf rajzoló felhasználó együttműködő partnereként, akivel együtt kétfős kollaboratív csoportot alkotnak. A gráf itt csak modellezési célból születik, nincs futtatási fázisa. A gráf szerkesztésében a gép csupán a már meglévő csomópontok mozgatásával (pozíciójának változtatásával) vesz részt, komponensek hozzáadásához, elvételéhez csak a felhasználónak van joga. A pozícióváltoztatással a gép előre megadott kritériumokat elégít ki, például biztosítja, hogy pipe-line típusú gráfok (elágazás nélküli gráfok) csomópontjai egyenlő távolságban és egy egyenes mentén legyenek egymástól, vagy hogy egy csillag struktúrában minden él azonos hosszúságú legyen, és a szomszédos élek azonos szöget zárjanak be egymással. Mivel a felhasználó és a számítógép a gráf egymástól

független paramétereit változtathatja, ezért nincsenek a rendszerben konfliktushelyzetek, nincs szükség konkurenciakezelésre.

Nagy elemszámú objektumhalmazból építkező tudásbázisok létrehozása szintén gráf építési folyamatként fogható fel. Ilyen rendszerekben sok ezer, vagy akár még annál is több objektum él egyidőben, és ezeket az objektumokat emberek és szoftverek töltik be. A tudásbázis építőinek feladata, az objektumok között a való életben meglévő kapcsolataikat reprezentáló élek berajzolása. Loren és Wroblewski (Terveen, Wroblewski, 1990), illetve Bollacker et al. (Bollacker et al, 2008) munkáikban olyan kollaboratív tudásbázis építő rendszereket írnak le, melyekkel egyidőben több felhasználó is dolgozhat. Ezek a szoftverek azonban csak azt teszik lehetővé, hogy egyidőben több különböző gráf szülessen meg ugyanazon objektumok felhasználásával, nem alkalmasak arra, hogy egyazon gráf építésébe több felhasználó bekapcsolódjon. A CSCW terminológiája szerint tehát ezek a megoldások nem valós idejű csoportmunka rendszerek, ellentétben az általam javasolt valós idejű gráf szerkesztő módszerrel.

Vasko és Dustdar munkája (Vasko, Dustdar, 2009) az elosztott, gráffal modellezhető internetes alkalmazások egy új válfajára, az úgynevezett „mashup”-okra definiál egy kollaboratív létrehozási módot. A mashup-ok internetes szolgáltatásként közzétett, egyszerű felületen keresztül futtatható, az interneten bárki által elérhető ingyenes közösségi szolgáltatásokból épített workflow-k (pl. GoogleMaps, Facebook, ingatlan-adatbázisok jelenhetnek meg benne komponensként). A Vasko és Dustmar által javasolt kollaborációs módszer három különböző szerepkörben dolgozó felhasználó számára tesz lehetővé mashup definiálást: létrehozó (creator), tulajdonos (stakeholder), adminisztrátor. Mivel azonban a szerepkörökhöz rendelt, a mashup-on végrehajtható műveleteket tartalmazó halmazok diszjunktak, ezért az egyidejű, szinkronizálatlan szerkesztés konkurenciakezelés nélkül is megvalósítható. A kollaboratív workflow fejlesztés dinamikus szerepkör allokációt igényel, a gráf definiálása közben egy adott részhez időről időre több embernek is hozzá kell tudni férnie, ezért előfordulhatnak azonos műveletek miatti konfliktusok.

2.4. Kollaboratív workflow szerkesztés zárolás segítségével

Az általam javasolt pesszimista zárolás alapú konkurenciakezelés használó kollaboratív fejlesztőrendszer a 2-3. ábra jobb oldalán látható replikált architektúra segítségével képes alacsony válaszidőket biztosítani a felhasználóknak:

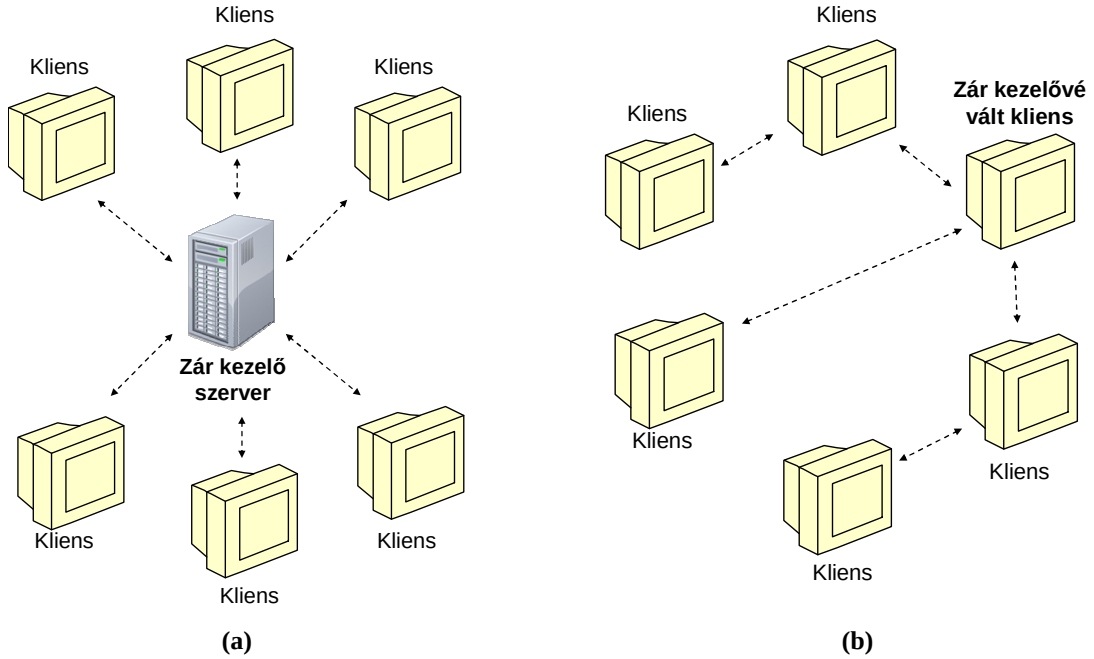
- A helyi felhasználó által végzett gráf módosítási műveletek azonnal lefuttathatók.
- A távoli utasítások zárolás használata miatt nem okozhatnak ütközést a helyi műveletekkel, ezért azok is azonnal futtathatók.

A zárolás a kliens oldali szerkesztőkörnyezetek és a zár kezelő rendszer között két utasítás bevezetését igényli:

- `LockRequest(R)`: lefoglalást kezdeményez a workflow R komponenshalmazán. R lehet egy vagy több él *és* egy vagy több csúcs: $R \subseteq (V \cup E)$. (R – request)
- `UnlockRequest(R')`: a workflow R’ zárolt komponenshalmazán a foglalás lemondását kezdeményezi. R’ a korábban lefoglalt R komponenshalmaz átszerkesztett változata.

Zár kezelésre (elfogadásra, megtagadásra, felszabadításra) automatikusan működő szoftverszolgáltatás használatát javaslom, mivel ez adhatja a legobjektívebb elbírálást, ráadásul ez eredményezhet leggyorsabban választ. Ezzel a zárat használó rendszerek egyik hátránya, a hosszú zárolási kérés elbírálási idők is csökkenthetők.

Maga a zár kezelést végezheti egy, az architektúra dedikáltan erre a célra üzemeltetett gépe (zár kezelő szerver), de a munkakört dinamikus módon betöltheti bármelyik kliens gép is. Míg a dedikált gép használata fix szereposztású kliens-szerver architektúrát és csillagszerű topológiát eredményez (ld. 2-5. ábra (a) része), addig a kliens használata dinamikusan leosztott szerepeket, és emiatt bonyolultabb protokollokat igényel ld. 2-5. ábra (b) rész. Kliens használata esetén elképzelhető például, hogy minden szerkesztési szakasz során a workflow szerkesztését elsőnek elkezdő kliens válik zár elbírálóvá, ő kezeli a teljes gráfon a zárakat (nem csak a saját részgráfján), és minden később csatlakozó először fel kell hogy ismerje, hogy már van zárolási felelős, hozzá kell küldeni a foglalási és zár felmondási igényeket. Fontos megjegyezni, hogy a gépeknek csak a saját szerepkörük megtalálásáig kell peer-to-peer alapon működniük, utána már itt is kialakulhat a csillagszerű topológia melyben a zár kezelő áll a középpontban.



2-5. ábra: Zár kezelés (a) dedikált szerver és (b) menedzserré váló kliens gép segítségével.

Mivel a dedikált zároló szerver használata egyszerűbb protokollokkal is megvalósítható, ezért inkább azt tartom célszerűbbnek a zárak kezelésére (vagyis a 2-5. ábra (a) részén látható megoldást). *A dolgozat további részét képező munka viszont független ettől a döntéstől, bármilyen szerepkör leosztás is van használatban, a további megállapítások, megoldások mind ugyanúgy használhatóak.*

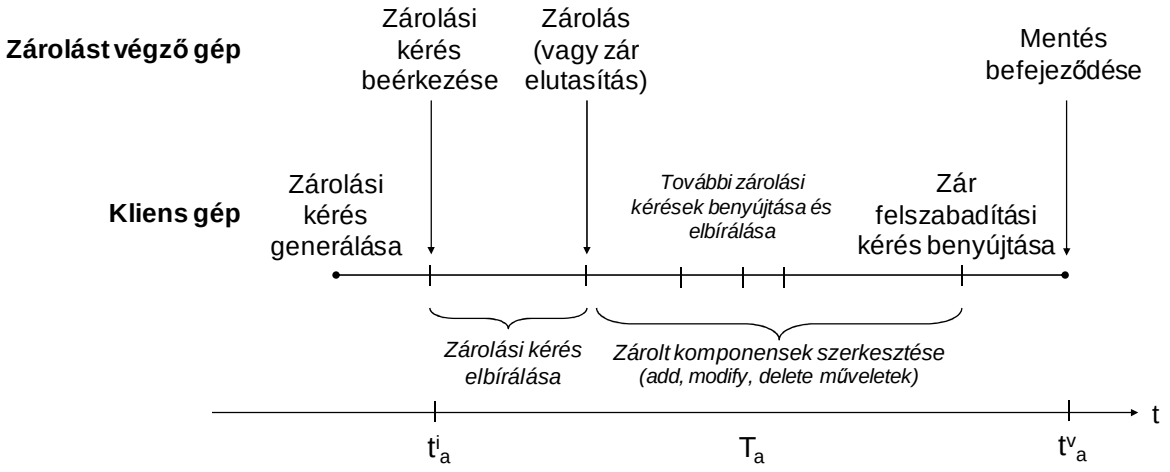
A fent felsorolt `LockRequest` és `UnlockRequest` üzenetek tehát a felhasználói szerkesztőkörnyezetektől a zár elbíráló szerepkörrel bíró számítógéphez továbbítódnak. Pesszimista zár használat következtében egy megosztott komponenst le kell foglalni, mielőtt az megváltoztatható. Emiatt a `Modify` és `Delete` utasításokat mindig megelőzi egy `LockRequest` üzenet. Az üzenetküldést kezdeményezheti explicit módon a felhasználó – például azzal, hogy a fejlesztőfelületen kijelöli a módosítandó workflow komponenseket, kérve a lefoglalásukat – de kiváltódhat a kérés implicit módon is magával a módosítással, vagy törléssel. Ez utóbbi variáció használatakor a módosítás vagy törlés addig ténylegesen nem hajtódik végre, amíg a környezet zárat nem szerzett az érintett komponensen.

Új komponensek létrehozása esetén (`Add` műveletek) vagy implicit lefoglalásra van szükség, vagy teljesen elmaradhat a lefoglalás:

- *Implicit lefoglaláskor* a komponens létrejötte után azonnal lefoglalásra kerül, nem kell külön foglalást kezdeményeznie rajta a felhasználónak. Mivel a komponens új, ezért az nem lehet senkinek sem lefoglalva, a zárolás mindenképp elfogadásra fog kerülni, az elbírálás eredményét akár nem is szükséges megvárni.
- Amennyiben az új komponenst valós időben nem osztja meg a felhasználó a többi együttműködő partnerrel, akkor *a lefoglalás teljesen el is maradhat*. Mivel a komponens csak egyetlen fejlesztőkörnyezetben él, ezért rajta nem lehetnek több felhasználó által generált, konfliktusban lévő műveletek. Az új komponenst viszont a felhasználói szakasz végén mindenképp propagálni kell a többi felhasználó felé, hogy az bekerülhessen a workflow végleges, mindenki által látható változatába.

Egy felhasználó workflow-n végzett munkáját a workflow megnyitás (`Open workflow`), és workflow mentés (`Save workflow`) vagy workflow eldobás (`Ignore changes`) utasítások foglalják keretbe. *A felhasználói szakasz felfogható egy olyan tranzakciónak, melyben a megnyitás a tranzakció indítása, a mentés a tranzakció commit-tal, az eldobás a tranzakció abortálással való lezárása. A tranzakció-indítás és lezárás között a 2.1. részben felsorolt, csomópont és él hozzáadást, módosítást és törlést végző utasítások sorozata található.* (ld. 2-6. ábra). Az egyszerűbb hivatkozhatóság kedvéért a továbbiakban egy felhasználói szakaszt „szerkesztési tranzakció”-nak, vagy egyszerűen „tranzakció”-nak fogom hívni. Egy *a* szerkesztési tranzakcióra a T_a jelölést, indulásának időpontjára a t^i_a jelölést, míg befejeződésének időpontjára a t^v_a jelölést fogom használni. (i – indul, v – végződik)

Egy tranzakció élete azonban több szakaszból áll. Fontos látni, hogy a mit értek kezdeti és végponton. Ahogy a 2-6. Ábrán látható, a tranzakció indulási pillanata a zárolási kérésének elbírráló rendszerbe való megérkezésére utal. Ezen pillanat, és a valói zárolás között eltelt szakasz a kérés elbírálása. Feltételezem, hogy az elbírálási szakaszban tranzakciók nem előzhetik meg egymást – vagyis amelyik tranzakció kérése hamarabb érkezett be, az hamarabb is kap választ. Mivel vizsgálataim függetlenek a konkrét implementációs környezettől, ezért sem a tranzakció indítási kérések, sem a tranzakció lezárási kérések elbíráló rendszerbe való eljuttatásának ideje nem ismert. Mivel a tranzakciók futását csak a kéréseik szerverhez való megérkezésétől tekintem, ezért ezt a bizonytalanságot kiküszöböltem. A tranzakció lezárása a változtatások elmentésének befejeződéséhez, és nem a kliens oldali tevékenységhez kötött. A kommunikációs késleltetéseket így itt is sem jelentenek gondot.

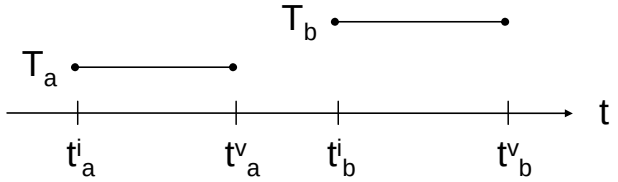


2-6. ábra: Egy T_a workflow szerkesztési tranzakció teljes életciklusa.

A könnyebb értelmezhetőség miatt átveszek további, az adatbázis rendszerek és a csoportmunka rendszerek témakörében széles körben használt definíciókat:

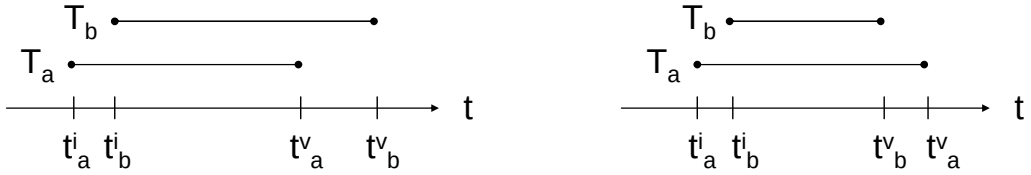
Def. 2: Egy T_a workflow szerkesztési ***tranzakció hamarabb indul*** egy T_b tranzakciónál, (jelölésben $T_a < T_b$) ha indulási időpontjaikra (zárolási kéréseik beérkezésének időpontjára, t^i_a-ra és t^i_b-re) igaz, hogy $t^i_a < t^i_b$. (Ld. 2-7. ábra.)

Def. 3: Egy T_a workflow szerkesztési ***tranzakció megelőz*** egy T_b tranzakciót, (jelölésben $T_a \rightarrow T_b$) ha indulási időpontjaikra (t^i_a-ra és t^i_b-re), és befejeződésük időpontjaira (változtatásaik mentésének időpontjaira, t^v_a-re és t^v_b-re) igaz, hogy $t^v_a < t^i_b$. (Ld. 2-7. ábra.)



2-7. ábra: A T_a tranzakció megelőzi a T_b tranzakciót: $T_a \rightarrow T_b$
(Emiatt természetesen a T_a tranzakció hamarabb indul, mint a T_b tranzakció: $T_a < T_b$)

Def. 4: A T_a és T_b workflow ***szerkesztési tranzakciók egyidejűleg futnak***, (jelölésben $T_a \parallel T_b$) ha indulásuk és befejeződésük időpontjaira igaz, hogy $t^i_b < t^v_a < t^v_b$ vagy $t^i_a < t^v_b < t^v_a$ (Ld. 2-8. ábra.)



2-8. ábra: Két tranzakció egyidejű futásának lehetőségei.

A tranzakciókra kimondott egyidejű futás reláció megfelel az Ellis és Gibbs „átfedésben van” (overlaps) relációjának (Ellis, Gibbs, 1989) és Sun és Ellis „konkurens” relációjának (Sun, Ellis, 1998.). (Jelölésre előbbi az „o” jelet, utóbbi a nálam is alkalmazott „||” jelet használja.) Ők azonban zárolás nélküli szerializációt használnak dokumentumok szerkesztésére, így a relációkat nem tranzakciók, hanem önmagukban álló műveletek között értelmezik. A zárolás nálam a szerkesztési tranzakciók között eredményez szerializációt.

Def. 5: Az olyan workflow szerkesztő rendszert tekintem ***egyfelhasználós rendszernek***, amelyben egyazon gráfon futó $T_1, T_2, ..., T_n$ szerkesztési tranzakciókra igaz, hogy $\forall i, j \in \{1,...n\}, i \neq j \Rightarrow T_i \rightarrow T_j$ vagy $T_j \rightarrow T_i$. (Az ilyen rendszerben egyidőben nem él több szerkesztési folyamat ugyanazon a gráfon.)

Def. 6: Az olyan workflow szerkesztő rendszert tekintem ***kollaboratív rendszernek***, amelyben egyazon gráfra vonatkozó $T_1, T_2, ..., T_n$ szerkesztési tranzakciók esetén előfordulhat, hogy $\exists a, b : T_a \| T_{b;}\ T_a, T_b \in \{T_1, T_2, ..., T_n\}$.

A zárolás alapú konkurenciakezelés kollaboratív rendszerekben védelmet nyújt a konfliktusban álló műveletek egyidejű futása ellen. Megosztott entitást szerkesztő T_1 tranzakció

O_1 művelete akkor van konfliktusban egy vele egyidőben futó T_2 tranzakció O_2 műveletével, ha (Connolly, Begg, 2004):

- Ugyanarra az adatelemre vonatkoznak (esetünkben ugyanarra a csomópontra vagy ugyanarra az élre) *és*
- Az adatelem végső állapota attól függ, hogy milyen sorrendben kerülnek végrehajtásra a műveletek.

A definíció értelmében a `Modify`, `Delete` és `Add` műveletek okoznak konfliktust, az olvasás viszont nem.

A pesszimista zárolás alapú működés menetét a 2-9. ábrán látható UML szekvencia diagram illusztrálja. A szerkesztési szakaszban két tranzakció és a zár kezelő szerver vesz részt. Kezdetben mindhárom gépen ugyanaz a workflow képe: a gráf 5 csomópontot (*A*, *B*, *C*, *D*, *E*), és 5 élt tartalmaz (*ab*, *ac*, *bd*, *be*, *ce*). Ez a közös workflow változat most az egyszerűség kedvéért a zár kezelő szerveren van. (Lehetne egy különálló gépen is, amelyhez a tranzakciókat futtató gépeknek és a zár kezelő szervernek hozzáférése van.)

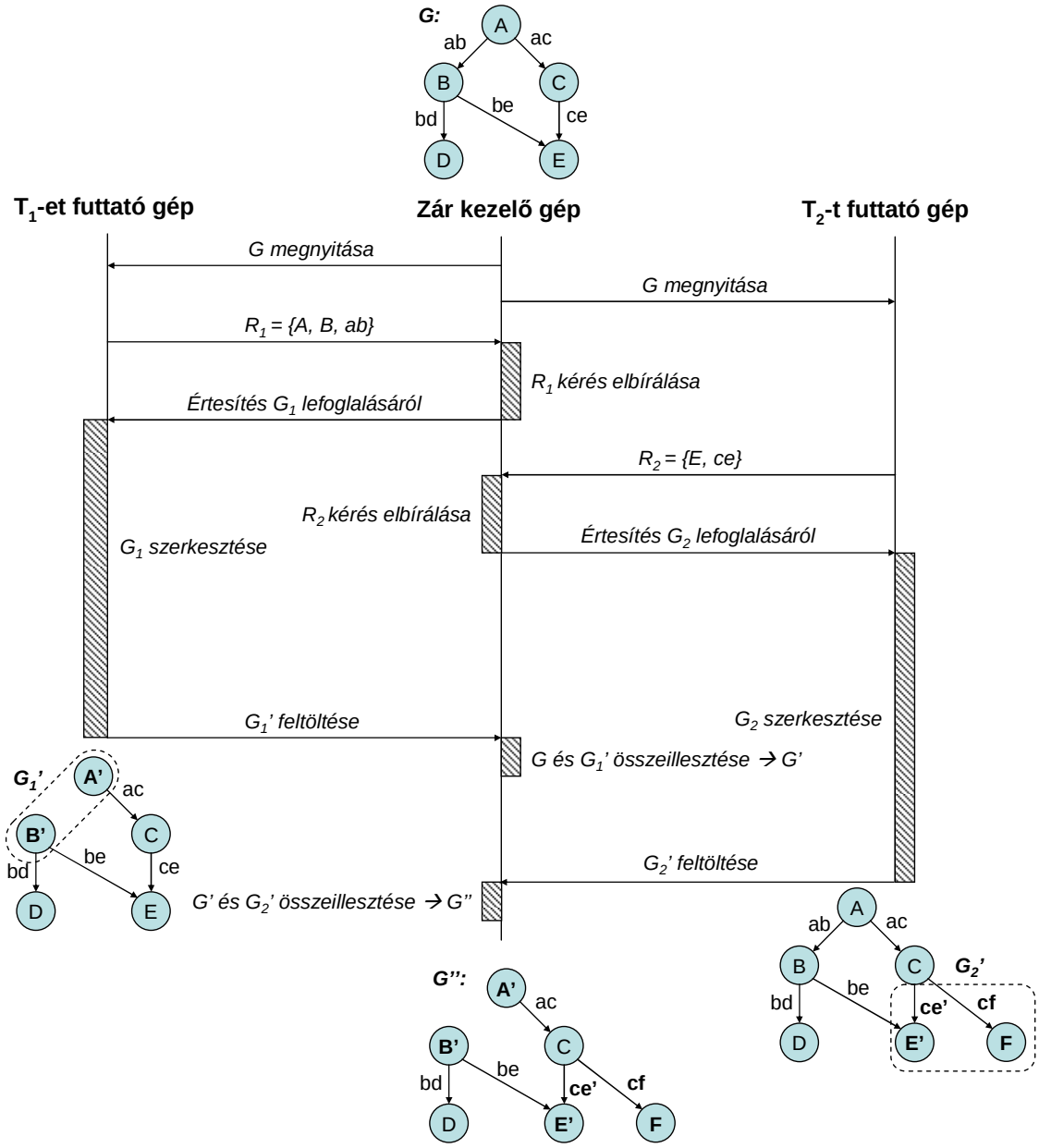
T_1 a gráf *A* és *B* csomópontjait, valamint az *ab* élt akarja módosítani. Lefoglalásra kéri ezért ezeket: R_1={*A*, *B*, *ab*}. *A kérés elbírálása során a szerver egyszerűen azt vizsgálja meg, hogy a komponensek le vannak-e már foglalva*[6]. Mivel nincsenek még lefoglalva, ezért a zárolás megtörténik: G_1={*A*, *B*, *ab*}. Erről a T_1-et futtató gép bizonyosan tudomást szerez, és engedi G_1 komponenseinek módosítását. A T_2-t futtató gép szintén értesülhet erről, (akár a szerver által kezdeményezett „push", akár a kliens által kezdeményezett „pull" módú kommunikációval. Ez célszerű is, mivel így a kollaboratív partnerek láthatják, hogy T_1 milyen komponenseket birtokol, vagyis növekszik a felhasználói tudatosság (awareness).

Miközben G_1-et szerkeszti a T_1 tranzakció tulajdonosa, a másik gép is lefoglalási kérést intéz a szerverhez. Ő a *ce* és *E* komponenseket kéri: R_2={*ce*, *E*}. Mivel ezek nincsenek még lefoglalva, megkapja őket: G_2={*ce*, *E*}. Erről szintén beállítástól függően T_1 értesülhet. A két tranzakció ekkor egyidőben, egyazon gráf különböző részein dolgozik. T_1 módosításaiként az *A* és *B* csúcsok új verziója (*A'* és *B'*) jön létre, az *ab* él pedig törlésre kerül. Ezek a változtatások valós időben megoszthatók a többi felhasználóval akár a szerveren keresztül, akár közvetlen kommunikációval. Ugyan lehet, hogy a változtatások később eldobásra kerülnek, ez akkor sem okozza más felhasználók munkájának elvesztését, mivel ezeket a komponenseket mások a T_1 lezárásáig nem szerkeszthetik.

A változtatások mentését T_1 az általa módosított komponenseknek, (vagy akár a teljes komponenshalmazának) a szerverhez való továbbításával kérheti. Ekkor a szerver előállítja az új permanens gráf verziót (G') az eredeti gráfból és a módosított komponensekből. *Ez a megváltoztatott (módosított vagy törölt) komponensek lecserélésével és az újonnan létrehozott komponensek hozzáadásával valósítható meg.* Ez a lépés azonban nem veszi figyelembe a T_2 által már elvégzett, de még nem elmentett változtatásokat. Amennyiben T_1 változtatásai korábban valós időben nem lettek propagálva a többi kollaboratív résztvevő számára, akkor az új gráf előállítása kapcsán most erre egyben nyílik lehetőség. Ez azonban opcionális.

T_2 a saját lefoglalt komponensei közül módosítja az *E* csúcsot és a *ce* élt (létrehozva *E'*-t és *ce'*-t), továbbá hozzáad a gráfhoz egy *F* csúcsot és egy *cf* élt. Változtatásai véglegesítése érdekében ő is a szerverhez küldi a módosított részgráfját (vagy csak a módosított és az újonnan létrehozott komponenseket), melyek bekerülnek a végleges változatba (G''), amely ezután a következő szerkesztési szakasz, vagy a futtatási fázis kiindulópontját adja.

[6] A 3. fejezetben, a workflow szemantikai konzisztenciájának a megőrzése miatt ennél bonyolultabb vizsgálatokról lesz majd szó. Egyenlőre ezt az egyszerű vizsgálatot használom csak.



2-9. ábra: Zárolás segítségével felosztott workflow csoportos fejlesztésének folyamata.

A rendszer működésével kapcsolatban a következő megállapításokat tartom érdemesnek külön kiemelni:

- Az egyes tranzakciók gráfon végzett változtatásai akár valós időben, akár a tranzakció lezárásakor propagálódhatnak a kollaboratív csoporton belül.
- A zárolási kérés elbírálása során a legegyszerűbb megoldás csak a már meglévő, és a foglalásra kért komponenshalmazok közötti átfedést tekinti. Átfedés esetén tiltja a lefoglalást, ellenkező esetben engedi azt. A dolgozat további részeiben ennél intelligensebb zároló elbíráló algoritmusokat ismertetek majd.
- Egy tranzakció sikeres lefoglalás esetén egy komponenshalmazt kap. Ez a matematikai definíció szerint nem részgráf, ugyanis lehetnek benne olyan élek is,

amelyek nem kapcsolódnak csomóponthoz. (Pontosabban a komponenshalmazon kívüli csomópontokhoz kapcsolódnak.) Ennek ellenére az egyszerűség miatt a továbbiakban „részgráf”-nak fogom hívni a tranzakciók számára zárolt komponenshalmazokat.

2.5. A rendszer tulajdonságai

Ebben a részben megvizsgálom a rendszer néhány, konkurenciakezelés szempontjából fontos alapjellemzőjét.

2.5.1. Holtpont (deadlock)

Mivel a rendszer alapvetően az adatbázis rendszereknél alkalmazott kétfázisú zárolás módszerét használja, és ezen felül megengedi egyidőben több komponens egyazon felhasználó számára történő lefoglalását, ezért kialakulhat holtpont (deadlock) (Bernstein, Goodman, 1987). Előfordulhat, hogy egy A felhasználó lefoglal egy *a* részgráfot, B felhasználó egy *b* részgráfot, és az egymás által birtokolt komponensek felszabadulására várva nem engedik el a saját részüket. Ezen az sem segít, ha egy lefoglalt részgráfot egy felhasználó nem terjeszthet ki, ugyanis akkor valószínűleg elengedés és azonnali újrafoglalással próbálkoznának.

Zárolást alkalmazó csoportmunka rendszerekben az ilyen szituációk előzetes felismerése, vagy utólagos feloldása tipikusan nem szoftver, hanem a felhasználók feladata (Sun, 2002)(Newman-Wolfe et al, 1992). A felhasználóknak vagy a csoportmunka által biztosított, vagy valamilyen azon kívüli csatornán keresztül kell kapcsolatba lépniük egymással, megtudniuk hogy kinek mi a szándéka a birtokolt komponenseivel, és kideríteni, hogy ki hajlandó mások érdekében elengedni a saját komponenseit. *Mivel csoportmunka rendszerekben emberek és nem programok a tranzakciók végrehajtói, ezért van lehetőség manuális holtpont-felismerésre és -feloldásra.*

Kollaboratív workflow fejlesztőrendszernek minimálisan a következő szolgáltatásokkal kell segítenie a felhasználókat a holtpont helyzetek kezelésében:

- A felhasználói környezetekben jelenjen meg, hogy a gráf mely komponensei éppen kinek a számára vannak lefoglalva.
- Legyen biztosítva valamilyen kommunikációs csatorna, melyen keresztül a felhasználók egymást elérhetik.

Munkámban nem cél a felhasználói tudatosság (awareness) témakörökében újat alkotni, ezek a szolgáltatások meglévő eszközök workflow fejlesztő környezetbe integrálásával biztosíthatók. Ilyen beilleszthető szolgáltatáscsomagot nyújt például a MAUI Toolkit (Hill, Gutwin, 2004), vagy a TeleEye (Pichiliani et al, 2009).

2.5.2. Igazságos rendezés (fairness), kiéheztetés (starvation)

Kölcsönös kizárást használó rendszerekben egy védett erőforrást – esetünkben egy workflow komponenst – egyszerre csak egy felhasználó kaphat meg. Eközben az erőforrásra több, más felhasználók várakozhatnak. A várakoztatás, majd a várakozók hozzáférése a felszabadult erőforráshoz valamilyen rendezési elv szerint történik. Egy rendszerben a rendezést igazságosnak mondjuk (angolul *fair*), ha az a személy kap meg hamarabb egy védett erőforrást, aki azt hamarabb kérte. Ha egy rendszer nem igazságos (vagy igazságtalan angolul *unfair*),

akkor nem beérkezés szerint, hanem valamilyen más sorrendben kerülnek a felhasználók kiszolgálásra.

Akár igazságos, akár igazságtalan rendszerről beszélünk, extrém esetekben előfordulhat, hogy valaki végtelenül hosszú várakozás után sem kapja meg a kért erőforrást. Ez a felhasználók „kiéheztetését” (starvation) eredményezi. Igazságos rendszer létrehozása várakozó sorok használatát igényli (Ellis, Gibbs, 1989). Egy várakozósor implementációja több féle lehet:

- *Nincs várakozó sor a rendszerben*: Ez a legegyszerűbben létrehozható változat. Ha egy felhasználó lefoglalási kérése nem elégíthető ki azonnal – mert a workflow komponens már foglalt – akkor a kérés nem kerül semmilyen várakozó sorba. A tényleges lefoglaláshoz újból igényelnie kell a komponenseket a felhasználónak egy olyan időpontban, amikor a foglalás már nem okoz zár ütközést. Az ilyen rendszer nem igazságos, ráadásul kiéheztetést is eredményezhet. Ehhez viszont az szükséges, hogy a felhasználó mindig éppen olyan időpontban kérje a foglalást, amikor a kért elem foglalt.
- *Van várakozó sor a rendszerben*: Ha egy felhasználó foglalási kérése zár ütközés miatt elutasításra kerül, akkor a kérése egy várakozó sor végére kerül. A sorban előtte álló kérések hamarabb kértek hozzáférést a workflow valamely komponenseihez. A várakozó sor implementációtól függően biztosíthat akár igazságtalan, akár igazságos rendezést:
    - *Igazságos várakozósor*: Egy szerkesztési tranzakció befejezésekor az általa birtokolt gráf komponensek felszabadulnak. Ekkor vesszük a várakozó sor első kérését (amelyik a legrégebben kért foglalást), megnézzük hogy az általa kért komponensek szabadok-e, és ha igen, akkor lefoglaljuk azokat. A felhasználó ekkor kiértesíthető arról, hogy a kért komponensek rendelkezésre állnak. Ennek a megközelítésnek az előnye, hogy kevés nyilvántartást igényel, egyszerűen eldönthető hogy ki jön legközelebb (vagy a sorban legelöl álló, vagy senki), és minden felhasználó egyszer sorra kerül. Ez utóbbi viszont csak akkor igaz, ha minden szerkesztési tranzakció véges időn belül befejeződik. Ez kollaboratív rendszerekben könnyen biztosítható a túl hosszúra nyúlt szerkesztést végző felhasználók figyelmeztetésével, esetleg a tranzakciók kényszerített lezárásával. Hátrány viszont, hogy egy nagy részgráfot igénylő felhasználó mögötti kérések feltorlódhatnak, arra várva, hogy a nagy gráfrész szabaddá váljon, a futó tranzakciók nagy része lezáródjon. Ilyen esetben a csoportmunka teljesítménye erősen lecsökken mind a gráfot egyidőben szerkeszteni képes felhasználók számát, mind a tranzakció befejeződések számát (throughtput) tekintve.
    - *Igazságtalan várakozó sor*: A sorban várakozó felhasználók bizonyos esetekben megelőzhetik egymást. Ha egy tranzakció befejeződése után a sorban legelőrébb álló olyan tranzakciót engedjük futni, amelyik nem okoz ütközést a gráfon lévő még megmaradt zárokkal, akkor biztosítható, hogy nem torlódnak fel a nagy részgráfot igénylő felhasználók mögött kérések. Ez egységnyi időre vetítve több tranzakció indulását, így nagyobb párhuzamos együttműködést eredményezhet. Viszont a nagy részgráfot igénylő felhasználókon nem segít, amennyiben folyamatosan érkeznek nála kisebb részgráfokat kérő lefoglalási kérések. Ilyenkor a nagy részgráfot igénylő felhasználóra kiéheztetés vár.

A rendszer kollaborációra biztosított teljesítményét és igazságosságát a 4. fejezetben tovább és részletesebben fogom analizálni. Előtte azonban a 3. fejezetben olyan továbbfejlesztést végzek rajta, amely az eddig tárgyalt módszerrel szemben a workflow alkalmazások konzisztenciáját magasabb, *szemantikai szinten* is képes megőrizni. A 4. fejezet vizsgálatai így majd egy intelligensebb rendszerre vonatkozhatnak.

3. Workflow alkalmazások szerkesztés közbeni konzisztenciája

Valós idejű kollaboratív szerkesztőrendszerek megalkotása közben felmerülő egyik legkomolyabb probléma a megosztott entitások konzisztenciájának biztosítása (Ignat, Norrie, 2008)(Ellis et al, 1991)(Chen, 2006). Csoportmunka rendszerekben a konzisztenciát két szinten lehet vizsgálni (Sun, 2002):

1. *Szintaktikai konzisztencia*: Annak biztosítása, hogy a replikált architektúrában a különböző gépek által kezelt másolatok azonosak, minden kollaboráló felhasználó ugyanazt a képet látja.
2. *Szemantikai konzisztencia*: Annak biztosítása, hogy a több felhasználó munkája által létrehozott adathalmaz a rá vonatkozó *integritási* feltételeknek megfelel, és emiatt azon további műveletek a szerkesztés után probléma nélkül végrehajthatók.

Ahogyan azt a 2.2.2 fejezetben részleteztem, a szintaktikai konzisztencia biztosítása három probléma elkerülésére ad megoldást: konvergencia probléma, oksági viszony megsértésének problémája, szándék sérülés problémája. Ezek elkerülésére a már ismertetett munkák időbeni eltolásával, zárolás alkalmazásával és szerializáció módszerekkel van lehetőség. *A szintaktikai konzisztencia követelményrendszere tehát csoportmunka rendszerekben általánosan definiálható, általános módon kezelhető, függetlenül a megosztott entitás jellegétől. A szemantikai konzisztencia ezzel szemben a megosztott entitás struktúrájához, illetve az azon értelmezett műveletekhez erősen kötődik, attól nem választható el. Más integritásszabályok vonatkoznak ugyanis egy szöveges dokumentumra, egy XML dokumentumra, egy CAD alkalmazásra, vagy egy workflow-ra. Sőt, ahogyan azt több publikáció is kimondja, a szintaktikai konzisztencia biztosítása általában nem elegendő a szemantikai konzisztencia biztosításához* (Skaf-Molli et al, 2003)(Sun, 2002).

A szemantikai konzisztencia garantálja, hogy az egyidejű változtatások logikailag, és nem csak szintaktikailag adnak helyes eredményt. A logikai hibák minél korábbi észlelése és javítása egyfelhasználós környezetekben is fontos. Többfelhasználós fejlesztőkörnyezetekben azért kap még nagyobb jelentőséget, mert több ember egymásra épülő munkája esetén, a csoport tudásbeli, időbeli, térbeli megosztottsága miatt még költségesebb a későn felismert hibák javítása. Az időben nem észlelt hibák a megosztott alkalmazásban tovagyűrűznek, és rendkívül megnehezítik a korrekciót (Dewan, Riedl, 1993).

Workflow-k esetén a szemantikai konzisztencia megőrzése egyfelhasználós környezetekben is létezik, a jelenlegi egyfelhasználós workflow szerkesztő rendszerek többsége nyújt konzisztencia megőrzési funkciókat. Workflow alkalmazásokban a szemantikai konzisztencia két szinten jelentkezik:

1. Komponens szintű konzisztencia
2. Gráf szintű konzisztencia

3.1. Komponens szintű szemantikai konzisztencia

Komponens szintű konzisztencia a gráfokat alkotó csomópontok és élek önmagukban tekintett helyességét jelenti. Ezen a szinten a konzisztencia szabályok rendszerfüggőek, annak

függvényében változnak, hogy pontosan mit jelképez a workflow-ban egy-egy csomópont, egy-egy él.

- Egy Web szolgáltatásokat, mint csomópontokat tartalmazó workflow-ban csomópontra vonatkozó komponens szintű konzisztencia feltételek lehetnek, hogy a Web szolgáltatás helyét azonosító paraméter URL formátumú legyen, létező WSDL fájlra mutasson, ne legyen NULL (üres) érték.
- Egy számítási feladat (job) típusú csomópontokat kezelő workflow rendszer esetén csomópontra vonatkozó konzisztencia feltételek lehetnek, hogy a bemenő adatokat megadó paraméterek (csatornák) fájl elérési út típusú hivatkozásokat tartalmazzanak, létező fájlokra mutassanak, ne legyenek NULL (üres) értékek.
- Például élre vonatkozó komponens szintű konzisztencia feltételek lehetnek, hogy az élben megadott adatátviteli út NULL értékű lehet, vagy DNS formátumban szereplő szerverek listáját tartalmazza, és utóbbi esetben a szerverek létezzenek.

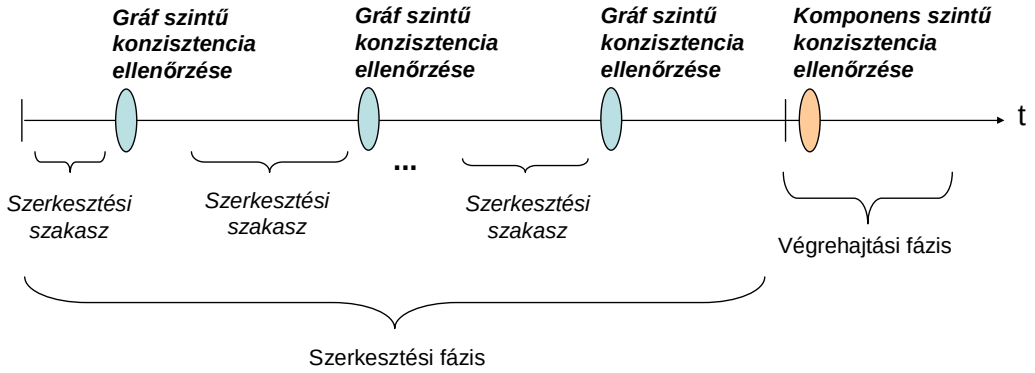
Egyfelhasználós workflow szerkesztő környezetekben a komponens szintű konzisztencia feltételek betartatása a szerkesztőrendszer feladata, és tipikusan utólagosan történik. Ezek a feltételek ugyanis nem kell, hogy minden pillanatban fennálljanak a gráfra, elég őket csak a szerkesztési fázis végén ellenőrizni. Konzisztenciatörés esetén a törést okozó komponensek további szerkesztésre van szükség, addicionális szerkesztési szakaszok indításával. Szerkesztési fázis közben egyébként is lehetetlen lenne minden komponens szintű konzisztencia feltétel betartatása, mivel például a fenti első pontban ez egy Web szolgáltatás workflow esetén azt jelentené, hogy a felhasználónak egyetlen szerkesztési tranzakcióval kell megadnia minden Web szolgáltatás elérési útját. Ez nyilvánvalóan lehetetlen, ezért a rendszer hagyja a kitöltetlen, vagy nem helyesen kitöltött komponensek elmentését is (Ld. 3-1. ábra). Ilyen módon működik a legtöbb workflow fejlesztőrendszer, például a P-GRADE Portál is. A P-GRADE Portálban lehetőség van szerkesztési tranzakciók lezárására úgy, hogy a gráf komponenseire nem teljesülnek a komponens szintű konzisztencia feltételek – maradhatnak például olyan csomópontok a gráfban, amelyekhez nincs „futtatandó program” (executable) megadva.

A komponens szintű konzisztencia feltételek betartatása az előző fejezetben ismertetett kollaboratív környezetben nem jelent problémát, mivel egy-egy komponens egyetlen felhasználó tulajdonában van, komponenseken belüli bontás nem fordulhat elő. Emiatt lehetetlen, hogy egy komponens összetett (több paraméterét) érintő konzisztencia feltétel azért nem elégíthető ki, mert több különböző felhasználó között vannak ezek a paraméterek szétosztva. *A komponens szintű konzisztencia feltételek betartatása tehát komponens szintű zárolást használó kollaboratív workflow rendszerben is ugyanúgy kerülhet kivitelezésre, ahogyan egyfelhasználós esetben történik.*

3.2. Gráf szintű szemantikai konzisztencia

A gráf szintű konzisztencia feltételek komponensek együttesére, több komponens közötti kapcsolatokra definiálnak szabályokat. A gráf szintű konzisztencia feltételek ellenőrzése és betartatása amiatt bonyolultabb a komponens szintű feltételekénél, mert csak a gráf egy részének, esetleg az egész gráfnak az ismeretében végezhető el. A komponens szintű konzisztencia kritériumokhoz hasonlóan a gráf szintű konzisztencia kritériumok listája is attól függ, hogy az adott rendszer milyen típusú workflow-kat támogat. Míg azonban a komponens szintű kritériumok a gráfot alkotó elemek típusától függnek, addig a gráf szintű kritériumok számára a komponensek típusa mellékes. *A gráf típusú konzisztencia feltételek számára a gráf, mint logikai struktúra felépítése a fontos.*

Egyfelhasználós workflow környezetek a gráf szintű konzisztenciát minden szerkesztési tranzakció végén ellenőrzik, biztosítva ezzel egyrészt azt, hogy strukturálisan helyes workflow szolgál majd a következő tranzakció számára kiindulási pontként, másrészt, hogy – abban az esetben, ha ez volt a szerkesztési fázis utolsó szakasza – a gráf komponens szintű ellenőrzés után azonnal futtatható. A futtathatóság biztosítja, hogy a workflow-t a feldolgozó motor képes lesz bejárni.



3-1. ábra: Komponens és gráf szintű konzisztencia feltételek ellenőrzésének időpontjai egy workflow életében. A gráf szintű konzisztencia legkésőbb minden szerkesztési szakasz végén, a komponens szintű konzisztencia viszont csak a futtatási fázis elején kerül betartatásra.

Egyfelhasználós környezetekben a gráf szintű konzisztencia feltételek ellenőrzése történhet minden workflow-n elvégzett művelet után, vagy a tranzakció lezárásakor. Előbbi esetben a workflow nem tud „eltávolodni” a konzisztens gráf állapottól, hiszen minden változtatás konzisztens gráfból konzisztens gráfot kell hogy adjon. Utóbbi esetben viszont előfordulhat, hogy a felhasználónak több, már elvégzett módosítását kell kompenzálnia miután kiderül, hogy a workflow jelen állapotában nem menthető el. Akár minden módosítás után, akár csak a mentéskor ellenőrződnek le a gráf szintű konzisztencia kritériumok egy workflow-ra, tény, hogy *egyfelhasználós környezetben a szerkesztőrendszer ismeri a teljes gráfot, bármikor képes annak struktúrájára vonatkozó feltételeket ellenőrizni, konzisztenciatörés esetén beavatkozni. Kollaboratív szerkesztés esetén ez a feltétel nem áll fenn.*

A dolgozat ezen fejezetében a következő gráf szintű konzisztencia kritériumok kollaboratív rendszerben való betartatására adok megoldásokat:

1. A gráfban nem lehet szabad él, vagyis olyan él, amelynek vagy egyik, vagy másik, vagy mindkét végpontja nem létező csomópontra mutat. (Az él valamelyik vége „lekötetlen”, még szabad.)
2. A gráf egyetlen komponenséhez sem kapcsolódhat többszörös bejövő él, vagyis nem fordulhat elő, hogy egynél több él használja célként ugyanazon csomópont ugyanazon bemenő csatornáját.
3. A gráfban nem lehet kör.

A szabad élek, többszörös bejövő élek, körutak a gráf csomópontok és élek jellegétől függetlenül bármilyen rendszerben problémát jelenthetnek. Ezen három konzisztencia feltétel fontos kritérium számos workflow rendszerben, elsősorban a tudományos workflow-k témaköréből, ahol túlnyomó többségben van az aciklikus gráfok alkalmazása. Ilyen konzisztencia kritériumokkal rendelkező workflow eszközök például a következők: DAGMan,

P-GRADE Portal, Taverna, Triana. Vannak olyan rendszerek is, ahol a három kritérium közül nem mindegyik áll fenn. Például a Kepler környezetben megengedett a ciklikus, vagyis kört alkalmazó workflow-k létrehozása. Sőt, egy Kepler környezettel készített workflow valamely csomópontja többszörös bejövő élekkel is rendelkezhet. Azonban az ilyen csomópont speciális csomóponttípus megjelölést igényel (Multi-channel csomópontot, ami egy csomópont-paraméter beállítását jelenti), a normál csomópontokra továbbra is él és betartandó feltétel a bejövő élek kizárólagossága.

A három féle, gráf szintű konzisztencia feltétel egyike sem ellenőrizhető le egyetlen gráfkomponens ismeretében, az ellenőrzéshez több komponensre, egy részgráfra van szükség. Az előző fejezetben definiált kollaboratív szerkesztés folyamán viszont minden kliens oldali fejlesztőkörnyezet a gráfnak csak egy részét ismeri: azt a részt, amelyet az adott felhasználó munkája közvetlenül érint, ideális esetben egyetlen komponenssel sem többet. A gráf többi részéről, a le nem foglalt vagy a mások által lefoglalt komponensekről a szerkesztő nem feltétlenül rendelkezik naprakész verzióval. *A gráf szintű konzisztencia kritériumok ellenőrzése ezért kollaboratív esetben nem történhet az egyfelhasználós rendszerekben sikerrel alkalmazott módokon.*

3.3. Konzisztencia biztosítása csoportmunka környezetekben, workflow környezetekben – szakirodalmi áttekintés

Mivel a szintaktikai konzisztencia feltételrendszere általánosan, a megosztott entitások struktúrájától függetlenül megadható, ezért a CSCW szakirodalom erőteljesen a szintaktikai szemantika biztosítására összpontosít. Ezen a témakörön belül is a szöveges dokumentumok szintaktikai konzisztenciájának biztosítása a leginkább tárgyalt téma (Preston, 2007) (Sun et al, 1998) (Ignat, Norrie, 2008) (Prakash, Knister, 1992) (Ressel et al, 1996) (Suleiman et al, 1997) (Sun, Sun, 2006) (Sun, 2002) (Noel, Robert, 2003)(Oster et al, 2006)(Gu, Yang, Zhang, 2005).

Szöveges dokumentumok esetén a szintaktikai fölött szemantikai konzisztencia biztosítása annak garantálását jelenti, hogy a felhasználók közös munkája nemcsak hogy minden gépen azonos eredményt ad, de az így kapott mondatok *nyelvtanilag is helyesek lesznek*. Ez a probléma máig nem megoldott, és szótárprogramok csoportmunka környezetekkel való integrálását igényli (Sun, 2002)(Xue, Orgun, Zhang, 2002).

Yang és Li munkái (Yang, Li, 2004) (Yang, Li, 2005) szintén replikált architektúrájú CSCW környezetekben való szintaktikai konzisztencia biztosítására adnak megoldást. A megosztott entitás náluk fa struktúraként ábrázolható fájlrendszer és nem szöveges dokumentum. Az általuk létrehozott rendszerben a fastruktúra különböző objektumaira (könyvtárak, fájlok) más-más konkurenciakezelési mód választható, így egyetlen fájlrendszeren belül egyidőben használható a szabadabb módosítást lehetővé tevő művelet transzformációval megvalósított szerializáció, és a kötöttebb, de garantált konfliktuskezelést nyújtó zárolási módszer különböző változatai is.

Xue és Orgun cikkükben (Xue, Orgun, 2005) a dokumentumok különböző verzióinak elkészítésével garantálják a kollaboratív szövegszerkesztés végére a szemantikai konzisztenciát. Az architektúra itt is replikált, a konzisztencia biztosítás miatt senkinek a dokumentumon végzett változtatása nem kerül eldobásra, viszont egy-egy kollaboratív szakasz végén több dokumentumpéldány is létrejöhet, melyet valakinek kézzel kell egybefűznie. A több különböző dokumentumváltozat miatt azonban ez a módszer nem biztosít szintaktikai konzisztenciát, ebből is látszik, hogy a két konzisztenciaszint független egymástól.

A Xue és Orgun által használt „multi-version” módszer széles körben használt adatbázisokban való konzisztenciaőrzéshez (Bernstein, Goodman, 1987). Ezek a módszerek az adatelemek több változatának létrehozásával kerülik el azt, hogy az inkonzisztens adat

létrehozását eredményező tranzakciót abortálni kelljen, ami az addig elvégzett módosításainak visszaállításával – és felesleges pluszmunkával – jár. Csoportmunka szoftverek csak nagyon ritkán alkalmazzák a „multi-version” módszert (Pedersen et al, 1993)(Sun, Chen, 2002) mivel az elkészült objektum több verziójának utólagos, manuális összefűzését igényli. Bizonyos körökben a több verziót létrehozó szerkesztőrendszereket nem is tekintik csoportmunka rendszereknek, amiatt, mert a funkciójuk teljes egészében kiváltható verziókövető rendszerekkel (Xue, Orgun, 2005).

Bhola et al cikke (Bhola et al, 1998) objektum halmazok kollaboratív módosítására ad zárolás alapú megoldást. A halmaz objektum elvételével, hozzáadásával módosítható. A nehézséget kollaboratív módosítás során az adja, hogy egy-egy objektum tartalmazhat további objektumokat, akár olyanokat, amelyek már eleve a halmazban vannak. Ilyen esetben ezen objektumok verziójának konzisztenciáját kell biztosítani. Annak ellenére, hogy ezek az objektumhierarchiák gráfként ábrázolhatóak, más szemantikai konzisztencia szabályok vonatkoznak ezen gráfokra, és a workflow alkalmazások gráfjaira. Mivel itt az objektum-tartalmazás jelenti a gráf egy-egy élét, és ezen tartalmazás adatelemként nem jelenik meg a rendszerben (nincsenek paraméterei, a tartalmazás csak egy igen/nem bit), ezért nem alakulhatnak ki sem szabad élek, sem többszörös bejövő élek. Az objektumok „körben egymásba tartozásaként” létrejövő kör pedig már az objektum-orientált nyelv által, esetükben a Java nyelv által eleve tiltott, a hierarchia fagráfként adott.

Workflow alkalmazások kollaboratív szerkesztés közbeni helyességének biztosításához kapcsolódó eredmények a 2.3 fejezetben tárgyalt kapcsolódó munkák közül egyedül Held és Blochinger cikkeiben találhatóak (Held, Blochinger, 2008)(Held, Blochinger, 2009). A Hobbes rendszerben a kollaboratív csoport egy kitüntetett tagja (a workflow tulajdonosa) a kollaboratív szerkesztési szakasz végén létrejött gráfot át kell hogy nézze, az esetleges hibákat *kézzel* kell hogy javítsa. Amennyiben egyedül nem tudja elvégezni a javítást, akkor további kollaboratív szakaszokat kezdeményezhet. Viszont ennek a manuális javításnak elsősorban nem a konzisztencia biztosítás, hanem a workflow-optimalizáció a szerepe. Feltételezés ugyanis, hogy a kollaboratív csoport tagjai esetleg nem a leghatékonyabb futást eredményező gráfot hozták létre. A hatékonysági problémák felismerése a gráf futtatása nélkül nehéz, a Hobbes ezért különböző, a BPEL nyelv sajátosságait figyelembe vevő metrikák generálásával segíti a felhasználót az optimalizálásra szoruló workflow részek felismerésében.

A 2.4 fejezetben ismertetett zárolási kérés elbíráló algoritmus képes a workflow-k szintaktikai konzisztenciájának betartására. Nem képes azonban a 3.2 fejezetben tárgyalt három gráf szintű konzisztencia kritériumot biztosítani. A 3. fejezet hátralevő részeiben ismertetem, hogy a 2.4 fejezet „naiv” zárolási kérés elbíráló algoritmusával miért nem garantálható a workflow-k szemantikai konzisztenciája, illetve megadok olyan gráf felosztó algoritmusokat melyekkel ezek a konzisztencia feltételek biztosíthatók, méghozzá úgy, hogy nem fordulhat elő a workflow-n elvégzett változtatások eldobása, az emiatti adatvesztés. *Az új módszerek lényege, hogy egy intelligens zárolási kérés elbíráló algoritmus csak olyan részgráfokat enged meg lefoglalni, melyeken belül amennyiben biztosítottak a gráf szintű konzisztencia feltételek, akkor azok a gráf egészére is biztosítva lesznek. Mivel egy-egy részgráfon belül a szemantikai konzisztenciát a kliens oldali szerkesztőkörnyezetek képesek garantálni, ezért az ilyen intelligens gráf felosztó algoritmusok a teljes workflow-k szintjén biztosítják a szemantikai helyességet még többfelhasználós, kollaboratív szerkesztési esetekben is.* A kapcsolódó szakirodalmakban nem találtam hasonló megoldásokat, így ez a gráf felosztáson alapuló szemantikai konzisztenciabiztosítási mód teljesen újszerű.

3.4. Workflow alkalmazások gráf szintű konzisztencia feltételeinek biztosítása

3.4.1. Szabad élek kialakulása elleni védelem

Szabad él akkor alakul ki egy workflow gráfban, ha egy ki- vagy bemenő élekkel rendelkező csomópont törlésre kerül a gráfból úgy, hogy a hozzá kapcsolt élek megmaradnak[7]. Az eredmény *szabad él, vagyis olyan él, amelynek vagy egy, vagy mindkét vége a "levegőben lóg", nem kapcsolódik csomóponthoz.* Szabad élek ellen védekezni minden workflow rendszerben azért fontos, mert az ilyen élek a workflow végrehajtás közben a futtatás megakadását (ha az él forráscsomópontja hiányzik), vagy adatvesztést okozhatnak (ha az él célcsomópontja hiányzik.). Egyfelhasználós környezetekben szabad élek elleni védelem a következő módon biztosítható:

1. Egy csomópontot a rendszer nem enged törölni mindaddig, amíg van hozzá kapcsolva él. Így egy csomópont törlése előtt a felhasználónak törölnie kell minden, a csomóponthoz kapcsolódó élt, ezután a csomópont törlése biztosan nem hagy hátra szabad élt. *vagy*
2. Ha a felhasználó olyan csomópontot töröl, amelyhez él is kapcsolódik, akkor a rendszer automatikusan törli ezeket az éleket. Ez tulajdonképpen egy egyszerű művelet transzformációs megoldás, amely a „csomópont törlés" műveletet egy „csomópont törlés és *N* db él törlés" művelettel helyettesíti.

Ha ezen módszerek valamelyikét változtatás nélkül adaptálnánk egy kollaboratív workflow fejlesztőrendszerre, akkor előfordulhatna, hogy két különböző felhasználó birtokolja a törlendő csomópontot illetve egy hozzá kapcsolódó élt. Ha ekkor a fenti 1. számú megoldást alkalmaznánk, akkor a csomópont nem törölhető, vagyis az azt birtokló felhasználó nem módosíthatja tetszőlegesen az általa birtokolt gráfrészt. Ha a fent ismertetett 2. megoldást választanánk, akkor viszont a csomópontot birtokló felhasználó a csomópont törlésével befolyásolja a másik felhasználók munkáját, hiszen hozzáadná (az művelet transzformáción keresztül) az "él törlés" műveletet az élt birtokló felhasználó tranzakciójához – ezzel esetlegesen megsemmisítve az ő értékes, élen addig elvégzett módosításait. Ezek a megoldások tehát önmagukban elég zavaró jelenségeket okoznának a fejlesztőknek, és emiatt nem megfelelőek kollaboratív workflow rendszer számára.

Egy többfelhasználós kollaboratív környezetben kulcsfontosságú, hogy egy csomópont és az ahhoz kapcsolódó élek ne kerüljenek különböző felhasználók fennhatósága alá. Ennek betartását, és így egy többfelhasználós rendszerben szabad élek elleni védelmet nyújtani képes a következő protokoll (továbbiakban P1-nek hívott), amely alapjául szolgálhat a szabad élek elleni védelmet nyújtó, zárolási kéréseket elbíráló algoritmusoknak:

Protokoll 1 (P1): Szabad élek elleni védelem

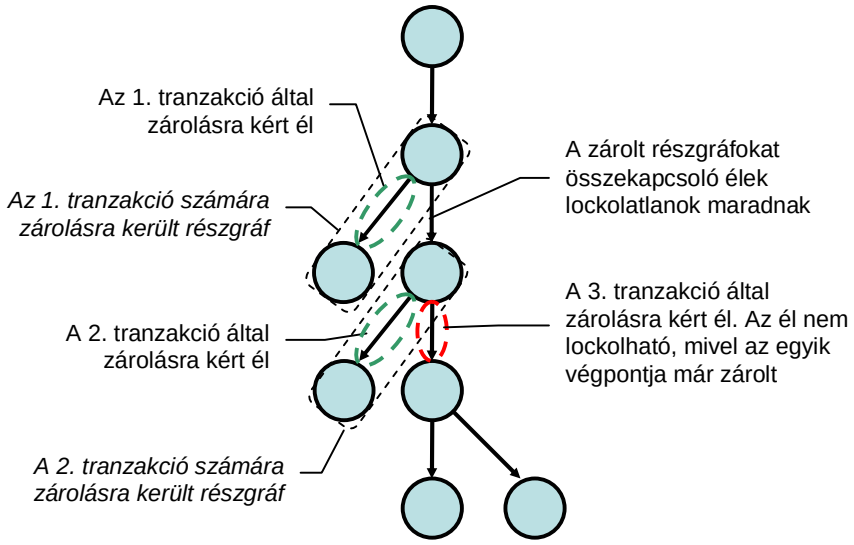
1. Egy él csak akkor kerülhet egy tranzakció számára lefoglalásra, ha annak mindkét végpontja (csomópontja) szintén lefoglalható ezen tranzakciónak. Ez a feltétel biztosítja, hogy *egy felhasználó által birtokolt részgráf biztosan csomópontokkal lesz körülzárva*. Azok az élek, amelyek különböző felhasználók által birtokolt részgráfokat kapcsolnak össze emiatt nem kerülhetnek senki számára lefoglalásra. A

[7] Feltételezem, hogy egy gráf szerkesztő környezet csak már létező csomópontok között enged új élt létrehozni, ezért „él létrehozás" művelet nem eredményezhet szabad élt.

3-2. ábrán látható egy olyan kollaboratív workflow szerkesztési eset, amely ennek a protokollnak megfelelően osztja fel a gráfot a felhasználók között.

2. Egy lefoglalt részgráfon belül a kliens oldali szerkesztőrendszer védekezhet szabad él kialakulása ellen ugyanúgy, ahogyan egyfelhasználós rendszerek teszik (Azaz az előző felsorolás 1. vagy 2. módszerét használva.)
3. Ha egy szerkesztési tranzakció befejeződik, akkor az általa végzett módosítások mind belekerülnek a teljes gráfba. Ezen a ponton derül ki, hogy a tranzakció törölt-e olyan csúcsot, amely más által birtokolt, vagy még nem lefoglalt workflow részekhez éllel kapcsolódik. Ha van ilyen kapcsolat, akkor a rendszernek az integrálás során törölnie kell ezeket az éleket, különben ezek szabad élekké válnának. Mivel P1 1. pontja biztosítja, hogy az ilyen élek egyetlen felhasználó által lefoglalt részgráfjának sem részei, ezért ez az automatikus „él törlés” művelet nem jelent külső befolyást vagy kidobott munkát egyetlen felhasználóra nézve sem.

P1 biztosítja, hogy a felhasználók a részgráfjaikon belül továbbra is bármit módosíthassanak és biztosak lehessenek benne, hogy ennek a részgráfnak a szerkesztésébe sem más, sem a rendszer közvetlenül nem avatkozhat bele. *A szerkesztésük eredményeként kapott részgráfok összeillesztésével szabad élektől mentes gráf jön létre, amely tartalmazza majd minden felhasználó gráfon végzett módosítását.*



3-2. ábra: Szabad élek elleni védelmet biztosító protokoll használat közben.

A protokoll tehát képes szabad élek kialakulása ellen védekezni, önmagában azonban nem tekinthető még zárolási kérés elbíráló algoritmusnak (a 2-9. ábrán látható zárolást végző szerveren belül működő, a zárolás megadásáról és megtagadásáról döntő algoritmusnak). *A protokoll azonban beintegrálható tetszőleges zárolási kéréseket elbíráló algoritmusba és ilyen esetben az adott zárolási kéréseket elbíráló algoritmus szabad élek elleni védelmet nyújt.* (Erre konkrét példát először a 3.5 részben fogok mutatni.)

Állítás: Egy P1-et implementáló zárolási kéréseket elbíráló algoritmust használó rendszerben semmilyen szerkesztési tranzakció nem tud szabad élt létrehozni.

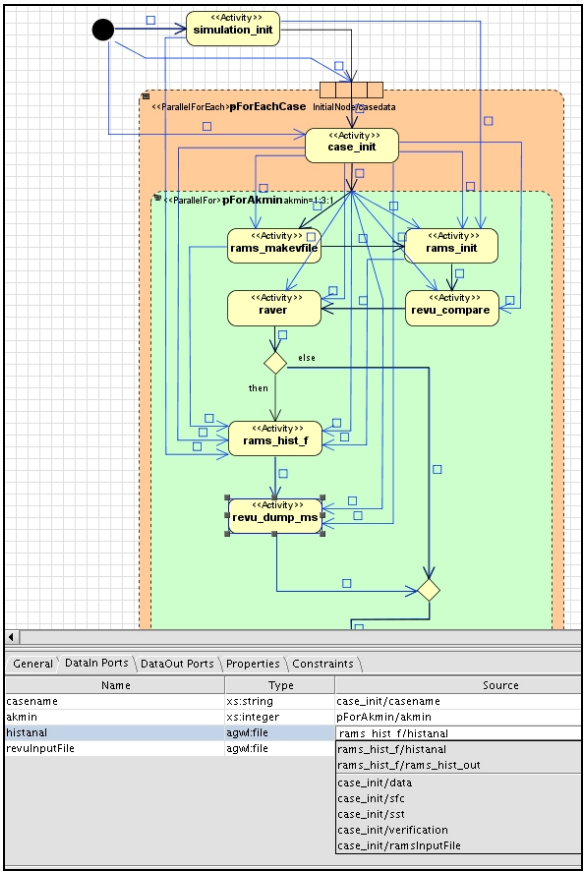
Bizonyítás:
Tegyük fel, hogy egy G gráfon a G_i komponenshalmazt szerkeszteni engedte a T_i tranzakció számára egy, az 1. protokollt implementáló zárolási kéréseket elbíráló algoritmus. Tegyük fel, hogy a T_i tranzakció töröl egy N csomópontot G_i-ből és hogy ehhez a csomóponthoz a G gráfban legalább 1 db él (e él) kapcsolódik. Ekkor e-re nézve az alábbi esetek valamelyike áll fenn:

1. $e \in G_i$ (e a T_i tranzakció számára van lefoglalva)
   Ebben az esetben a T_i-t futtató szerkesztőrendszer eltávolítja az e élt, nem marad hátra szabad él.
2. $e \notin G_k \mid k = 1,...,n$ (e egyetlen lefoglalt részgráfnak sem része)
   Ekkor T_i lezárásakor a szerkesztőrendszer el fogja távolítani a gráfból az e élt, szabad élként az nem marad hátra.

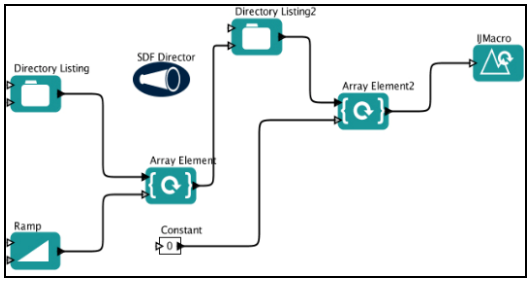
Egy harmadik, $e \in G_j \mid j \neq i$ (e egy valamilyen $T_j \neq T_i$ tranzakció számára van lefoglalva) eset nem állhat fenn, ugyanis a protokoll nem engedné T_i és T_j egyidőben való futását, ekkor ugyanis $N \in G_j$-nek fenn kéne állnia, ami lehetetlen, hiszen egyetlen komponens sem lehet egyidőben több tranzakció számára lefoglalva.
□

3.4.2. Többszörös bejövő élek elleni védelem

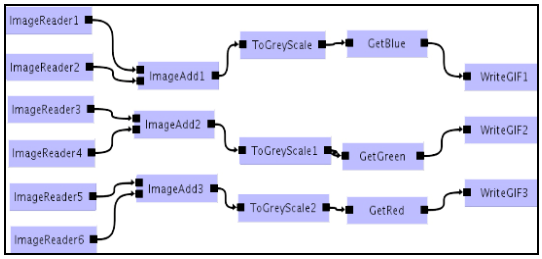
Ugyan a korábbiak során egyszerűen úgy kezeltem az éleket, hogy azok csomópontok között futnak valójában egy workflow gráfban egy él nem egy csomópontra, hanem csomóponton belüli paraméterre hivatkozik forrásként, és célként. Egy ilyen paraméter nem lehet akármilyen: él forrásaként csak „komponens által előállított kimenő adat” paraméter szolgálhat, míg célként csak „komponens által elvárt bemenő adat” paraméter szerepelhet. Egy-egy ilyen csomópont ki- és bemenő adatát reprezentáló paramétert a különböző workflow környezetek hol „portként”, hol „csatornaként” emlegetik, de valójában minden esetben a csomópont egy paraméteréről van szó, amely adatvégpontot reprezentál. Az, hogy a gyakorlatban egy csomópont által termelt adat vagy vezérlési utasítás, illetve az általa elvárt adat vagy vezérlési utasítás valójában a csomóponthoz szervesen hozzátartozó paraméter, abból következik, hogy a csomópont jelentésének a megváltozása (pl. Másik Web szolgáltatás megadása, vagy egy másik job megadása) *magával hozza a ki- és bemenő csatornák számának és jelentésének a megváltozását* is. A csomópontból adódik tehát hogy hány darab és milyen típusú, milyen jelentéssel bíró ki- és bemenő csatorna van. A csatornák különböző workflow rendszerekben való megjelenítésére mutat példákat a 3-3. ábra. *Mivel élekhez és csomópontokhoz tartozó paramétereket a 2.1 fejezetben bevezetett modell tartalmaz, ezéert azon nem kell változtatni.*



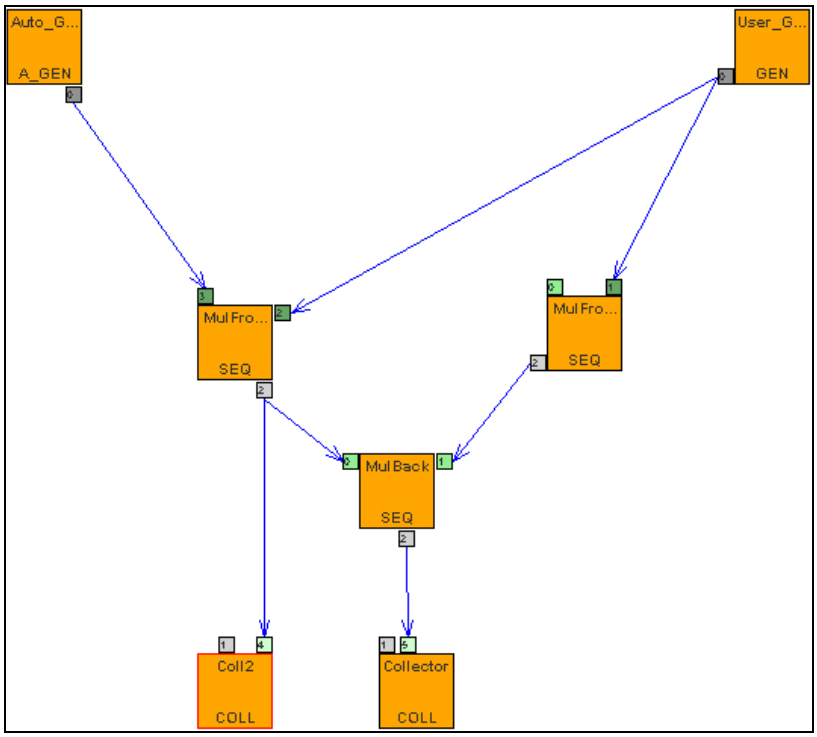
ASKALON



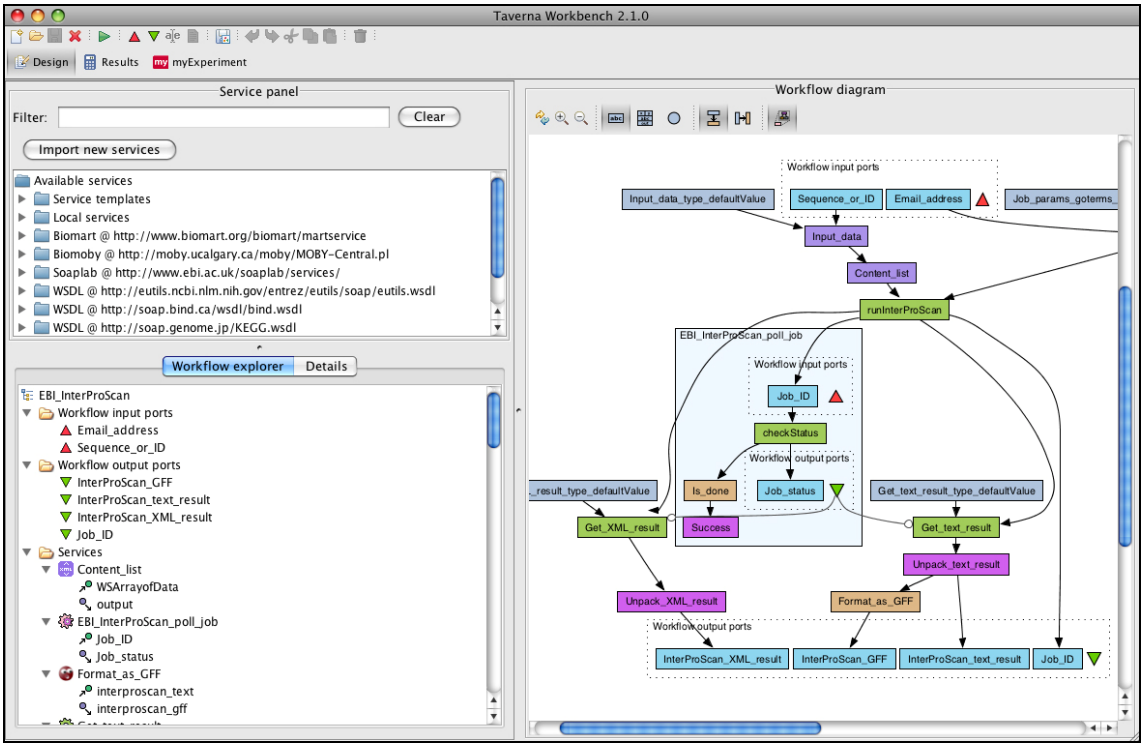
Kepler



Triana



P-GRADE Portal



Taverna

3-3. ábra: Csomópontok ki- és bemenő adatcsatornáinak megjelenítése a legelterjedtebb workflow környezetekben. A csatornák száma és típusa minden esetben a csomóponttól függ, tehát annak paramétereként fogható fel. A P-GRADE Portal, Kepler és Triana környezetek grafikusan, a csomópontokhoz kötve is ábrázolják az adatcsatornákat (kis dobozokként), míg az ASKALON és Taverna eszközökben egy csomópont kijelölése után külön ablakban lehet látni annak adatcsatornáit.

A 2. gráf szintű kritérium azt írja elő, hogy *egy csomópont semelyik bemenő csatorna paraméterére sem hivatkozhat egynél több él. A kritérium biztosítja, hogy a különböző éleken keresztül az adott csatornán beérkező adatok nem írják felül egymást, vagyis a csomópont futásakor nem lesznek elveszett adatok, a csomópont futása kiszámítható és megismételhető.*

Kimenő élekre esetén nincs ilyen megkötés, mivel itt egy csomópont által megtermelt adatelem akár több további csomópont számára is szolgálhat bemenő adatként. Ilyen esetben több kimenő él kapcsolódik ugyanahhoz a „kimenő csatorna” csomópont paraméterhez. (Ld. például 3-3. ábrán a P-GRADE Portal workflow-ban a „Mul Fro...” elnevezésű csomópont „2” számú kimenő csatornáját, amely két további bemenő csatorna számára is szolgáltat adatot.)

Egyfelhasználós környezetben a szerkesztőrendszer tud a gráfban zajló összes él létrehozási műveletről, és képes megakadályozni ugyanahhoz a bemenő csatornához második él hozzáadását. Kollaboratív környezetben ez az út nem járható. *Kollaboratív környezetben ennek a konzisztencia feltételnek a betartása úgy biztosítható, ha lehetetlenné tesszük, hogy egyazon bemenő csatornához egyidőben több felhasználó is élt köthessen.* Mivel komponens szintű zárolást használunk, ezért ez azt jelenti, hogy egy csomóponthoz csak az köthet bemenő élt, aki a komponenst zárolta. Ebben az esetben a többszörös bejövő élek ellen a csomópontot birtokló felhasználó szerkesztőrendszere tud védekezni, biztos lehet benne, hogy a csomóponthoz más nem köt bejövő élt.

Állítás: A 3.4.1 fejezetben ismertetett P1-et implementáló zárolási kéréseket elbíráló algoritmust használó kollaboratív rendszerben nem lehet többszörös bejövő élt létrehozni.

Bizonyítás:
Tegyük fel, hogy egy *G* gráfban a T_i tranzakció létrehoz az N_1, N_2 csúcsok között egy *e* élt, és ez az él N_2-ben többszörös bejövő élt eredményez. Ekkor N_2-re nézve az alábbi esetek valamelyike áll fenn:

1. $N_2 \in G_i$ (N_2 a T_i tranzakció számára van lefoglalva)

   Ebben az esetben N_2-höz más tranzakció nem adhatott élt, vagyis az az él, amellyel együtt e többszörös bejövő élként jelenik meg T_i indulásakor már létezett, és emiatt a T_i-t futtató szerkesztőrendszer számára ismert. Emiatt a szerkesztőrendszer felismeri hogy az és eltávolítja az *e* élt, tehát nem maradhat meg többszörös bejövő élként.
2. $N_2 \notin G_i$ (N_2 a T_i tranzakció számára nincs lefoglalva)

   Ebben az esetben a protokoll szerint az *e* él nem jöhet létre, egy élt ugyanis csak akkor birtokolhat egy felhasználó, ha annak mindkét végpontját birtokolja. Az *e* él tehát ez esetben sem maradhat meg többszörös bejövő élként.

□

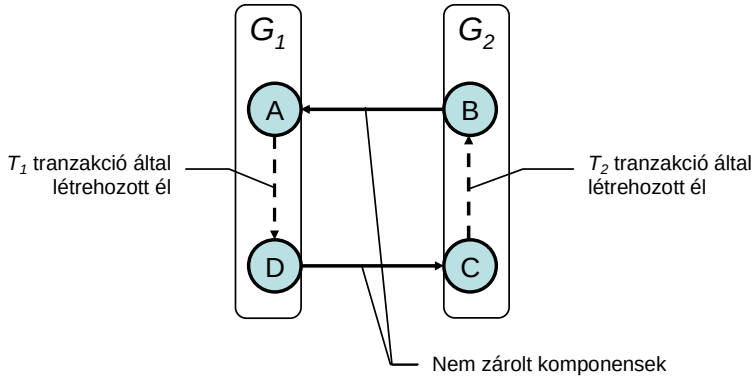
A fenti két bizonyítás értelmében tehát *egy P1-et implementáló zárolási kéréseket elbíráló szerver mind a szabad, mind a többszörös bejövő élek ellen védelmet nyújt*.

3.4.3. Kör kialakulása elleni védelem

Kör létezése egy workflow-ban azt jelenti, hogy a gráfban van olyan csúcs, melyből kiindulva az irányított éleket követve oda visszajuthatunk. Számos workflow környezetben az aciklikusság fontos gráf szintű konzisztencia feltétel. Ugyan egyre több workflow rendszer képes kört tartalmazó gráfokat is futtatni, számos alkalmazási területen (például a Grid workflow alkalmazásokban) még mindig túlsúlyban vannak a ciklust kezelni nem képes megoldások. Az aciklikus gráfok támogatása azért is különösen fontos területe a workflow kutatásoknak, mivel még *a ciklust tartalmazó workflow-k egy része is aciklikus workflow-kra vezethető vissza*. Ha ugyanis egy ciklikus workflow-ra annak futtatása előtt ismert, hogy cikluson hány alkalommal kell végrehajtani, akkor a ciklusban szereplő komponensek egymás utáni gráfba másolásával aciklikus workflow hozható létre.

A ma legelterjedtebb, workflow-k aciklikusságát megkövetelő grides workflow rendszerek a következők (GWF 2010): BioOpera, DAGMan, GridAnt, Chimera, Concrete Workflow Generator, Pegasus, P-GRADE Portal, Taverna (scufl nyelvet használ), Triana, XWFL (XML-based WorkFlow Language).

Egyfelhasználós esetben – a korábbiakhoz hasonlóan – a szerkesztőrendszer tud a gráfban zajló összes él létrehozási műveletről, látja a gráf teljes felépítését és képes megakadályozni a gráfban kör kialakulását. Kollaboratív fejlesztés esetén ez az út nem járható, mivel a gráf aktuális állapota nem ismert egyetlen szerkesztőkörnyezet számára sem. Emiatt az előző fejezetben ismertetett „naiv” zárolási kéréseket elbíráló algoritmus gráf felosztása esetén előfordulhat, hogy több, egyidőben dolgozó felhasználó munkája a gráfban kört eredményez úgy, hogy külön-külön egyik részgráfban sincs kör. A 3-4. ábra egy egyszerű példát mutat ilyen esetre: sem G_1, sem G_2 részgráfokban nincs kör, mindkét tranzakció változtatásait elfogadva mégis kör jön létre a teljes gráfban.



3-4. ábra: Több, egyidőben futó olyan tranzakció, amely külön-külön aciklikus gráfot eredményez, együtt adhat ciklikus gráfot. G_1 a T_1 tranzakció számára lefoglalt részgráf, G_2 a T_2 tranzakció számára lefoglalt részgráf.

Egy központi „ciklus ellenőrző szolgáltatás” beillesztése a kollaboratív architektúrába egyrészt az offline (hálózatról lecsatlakozott módon) gráfot szerkesztő felhasználók miatt sem lehetséges, másrészt a hálózati késleltetés miatt nem is praktikus. Az aciklikusság utólagos, tranzakció lezáráskor való ellenőrzése viszont abortált tranzakciókat eredményezne, ekkor ugyanis a felhasználók egymás munkáját befolyásolhatnák, saját szerkesztésükkel konzisztenciasértővé tennék a később lezáródó tranzakciók változtatásait.

Sajnos a szabad élek, illetve a többszörös bejövő élek elleni védelemhez hasonló módon nem hozható létre egyetlen protokoll a kör elleni védelemre. Emiatt a következő fejezetekben több különböző – és ahogy az 4. fejezetben bemutatom – nem ugyanolyan hatékonyságú megoldást adok. *A megoldások közös jellemzője, hogy a megosztott gráfban a zárolási kéréseket elbíráló menedzser úgy próbálja a lefoglalt részgráfokat kialakítani, hogy az azokon belül garantált körmentesség a teljes gráfban is garantálja a körmentességet. Mivel a menedzser által létrehozott részgráfokon belül a kliens oldali szerkesztőrendszerek biztosítani tudják a körmentességet, ezért a körmentes részgráfok összefűzésével kapott teljes gráf is körmentes lesz.*

3.5. Konzisztenciaőrző, zárolási kéréseket elbíráló algoritmus A1

Ebben a fejezetben megadom az 2. fejezetben definiált zárolási kéréseket elbíráló algoritmus olyan kiterjesztését, amely képes biztosítani a workflow-kon a fent részletezett 3 féle gráf szintű konzisztencia feltételt a tranzakciók abortálása vagy utólagos kompenzációja nélkül. *Az új algoritmus implementálja az A 3.4.1 fejezetben ismertetett P1-et – emiatt tehát szabad élek és többszörös bejövő élek kialakulása elleni védelmet nyújt – továbbá csak olyan konfigurációkban enged részgráfokat lefoglalni, melyeken belüli körmentesség a teljes gráfban is körmentességet eredményez.* Az algoritmus gyakorlatban is könnyen leprogramozható, a felhasználók számára egyszerűen megérthető és használható logika alapján működik. *A körmentességet az algoritmus a következő megoldással biztosítja:*

> Egy T_i tranzakció számára egy $G(V, E)$ gráf $N \in V$ csomópontja akkor és csak akkor foglalható le, ha az N-ből irányított utak mentén elérhető csomópontok és élek egyike sem foglalt még. A T_i tranzakció számára az összes ilyen, N-ből irányított utak mentén elérhető csomópontot és élt is le kell foglalni.

A teljes zárolási kéréseket elbíráló algoritmus ezek után metakóddal a következő módon adható meg:

Algoritmus 1. Konzisztenciaőrző, zárolási kéréseket elbíráló algoritmus 1. (továbbiakban A1):

```
Input: Ri - List of objects that Ti intends to write in graph G (modify,
delete)
Output: Gi sub-graph, the locking set of Ti. If Gi is empty then the lock
request must be denied.

1) Gi = Ri
//Temporary variables. A component can be edge or node:
2) edge e;  node N;  component c;
3) For (every edge of Gi) {
      a) e = current edge of Gi
      b) add e.source to Gi
      c) add e.sink to Gi
   }
4) For (every node of Gi) {
      a) N = current node of Gi
      b) For (every component of G) {
            i)  c = current component of G
            ii) if (pathExists(N, c)==TRUE) then add c to Gi
      }
   }
//Check lock compatibility, in case of incompatibility empty Gi:
5) For (every component of Gi) {
      a) c = current component of Gi
      b) If (c.isLocked()==TRUE) then {
            i)  Gi = {}
            ii) Exit loop
      }
   }
6) Return Gi
```

Az A1 algoritmus az R_i tranzakcióindítási kérés elbírálásakor – a 2. fejezetben ismertetett naiv, egyszerű ütközést figyelő algoritmussal megegyező módon – először a lefoglalandó részgráfhoz ad minden, a zárolási kérésben szereplő komponenst (1. lépés). Ezután a P1 értelmében a lefoglalandó elemek halmazához adja minden R_i-ben szereplő él végpontjait (2-3. lépés). A 4. lépésben kiterjeszti a foglalandó részgráfot minden olyan komponenssel, amelyek a már G_i részgráfban szereplő csomópontokból a gráfban irányított élek mentén elérhetők. Ehhez az algoritmus a `boolean pathExists(node N, component c)` függvényt használja, amelynek feladata ellenőrizni, hogy az első paramétereként kapott csomópontból, a második paramétereként kapott komponens (él vagy csomópont) irányított éleket követve elérhető-e[8]. A függvény kódját nem adom meg, az bármilyen útkereső algoritmussal létrehozható. Az 5. lépésben ellenőrzi, hogy a lefoglalásra szánt részgráf lefoglalása nem okoz-e ütközést a már futó tranzakciók részgráfjaival. Ehhez egyenként végigmegy a foglalásra szánt komponenseken és megnézi, hogy azok foglaltak-e már. Amennyiben valamelyik komponens már foglalt, kiüríti a lefoglalandó komponensek halmazát és befejezi az ellenőrzést. A 6. lépésben visszaadja az algoritmust használó zár menedzser számára a lefoglalandó részgráfot, amely üres részgráf

[8] Csomópontba vezető útnak csomópont, élbe vezető útnak él az utolsó eleme.

esetén megtagadja az indulást, nem üres részgráf esetén viszont lefoglalja a G_i halmaz komponenseit.

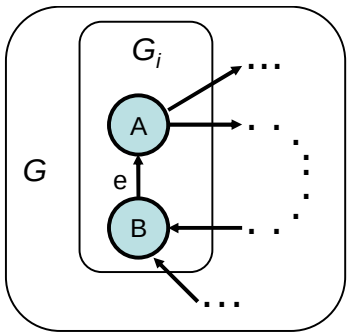
Az A1 algoritmus a G gráf olyan felosztását adja, melyben az egyidőben futó T_1, T_2,..., T_n tranzakciók G_1, G_2,..., G_n részgráfjaira igaz, hogy $G_i \cap G_j = \emptyset, 1 \le i, j \le n, i \ne j$. P1 implementálása miatt minden részgráf csomópontokkal van körülzárva.

Állítás:
Az A1 algoritmus a gráfok olyan részgráfokra való particionálását adja, melyeken belül ha biztosított a körmentesség, akkor a részgráfok összeállításaként kapott új gráfon belül sem lehet kör.

Bizonyítás:
A bizonyítás megmutatja, hogy amennyiben egy, az A1 algoritmussal felosztott gráfban kialakul kör, akkor a kör minden komponense egyazon részgráfon belül kell hogy legyen, vagyis nem lehet több részgráfon átívelő kör. Mivel a részgráfokon belüli körök kialakulása ellen a kliens oldali szerkesztőrendszerek védekeznek, emiatt kijelenthető lesz, hogy a teljes gráfban nem lehet kör.
Tegyük fel hogy egy G gráf G_i részgráfja a T_i szerkesztési tranzakció számára az A1 algoritmussal lett lefoglalva. Tegyük fel, hogy T_i létrehoz egy *e(B, A)* élt a G_i részgráfon belül, és hogy ez az él kört eredményez G-ben. A szerkesztési eset a 3-5. ábrán látható.

Megállapítások:

- Mivel T_i képes létrehozni az A és B csúcsok között új élt, ezért az A és B csúcsok biztosan a G_i részgráfon belül vannak. (Ezen a ponton kihasználom, hogy A1 implementálja P1-et)
- Mivel a *e(B, A)* él *G*-ben kört eredményez, ezért léteznie kell egy A→B irányított útnak. Ez az út a következő módokon jöhetett létre:
  1. Az út létezett T_i elindulása előtt. Ebben az esetben *az út minden komponensének* G_i-n belül kell lennie, mivel annak elemei elérhetők *A*-ból, és így azokat az A1 algoritmus a 4) lépésben G_i-hez kellett hogy adja. Vagyis ekkor a kör minden eleme G_i-n belül helyezkedik el.
  2. Az út valamelyik komponensét T_i hozta létre. Ebben az esetben az adott elem G_i része kell hogy legyen, hiszen egy újonnan létrehozott komponens mindig a létrehozó tranzakció számára kerül azonnal lefoglalásra. Vagyis a kör minden eleme ekkor is G_i-n belül helyezkedik el.



3-5. ábra: Ha a T_i tranzakció által a G_i részgráfon belül létrehozott él kört eredményez a gráfban, akkor a kör minden komponense G_i-n belül kell hogy legyen.

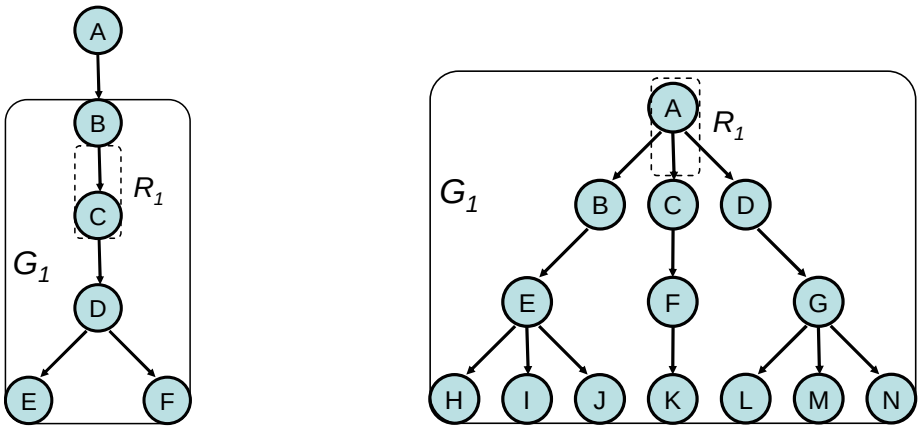
□

A fenti bizonyítással beláttam, hogy az A1 algoritmus segítségével megőrizhető a gráfok körmentessége még kollaboratív szerkesztés esetén is. Mivel az A1 algoritmus implementálja a P1-et, amelyről korábban beláttam, hogy védekezik szabad és többszörös bejövő élek ellen, ezért nincs szükség külön bizonyításra hogy ez a tulajdonság A1-re is igaz.

3.5.1. Az A1 algoritmus értékelése

Az A1 algoritmust használva egy kollaboratív workflow szerkesztőrendszer úgy képes megőrizni workflow alkalmazások gráf szintű szemantikai konzisztenciáját, hogy az egyidejű szerkesztések nem eredményeznek abortált tranzakciókat, a felhasználók nem okozhatnak egymásnak kárba veszett munkát. A foglalási logika egyszerűen átlátható, mivel az az élek irányításának megfelelően halad végig a gráfon. Viszont ha nem szabad minden komponens, amelyet az algoritmus egy tranzakció számára foglalásra szán, akkor nem foglalódik le semmi, a foglalási kérés elutasításra kerül és a felhasználónak el kell halasztania a munkáját. Mivel a 2. fejezetben szereplő algoritmushoz képest az A1 algoritmus általában jóval nagyobb részgráfokat akar lefoglalni, ezért az ütközés esélye is megnő. Ez a hátrány legerőteljesebben akkor jelentkezik ha:

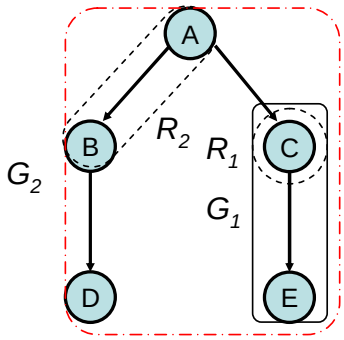
- A workflow kevés elágazást tartalmaz, pipeline-szerű: Ilyenkor egyetlen komponens foglalása is "végigszalad" a workflow-n, az elsőnek igénylő felhasználó számára a workflow nagy része lefoglalódik, másoknak alig marad valami. (Ld. 3-6. ábra (a) része.)
- A felhasználó a gráf „gyökeréhez" közel lévő komponenssel akar dolgozni, erre kér foglalást. Mivel az ilyen komponensből a gráf nagy része irányított élek mentén elérhető, a kérés a gráf nagy részének foglalását eredményezi. (Ld. 3-6. ábra (b) része.)



(a) Pipeline-szerű gráf **(b) Gyökér csomópont kérve szerkesztésre**

3-6. ábra: Példák olyan gráf struktúrákra és szerkesztési esetekre, melyekben az A1 algoritmus különösen nagy részgráfot foglal le.

Az A1 algoritmus másik gyengesége, hogy a részgráf kiterjesztés miatt még akkor is előfordulhat a foglalandó részgráfok közötti ütközés – és emiatt tranzakció-tiltás – ha a felhasználók valójában nem is akarnak ugyanazon komponenssel vagy gráfrésszel dolgozni. Egy ilyen példa látható 3-7. ábrán.



3-7. ábra: Példa olyan kollaboratív szerkesztési esetre melyet az A1 algoritmus tilt, holott a felhasználók munkája különböző gráfrészeket érint.
A T_2 tranzakció számára G_2 nem foglalható le, holott $R_1 \cap R_2 = \varnothing$.

A konklúzió, hogy ugyan A1 teljesíti a konzisztencia feltételeket, sok esetben nem eredményez hatékony kollaboratív munkát. Új, jobb zárolási kéréseket elbíráló algoritmusra van szükség, amely a példaként felhozott esetekben, és azokhoz hasonlókban lehetővé teszi az egyidejű munkát. Az új módszernek természetesen továbbra is biztosítania kell a gráf szintű konzisztenciát.

3.6. Zárolási kéréseket elbíráló algoritmus A2

Tegyük fel, hogy egy *A* felhasználó számára le kell foglalni egy részgráfot, amelynek bizonyos része, vagy akár az egésze egy *B* felhasználó számára már le van foglalva. Ilyen esetben A1 mindig ütközést jelez és *A* számára semmit sem enged lefoglalni. A 3-7. ábrán is látható volt, hogy ilyenkor nem feltétlenül azért van ütközés, mert a két felhasználó azonos gráf elemeket akar szerkeszteni. *A1-et használva előfordulhat, hogy az ütközés olyan komponenseken áll fenn, amelyeket a konzisztenciaőrzés miatt a rendszer adott a felhasználók részgráfjaihoz.* A hozzáadást ugyan nem lehet elkerülni, hiszen ezzel biztosított, hogy a részgráfokon futó tetszőleges tranzakció eredménye a gráfba visszaintegrálható tranzakció abortálás vagy kompenzáció nélkül. A foglalás okait viszont meg lehet egymástól különböztetni.

Az új algoritmus két féle zár típust használ és így megkülönbözteti a két féle okból lefoglalt komponenseket egymástól. Az egyik zárral olyan komponenseket jelöl melyeket a felhasználó valóban módosítani akar, a másikkal olyanokat, melyeket a rendszer konzisztenciaőrzés miatt foglal le. Az A1 algoritmus egy fajta zár típust ismert. Ezt a típust *user lock* néven megtartjuk. Az újonnan bevezetendő zár típus neve *system lock***.** A két zár típus jellemzői:

User lock:

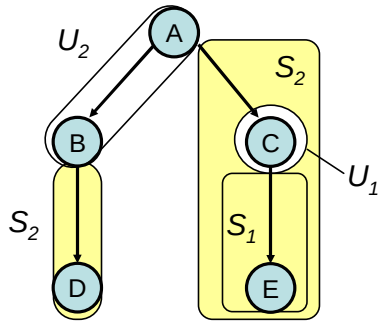
- Azokra az elemekre kerül, melyek azért vannak lefoglalva egy felhasználó számára, mert azokat ő a szerkesztőkörnyezetében szerkesztésre bejelölte.
- Egy felhasználó a saját user lock-kal rendelkező komponenseit szabadon szerkesztheti. (módosíthatja, törölheti)
- Új komponens a létrehozásakor user lock-kal lefoglalásra kerül a létrehozó tranzakció számára.
- Új él csak akkor hozható létre, ha a végpontjai user lock-kal lefoglalhatók. (Vagy másképpen: Két csomópont között új él létrehozásakor a csomópontok *user* lock-kal lefoglalásra kerülnek az él létrehozója számára.) Ennek oka, hogy a végpontokat

lefoglalni mindenképp kell, különben más tranzakció megszerezheti azokat, szabad élt, vagy többszörös bejövő élt eredményezve. System lock azért nem jó, mert – ahogy azt mindjárt látni fogjuk – system lock-kal rendelkező komponenst más felhasználó bizonyos esetekben szerkesztésre megszerezhet, ismételten esélyt kapva szabad él, vagy többszörös bejövő él létrehozására.

System lock:

- Azokra az elemekre kerül, amelyek a rendszer által azért lesznek lefoglalásra kiválasztva, hogy a gráf konzisztenciája megőrizhető lehessen tetszőleges szerkesztési tranzakciók esetén is.
- System lock-kal megjelölt komponenst a lock tulajdonosa nem szerkesztheti (nem módosíthatja, nem törölheti). Az ilyen zárral rendelkező komponensek a felhasználók számára még nem lefoglalt komponensként látszanak.
- System lock-kal megjelölt csomóponthoz a felhasználó élt *nem* köthet.

A kétféle zár típus használatát demonstrálja 3-8. ábra a 3-7. ábrán már megismert szerkesztési esettel.



3-8. ábra: User és system lock használata a 3-7. ábrán illusztrált szerkesztési esetben. Az U_1 és S_1 halmazokba tartozó komponensek a T_1 tranzakció számára user, illetve system lock-kal lefoglalt komponenseket tartalmazzák. Az U_2 és S_2 halmazokba a T_2 tranzakció számára user, illetve system lockkal lefoglalt komponensek tartoznak.

Adott egy *A* felhasználó, aki már lefoglalt bizonyos komponenseket a gráfból. Jön egy második, *B* felhasználó, aki bizonyos komponenseket kér a gráfból. A két különböző zár használata miatt most már többféle zár ütközés is előfordulhat. Ezek a következők:

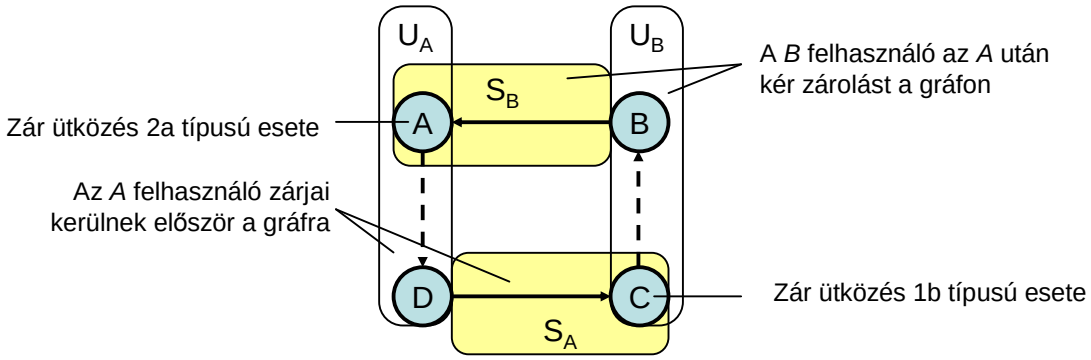
1. Az ütköző komponensek azért vannak *A* részgráfjában, mert ő azt szerkesztésre bejelölte a szerkesztőrendszerében, vagyis ***A user lock-kal bír a komponenseken***. Ezen belül további két eset lehetséges:

   1.a eset:
   B-nek azért került be az ütközést generáló komponens a lefoglalandó elemei közé, mert azt ő bejelölte a szerkesztőrendszerében, vagyis ***B-nek is user lock-ra lenne szüksége a komponensen***.
   - Ilyen esetben ***meg kell tagadni a zárolást B számára***, mert mindketten ugyanazt a komponenst akarják módosítani. Több felhasználó munkája egyazon komponensen nyilvánvaló ütközést eredményez. (A dolgozat 5. fejezetében tárgyalásra kerülő módszerekkel ez a kikötés feloldható.)

1.b eset:

B-nek azért került be a komponens a lefoglalandó elemei közé, mert azt a rendszer akarja zárolni számára a konzisztenciaőrzés miatt. Vagyis ***B-nek system lock-ra van szüksége*** a komponensen.

- Ha a zárolást ebben az esetben megtagadjuk, azzal biztosan konzisztenciát garantáló részgráfokat eredményezünk, hiszen úgy teszünk, ahogyan A1 is tesz: bármilyen ütközés esetén tilt. Viszont ezzel feleslegesen túl szigorúak lehetünk. *A zárolás ebben az esetben megengedhető, de csak akkor, ha a 2a pontban részletezett ütközési esetben tiltjuk a zárolást.* Ha mind itt az 1.b, mind a későbbi 2.a esetben engednénk a zárolást, akkor előfordulhatna olyan eset, hogy két tranzakció a saját user lock-kal lefoglalt gráfrészén dolgozik, azon belül nem alakít ki kört, viszont a teljes gráfban mégis kör jön létre. Egy ilyen esetre mutat példát a 3-9. ábra. Ahogy az ábrán látható, ha az 1.b és 2.a ütközés esetén is engedünk zárolást, akkor egy ilyen, inkonzisztenciát eredményező szituáció alakulhat ki. Ha viszont mindkét esetben tiltunk, akkor – ahogy az a későbbiekben bemutatásra kerül – túl szigorúak vagyunk. Optimális és elegendő megoldás az egyik esetre való engedés. Ezt az állítást a későbbiekben bizonyítom.



Jelmagyarázat:

U_A – A felhasználó User lock-ja
S_A – A felhasználó System lock-ja
U_B – B felhasználó User lock-ja
S_B – B felhasználó System lock-ja
Folyamatos élek – A zárolások előtt már létező gráf élek
Szaggatott élek – A és B által létrehozott élek

3-9. ábra: System lock és user lock használata egy inkonzisztens gráfot eredményező szerkesztési esetben.

2. Az ütköző komponens azért van *A* számára lefoglalva, mert azt a lockoló algoritmus a zárolandó elemek közé sorolta azért, hogy konzisztenciát biztosító részgráfok keletkezzenek. Vagyis olyan komponensről van szó, amelyet *A* nem akar módosítani és így az ***System lock-kal van A számára lefoglalva***. Ezen belül is két aleset lehet:

2.a eset:

B-nek azért kell a komponenst lefoglalni, mert azt ő szerkesztésre bejelölte, vagyis ***B-nek User lock-ra van szüksége***.

- Ahogy az 1b pontban írtam ekkor engedhetünk is, de meg is tagadhatunk. Ha engedünk, akkor az 1b esetben tiltani kell, ha megtagadunk, akkor 1b esetben lehet engedni.

2.b eset:

B-nek azért került be a komponens a lefoglalandó elemei közé, mert azt a rendszer akarja zárolni számára, vagyis ***B-nek System lock-ra van szüksége***.

- Ebben az esetben minden további nélkül engedélyezhető a zárolás, hiszen valójában sem *A*, sem *B* nem akar dolgozni az ütközést okozó komponenssel, azt csak a rendszer foglalja le azért, hogy ezzel más felhasználók szerkesztési lehetőségét korlátozza.

A fentieket összefoglaló zár kompatibilitási tábla a következő módon néz ki:

		Már meglévő zár típusa		
		Nincs zárolva	User (U)	System (S)
Újonnan kért zár	User	**Yes**	1.a eset: U-U ütközés **No**	2.a eset: S-U ütközés **!1.b eset**[9]
	System	**Yes**	1.b eset: U-S ütközés **!2.a eset**[10]	2.b eset: S-S ütközés **Yes**

(**Yes** jelenti, hogy engedélyezhető az újonnan kért zárolás, **No,** jelenti, hogy nem. "**!**" a negáció operátora.)

Zárolással kapcsolatos további megkötések, tulajdonságok:

1. Egy tranzakció számára U lock-kal lefoglalt részgráf csak csomópontokkal lehet körülzárva. Ez hasonló az A1 algoritmus tulajdonságához, és a P1-nek a következménye. Azért kell a felhasználó számára él létrehozáskor U lock-kal lefoglalni a végpontokat (S lock nem jó), mert ez biztosítja, hogy törlésre más felhasználó semmilyen körülmények között nem szerezheti meg.
2. Egy tranzakció számára U és S lock-kal együttesen lefoglalt részgráf szintén csomópontokkal van körülvéve. Ez annak a következménye, hogy az U lock-kal lefoglalt részgráfot a rendszer az A1 algoritmushoz hasonló módon úgy egészíti ki (ezúttal S lock-kal), hogy a részgráfhoz minden abból az élek mentén elérhető elemet hozzáad. Mivel egy ilyen út csak csomóponttal végződhet, (szabad él nem lehet a gráfban), ezért az eredményül kapott részgráf is csomópontokkal van körülvéve.
3. Az előző két pont következménye, hogy egy tranzakció számára S lock-kal lefoglalt részgráf ugyanazon tranzakció U lock-kal lefoglalt részgráfjához kapcsolódik, és az U lock-kal foglalt részgráf felől éllel/élekkel, más irány(ok)ból csomóponttal/csomópontokkal határolt.

Állítás:
A fent bevezetett U és S lock-okat, illetve kompatibilitási tábla szabályait használva semmilyen szerkesztési tranzakció nem tudja megtörni egy workflow gráf szintű konzisztenciaszabályait.

[9] !1.b azt jelenti, hogy ha az 1.b eset cellájába **No** kerül, akkor ebbe a cellába **Yes** kell hogy kerüljön, és fordítva.
[10] !2.a azt jelenti, hogy ha a 2.a eset cellájába **No** kerül, akkor ebbe a cellába **Yes** kell hogy kerüljön, és fordítva

Bizonyítás:
Az, hogy szabad élt és többszörös bejövő élt semmilyen tranzakció nem tud létrehozni triviális, hiszen az A2 zárolási kéréseket elbíráló algoritmus implementálja P1-et, ugyanúgy ahogy A1 teszi.

Az aciklikusság bizonyítása során belátom, hogy amennyiben egy, az A2 algoritmus által felosztott gráfban kialakul kör, akkor

3. a kör minden komponense vagy egyazon részgráfon belül kell hogy legyen,

vagy

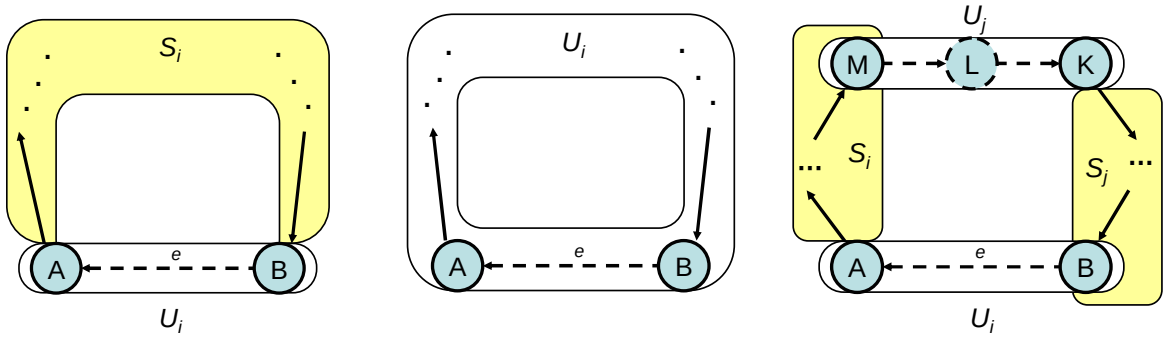
4. a kör létrehozásában részt vevő egyik tranzakciónak indulásakor U-S és S-U ütközést is kell hogy okozzon.

Ha egyetlen tranzakció részgráfján belül alakul ki kör, akkor ez ellen a kliens oldali szerkesztőrendszer védekezhet. A fenti kompatibilitási tábla értelmében viszont vagy az U-S, vagy az S-U ütközés tiltott, tehát nem futhat U-S és S-U ütközést *egyszerre* okozó tranzakció. A tranzakció hiányában viszont nem alakulhat ki kör. Vagyis belátom, hogy *az egyszerre U-S és S-U ütközést is okozó tranzakciók tíltása elégséges feltétel kör kialakulása elleni védelemhez.*

Tegyük fel, hogy egy *G* gráf $G_i = U_i \cup S_i$ részgráfja a T_i szerkesztési tranzakció számára az A2 algoritmussal lett lefoglalva. Tegyük fel, hogy T_i létrehoz egy *e(B, A)* élt a G_i részgráfon belül, és hogy ez az él kört eredményez *G*-ben.

Megállapítások:

- Mivel T_i képes létrehozni az A és B csúcsok között új élt, ezért az A és B csúcsok biztosan az U_i részgráfon belül vannak. (Ezen a ponton kihasználom, hogy A1 implementálja P1-et.)
- Mivel a *e(B, A)* él *G*-ben kört eredményez, ezért léteznie kell egy A→B irányított útnak. Ez az út a következő módokon jöhetett létre:
  4.1. Az út létezett T_i elindulása előtt. Ebben az esetben *az út minden komponensének* G_i-n belül kell lennie, mivel azok elérhetők *A*-ból. A komponensek közül lehet, hogy valamelyik S_i része, de lehet, hogy mindegyik U_i-be tartozik. Vagyis ekkor a kör minden eleme G_i-n belül helyezkedik el, így az *e* él létrehozását a T_i-t futtató szerkesztőrendszer megakadályozhatja. (Ld. 3-10. ábra (a) része.)
  4.2. Az út minden komponensét T_i hozta létre. Ebben az esetben az összes elem U_i része kell hogy legyen, hiszen egy újonnan létrehozott komponens mindig a létrehozó tranzakció számára kerül azonnal lefoglalásra. Vagyis a kör minden eleme G_i-n belül helyezkedik el, a T_i-t futtató szerkesztőrendszer megakadályozhatja az *e* él létrehozását. (Ld. 3-10. ábra (b) része.)
  4.3. Az út valamely komponensét egy $T_j \parallel T_i$ tranzakció hozta létre. Ekkor az U_j-ből U_i-be vezető gráf rész, és U_i-legalább egy komponense S_j-be kell hogy tartozzon, továbbá az U_i-ből U_j-be vezető gráf rész, és U_j-legalább egy komponense S_i-be kell hogy tartozzon. (Ld. 3-10. ábra (c) része.) Ennek következtésben ha $T_j < T_i$, akkor T_i elindulása U-S és S-U ütközést is okoz és emiatt indulása tiltott, ha pedig $T_j > T_i$, akkor T_j elindulása okoz U-S és S-U ütközést is és annak elindulása tiltott. Bármelyik eset is álljon fenn, a tiltott tranzakció munkája nélkül már nem jön létre a kör.



3-10. ábra: Kör kialakulásának lehetőségei egy A2 algoritmussal felosztott gráfban. Mindhárom esetben védekezik a rendszer a kör ellen.

A bizonyítás értelmében tehát akár az U-S, akár az S-U ütközés tiltott, az engedélyezett tranzakciók nem tudnak inkonzisztens gráfot létrehozni. Ehhez viszont szükség van egy eddig nem említett plusz feltételre: *a gráfon dolgozó tranzakciók S lock-jait nem elegendő tárolni, a már futó tranzakciók S lock-jait egy új tranzakció elbírálásakor újra kell generálni*. Ha ugyanis statikusan tárolnánk az S lock-okat és ez alapján állapítanánk meg az új tranzakció, és a már futó tranzakciók közötti ütközéseket, akkor előfordulhat olyan szerkesztési eset, melyben az A2 algoritmus engedi egy olyan tranzakció indulását, amely a már futó tranzakciókkal együtt képes kört létrehozni a gráfban. A 3-11. ábra egy példán keresztül demonstrálja ezt. A gráfot három tranzakció szerkeszti, melyek 1-1 él létrehozásával kört hoznak abban létre úgy, hogy egyik részgráfon belül sincs kör. Az első tranzakció páronként egyidőben fut a másik kettővel ($T_A \parallel T_B$; $T_A \parallel T_C$), a második és harmadik tranzakciók viszont egymás után futnak ($T_B \rightarrow T_C$). (Ld. még a 3-11. ábra bal oldali része.)

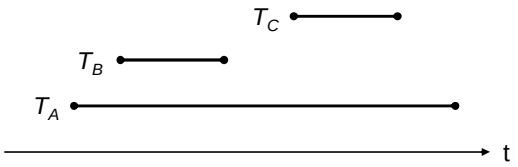
Az ábra a) része azt az mutatja, hogy milyen részgráfok alakulnak ki T_C elindulásakor, *ha az S lock-okat abban a pillanatban generálja a rendszer*. Eszerint T_A S lock-kal birtokolja a C, D és E csúcsokat, mivel T_B lezárásával a *cd* él létrejött. T_C indulása így U-S és S-U ütközést is okoz, vagyis T_C indulása tiltott.

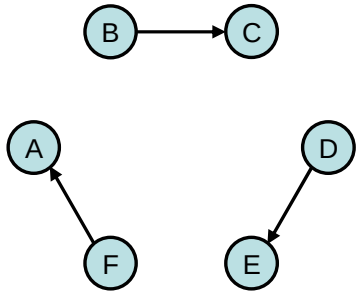
Az ábra (b) része azt mutatja, hogy milyen részgráfok alakulnak ki T_C elindulásakor, *ha a tranzakciók S lock-jait a rendszer T_A indulása óta tárolja*. Ekkor – mivel T_A elbírálásakor *cd* él még nem létezett – T_A nem birtokol S lock-ot a C, D és E csúcsokon. Ennek következtében viszont T_C indulása nem okoz S-U ütközést, vagyis indulása egy S-U ütközést engedő rendszerben lehetséges, de helytelen eredményre vezet.

$R_A=\{A, B\}$; $R_B=\{C, D\}$; $R_C=\{E, F\}$

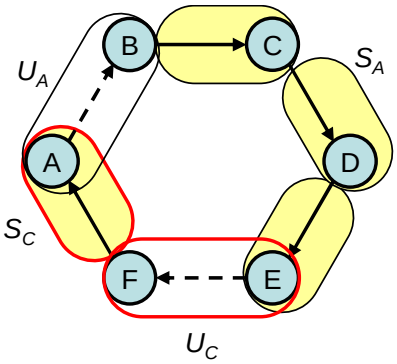
$T_A \| T_B \| T_C$; $T_A < T_B$; $T_A < T_C$; $T_B \rightarrow T_C$

$T_A = \{c(ab)\}$; $T_B = \{c(cd)\}$; $T_A = \{c(ef)\}$;

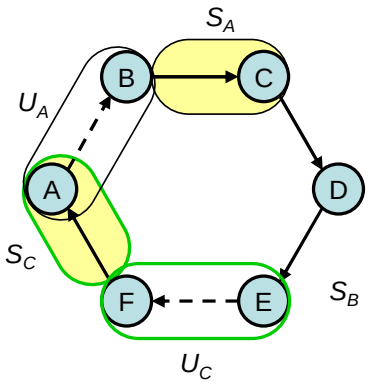




(a)



(b)



(c)

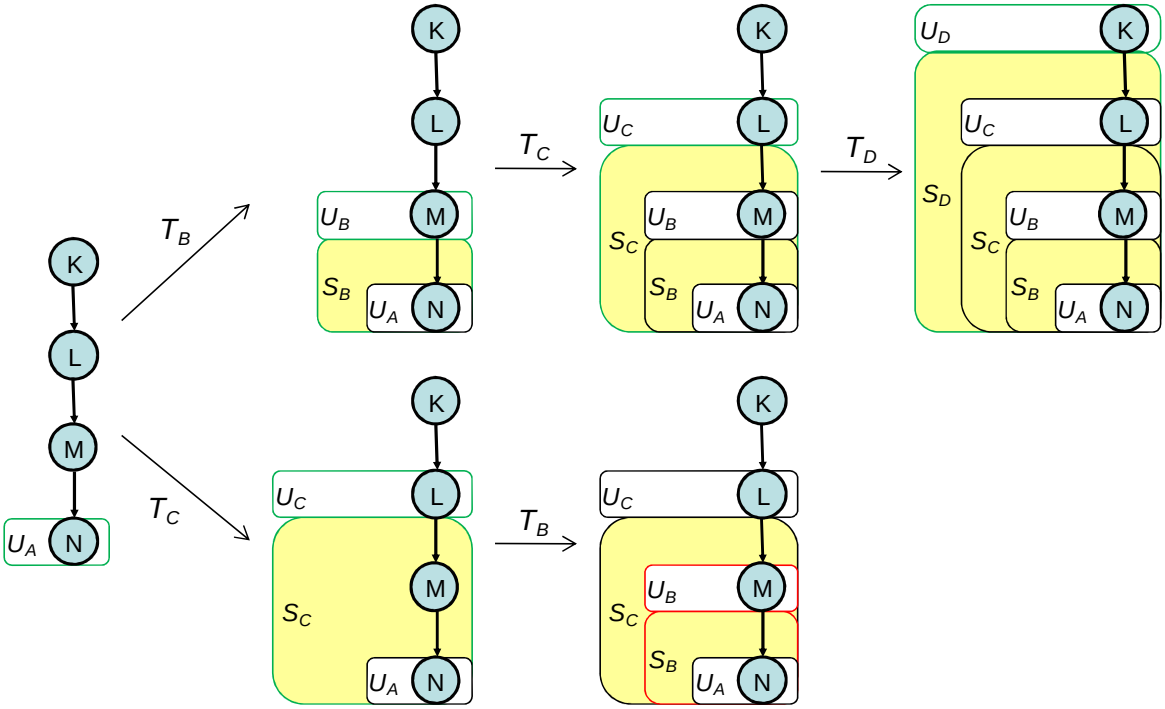
3-11. ábra: Körmentesség garantálásához az S típusú lock-okat minden tranzakcióindításkor újra kell generálni a gráfon, különben létrejöhetnek konzisztenciasértő tranzakciók. Az ábra (a) része a kiindulási gráfot mutatja. A (b) és (c) részek a gráf új állapotát és a zárakat mutatják az utolsó, vagyis a T_C tranzakció elindulásának pillanatában. A (b) esetben az S lock-ok tárolódnak a gráfon, a (c) esetben viszont csak a tranzakciók indulásakor kerülnek generálásra, nem tárolódnak permanens módon.

A 3-11. ábrán látható helyzetek elkerülése miatt tehát az A2 algoritmusnak minden tranzakció indulásakor újra kell generálnia a futó tranzakciók S lock-jait a gráfon. Ráadásul ezt nem akármilyen sorrendben, hanem a tranzakciók indulási sorrendjében kell ezt a generálást elvégeznie, ugyanis ellenkező esetben előfordulhat, hogy egy már futó tranzakció az újragenerálás során mind S-U, mind U-S ütközést okozza, és emiatt elvileg nem is futhatna. Ilyen esetre példa könnyen adható, ld. 3-12. ábra.

Az S lock-ok újragenerálásához kötött sorrendje miatt egy A2-t használó rendszernek azt is nyilván kell tartania, hogy a már futó tranzakciók milyen sorrendben indultak el. Ennek tárolása nem okoz problémát, például úgy, hogy a tranzakció indulásukkor egyedi, folyamatosan növekvő indexet kapnak, és ez az index tárolódik a gráf U lock-kal lefoglalt komponenseihez is. Az indexekkel ellátott U lock-ok alapján már egyértelmű hogy milyen sorrendben kell az S lock-okat generálni.

Egy tranzakció indulásakor a kliens oldali szerkesztőrendszer tetszőlegesen akár megkaphatja azt az információt, hogy számára milyen komponensek lettek S lock-kal kiválasztva, vagy sem. Ha megkapja az S lock-ra kiválasztott komponensei listáját, akkor elegendő az $U \cup S$ egyesített gráfrészen belül védekeznie kör kialakulása ellen, ugyanis ezen kívüli rész nem vehet részt semmiféle, az $U \cup S$ gráfrészbe tartozó komponenst tartalmazó körben. Amennyiben a szerkesztőkörnyezet nem kezeli, vagy nem kapja meg az S lock-olású részeket a zárolást végző menedzsertől, akkor lokális kör ellen az U lock-olású komponenseken

és a gráf permanens részén együttesen kell védekeznie – ugyanúgy ahogy az A1 algoritmus esetében.



$R_A=\{N\}$; $R_B=\{M\}$; $R_C=\{L\}$; $R_D=\{K\}$

R_D elbírálásakor T_A, T_B, T_C sorrendben újragenerálva a már meglévő S lock-okat egyik tranzakció sem okoz egyszerre S-U és U-S ütközést. T_A, T_C, T_B sorrendű újragenerálás esetén viszont T_B egyszerre U-S és S-U ütközést is okoz, vagyis nem is futhatna.

3-12. ábra: Az S lock-ok újragenerálását a tranzakciók indulási sorrendjének megfelelően kell végrehajtani, különben már futó tranzakciók eredményezhetnek tiltott ütközést.

A következőkben megadom az A2 algoritmus metakódját. A fenti bizonyítás értelmében az algoritmusnak vagy az U-S, vagy az S-U ütközést tiltania kell (az U-U ütközés mellett. Az S-S ütközést pedig bármikor engedheti.) A fenti kompatibilitási táblából viszont kétfajta ilyen algoritmus készíthető: az egyik az 1b típusú ütközést (U-S) engedi és a 2a típusút tiltja (S-U), a másik pedig épp fordítva: az 1b típusú ütközést (U-S) tiltja és a 2a típusút engedi (S-U). Egyébként mindkét algoritmusnak ugyanaz a feladata:

1. Megállapítani, hogy mekkora részgráfot kell lefoglalni a tranzakció számára U lock-kal.
2. Megállapítani a futó tranzakciók aktuális S részgráfjait.
3. Megállapítani, hogy mekkora részgráfot kell lefoglalni a tranzakció számára S lock-kal.
4. Összevetni a lefoglalási igényeket a már lefoglalt részgráfokkal, a zár kompatibilitási tábla alapján meghatározni, hogy lehet-e zárolni vagy sem. (A másolás már nem az algoritmus feladata. Az algoritmus továbbra is „mindent vagy semmit” elven működik: már egyetlen komponensnyi U-S és S-U átfedés esetén is tiltja a zárolást.)

Az A2 algoritmus első változata, melyben az U-S típusú ütközés engedett, az S-U típusú tiltott:

Újonnan kért zár	Már meglévő zár típusa: Nincs zárolva	User (U)	System (S)
User	**Yes**	**No** Eset 1a: U-U ütközés	**No** Eset 2a: S-U ütközés
System	**Yes**	**Yes** Eset 1b: U-S ütközés	**Yes** Eset 2b: S-S ütközés

```
Input: Ri - List of objects that Ti intends to write in graph G (modify,
delete)
Output: Ui sub-graph, the USER locking set of Ti. If Ui is empty then the lock
request must be denied.
Output: Si sub-graph, the SYSTEM locking set of Ti.

1) Ui = Ri;  Si = {}
//Temporary variables:
2) edge e;  node N;  transaction T;  sub-graph[]
   S[runningTransactions];  integer j;  component c;
3) For (every edge of Ui) {
      a) e = current edge of Ui
      b) add e.source to Ui
      c) add e.sink to Ui
   }
//Storing the SYSTEM locked subgraphs in S[].
4) For (every running transaction of G) {
   //Iterating through the transaction in the order of their start:
      T = current transaction;
      S[j++] = generateSystemLocks(T);
   }
//Storing the SYSTEM locked subgraph of Ti in Si:
5) For (every node of Ui) {
      a) N = current node of Ui
      b) For (every component of Ui) {
            i)  c = current component of Ui
            ii) if ((pathExists(N, c)==TRUE) and (c∉Ui)) then
                     add c to Si
      }
   }
//Checking lock U-U and S-U incompatibility, empting Ui, Si:
6) For every component of Ui
      a) c = current component of Ui {
      b) If ((c.isUserLocked()==TRUE) or (c∈S[])) then
            Ui = {};  Si = {}
   }
7) Return Ui, Si
```

Az algoritmus legelőször előkészíti U lock-kal való lefoglalásra a T_i által szerkesztésre kért komponenseket, és a kért éleket lezáró csomópontokat (1-3. lépés). Ezután előállítja egy tömb elemeiként azokat a részgráfokat, amelyek a gráfon már futó tranzakciók számára S lock-kal foglaltak (4. lépés). Ehhez egy `subgraph generateSystemLocks(transaction)` fejlécű függvényt hív meg egyenként a már futó tranzakciókra. Ennek a függvénynek a

paraméterként kapott tranzakció számára a gráfon tárolt U lock-ok alapján elő kell állítania a gráf S lock-kal lefoglalt részeit. Mivel ez teljesen úgy működik, ahogy az 5. lépésben a T_i tranzakció számára ezt megmutatom, ezért a függvény metakódját külön nem adom meg. Az 5. lépés után tehát ismertek a T_i számára U és S lock-okkal lefoglalandó részgráfok, továbbá a már futó tranzakciók számára adott S lock-kal lefoglalt részgráfok. Ekkor, a 6. lépésben az algoritmus megnézi a zár kompatibilitási tábla alapján hogy engedhető-e a foglalás, vagyis hogy valamelyik T_i által U lock-kal kért komponens nem okoz-e U-U vagy S-U ütközést. Ha bármelyik okoz, akkor üres részgráfokat ad vissza, ha egyik sem okoz, akkor a korábbi lépésekben előállított U_i és S_i részgráfokat. Ezek után a zároló menedzser el tudja végezni az U_i halmazba tartozó komponenseken az U lock-kal való lefoglalásokat, majd vissza tudja adni az indulást kért tranzakciónak az U_i, és ha akarja az S_i részgráfot is. Az S lock-okat nem kell tárolni.

A fentihez hasonlóan elkészíthető ***az A2 algoritmus második változata,*** melyben az U-S típusú ütközés tiltott, az S-U típusú ütközés viszont engedett:

Újonnan kért zár	Már meglévő zár típusa: Nincs zárolva	Már meglévő zár típusa: User (U)	Már meglévő zár típusa: System (S)
User	**Yes**	**No** Eset 1a: U-U ütközés	**Yes** Eset 2a: S-U ütközés
System	**Yes**	**No** Eset 1b: U-S ütközés	**Yes** Eset 2b: S-S ütközés

```
Input: Ri - List of objects that Ti intends to write in graph G (modify,
delete)
Output: Ui sub-graph, the USER locking set of Ti. If Ui is empty then the lock
request must be denied.
Output: Si sub-graph, the SYSTEM locking set of Ti.
1) Ui = Ri;  Si = {}
//Temporary variables:
2) edge e;  node N;  transaction T;  sub-graph[]
   S[runningTransactions];  integer j;  component c;
3) For (every edge of Ui) {
      a) e = current edge of Ui
      b) add e.source to Ui
      c) add e.sink to Ui
   }
//Storing the SYSTEM locked subgraphs in S[]:
4) For (every running transaction of G) {
   //Iterating through the transaction in the order of their start:
      T = current transaction;
      S[i++] = generateSystemLocks(T);
   }
//Storing the SYSTEM locked subgraph of Ti in Si:
5) For (every node of Ui) {
      a) N = current node of Ui
      b) For (every component of Ui) {
            i)  c = current component of Ui
            ii) if ((pathExists(N, c)==TRUE) and (c∉Ui)) then
                        add c to Si
      }
   }
```

```
//Checking lock U-U incompatibility, empting Ui, Si:
6) For every component of Ui
     c) c = current component of Ui {
     d) If (c.isUserLocked()==TRUE) then {
          i)  Ui = {};  Si = {}
          ii) Return Ui, Si
     }
   }
//Checking lock U-S incompatibility, empting Ui, Si:
7) For every component of Si
     e) c = current component of Si {
     f) If (c.isUserLocked()==TRUE) then {
          i)  Ui = {};  Si = {}
          ii) Return Ui, Si
     }
   }
8) Return Ui, Si
```

Az első változathoz képest a második csak a zárolási ellenőrzés lépéseiben jelent újdonságot. Mivel itt mind az U_i , mind az S_i halmazok elemeire kell megnézni, hogy nincsenek-e még U lock-kal foglalva, ezért ehhez két ciklusra van szükség (6-7 lépések). Az első ciklus az U_i , a második az S_i halmazon megy végig. Akár U -U, akár S-U ütközést találnak üres halmazokat ad az algoritmus vissza, ellenkező esetben a korábbi lépésekben előállított U_i és S_i részgráfokkal tér vissza. Ezek után a zároló menedzser el tudja végezni az U_i halmazba tartozó komponenseken az U lock-kal való lefoglalásokat, majd vissza tudja adni az indulást kért tranzakciónak az U_i, és ha akarja az S_i részgráfot is. Az S lock-okat itt sem kell tárolni.

3.6.1. Értelmezés, és az A2 algoritmus újrafogalmazása

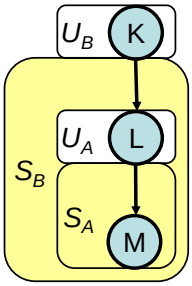
Az előző részben adott bizonyítás szerint biztos, hogy egy olyan tranzakció, amelynek esélye van a gráf körmentességének megtöréséhez indulásakor U-S és S-U ütközést is okoz a gráfon. Az ilyen tranzakciók kiszűrésének fent ismertetett két módozata hogy tiltjuk vagy az S-U, vagy az U-S ütközést okozó tranzakciókat. Akár az első, akár a második algoritmus változattal is bíráljuk el az indulást kért tranzakciót, a teljes gráf konzisztenciája tetszőleges szerkesztés esetén is megmarad tranzakcióabortálás és tranzakciókompenzáció nélkül.

A fenti két algoritmus változat között minden új tranzakcióindulási kérés elbírálásakor szabadon választhatunk. Egyazon gráfot szerkeszthetik mind az első, mind a második algoritmus változattal engedett tranzakciók, sőt a különböző változatok által engedett tranzakciók akár egyidőben is élhetnek a gráfon. Ennek oka, élt létrehozó szituáció csak együttes U-S és S-U ütközéssel jöhet létre. Mivel sem az első, sem a második változat nem enged ilyen tranzakciót, ezért az engedett tranzakciók egymásra nincsenek hatással, együttesen sem okozhatnak kört még „kevert” algoritmushasználat esetén sem.

Ugyan a két algoritmusváltozat bármelyike használható egy-egy lefoglalási kérés elbírálására, az eredményt tekintve közel sem mindegy hogy egy adott szituációban egy tranzakciót az egyik, vagy a másik algoritmussal bírálunk-e el, mivel előfordulhat, hogy az egyik engedi az indulást, a másik viszont nem. Példaként a 3-13. ábra3-12 mutat két tranzakciót (T_A es T_B) melyek egyazon gráfon kérnek zárolást. A T_A tranzakció a gráf L csomópontját szeretné szerkeszteni, T_B tranzakció pedig a K csomópontot. ($R_A = \{L\}$, $R_B = \{K\}$) Az ábra felső része mutatja az ezekhez a tranzakcióindítási kérésekhez generált U és S részgráfokat.

Az ábra alján lévő két táblázat mutatja, hogy az A2 zárolási kéréseket elbíráló algoritmus két változata hogyan reagál ezekre a kérésekre, ha azok R_A , R_B sorrendben érkeznek be ((a)

táblázat), illetve ha fordítva, R_B , R_A sorrendben érkeznek be ((b) táblázat). A táblázatok második oszlopában látható, hogy milyen sorrendben kerülnek zárak az adott esetben a gráf csomópontjaira. Az (a) esetben az *L* csomópontra U, S sorrendben kerülnek zárak, vagyis T_B elindulása U-S ütközést okoz. Az ilyen ütközést az első algoritmusváltozat engedi, a második nem, vagyis ekkor T_B az első változattal elbírálva indulhat, a másodikkal elbírálva viszont nem. A (b) esetben az *L* csomópontra S, U sorrendben kerülnek zárak, vagyis ekkor T_A elindulása S-U ütközést okoz. Az ilyen ütközést viszont a második algoritmusváltozat engedi, míg az első tiltja. A példából az látszik, hogy egy adott indulási kérés kimenetele szempontjából egyáltalán nem mellékes hogy az 1. vagy a 2. A2 változatot használja-e a rendszer. Míg az egyik engedi a párhuzamos szerkesztést, a másik feleslegesen tilthatja azt. A 3-13. ábrán látható példához hasonló szerkesztési eset természetesen bonyolultabb gráffal és kettőnél több tranzakcióval is készíthető.



Kérések sorrendje: R_A , R_B

Csomópont	Zárak csomópontra helyezésének időbeli sorrendje		Algoritmus első változat enged / tílt	Algoritmus második változat enged / tílt
K	U_B		✓	✓
L	$\boldsymbol{U_A}$	$\boldsymbol{S_B}$	✓	✗
M	S_A	S_B	✓	✓

(a)

Kérések sorrendje: R_B , R_A

Csomópont	Zárak csomópontra helyezésének időbeli sorrendje		Algoritmus első változat enged / tílt	Algoritmus második változat enged / tílt
K	U_B		✓	✓
L	$\boldsymbol{S_B}$	$\boldsymbol{U_A}$	✗	✓
M	S_B	S_A	✓	✓

(b)

3-13. ábra: Az A2 algoritmus két változatának különbsége a gyakorlatban.

A fenti okfejtés eredménye, hogy *egy tranzakció engedhető, ha akár az első, akár a második algoritmusváltozat engedi azt*. Vagyis a felhasználók számára az optimális, legnagyobb párhuzamosságot eredményező megoldás egy olyan egyesített A2 algoritmus amely, a fenti két változatot egyesíti, és csak akkor tiltja egy tranzakció indulását, ha azt mind az első, mind a második változat tiltja. Viszont bármelyik változat engedi az indulást az egyesített A2 algoritmus is engedi azt.

```
Input: Ri - List of objects that Ti intends to write in graph G (modify,
delete)
Output: Ui sub-graph, the USER locking set of Ti. If Ui is empty then the lock
request must be denied.
Output: Si sub-graph, the SYSTEM locking set of Ti.

1) (Ui, Si) = algorithm1(Ri)
2) If (Ui=={}) then
      (Ui, Si) = algorithm2(Ri)
3) Return Ui, Si
```

Az egyesített A2 algoritmus rendkívül egyszerű. Először meghívja az első változatot, ha az nem engedi a zárolást, akkor meghívja a második változatot. Akár az egyik, akár a másik változat is engedi a szerkesztést, az egyesített algoritmus megtalálja és ennek megfelelően szintén engedi azt. A továbbiakban ezt az egyesített algoritmust fogom A2-nek hívni.

3.7. Zárolási algoritmus A3

Az előzőekben tárgyalt, egyesített A2 algoritmus (mostantól ezt hívom A2 algoritmusnak) csak akkor tagad meg egy tranzakciónak indulást, ha annak zárjai egyszerre S-U és U-S ütközést is produkálnak. Az egymagában, vagy S-S ütközéssel egyidőben S-U vagy U-S ütközést okozó tranzakciókat mind engedi. (Az U-U ütközést okozókat pedig nyilvánvalóan tiltja.) *Vagyis egy már létező S zárra egy új tranzakció akár U, akár S foglalást rátehet egészen addig, amíg nem akar ezzel egyidőben egy, már szintén létező U zárra S foglalást is tenni.* Vagyis az S lock-ok generálása tulajdonképpen felesleges, és kidolgozható egy olyan módszer, amely csak U lock-ot használ és a kettős ütközések lehetőségét ismeri fel a S lock-ok újragenerálása nélkül. *De hogyan lehet a kettő ütközések előfordulását S lock-ok nélkül felismerni?*

Az A2 algoritmus esetén U-S vagy S-U zár ütközés tulajdonképpen olyan szituációkban fordul elő, amikor egy *A* tranzakció számára U lock-kal már lefoglalt részgráf, és egy újonnan induló *B* tranzakció számára U lock-kal lefoglalandó *részgráf* között irányított út fut. Ilyen esetben az irányított út elemei S lock-kal vannak/lesznek lefoglalva vagy az *A*, vagy a *B* tranzakció számára (attól függően, hogy melyik irányba fut az út), és emiatt S-U vagy U-S ütközés fog előfordulni (szintén attól függően, hogy melyik irányba fut az irányított út). Hogy a *B* tranzakció egyszerre S-U és U-S ütközést is produkáljon, ahhoz mind az *A* tranzakció U foglalású részgráfjából B tranzakció U foglalású részgráfjába, mind onnan visszairányított utak kell, hogy vezessenek. Ezek az utak nem feltétlenül összefüggőek a részgráfokon belül, hiszen kör még nem létezik a gráfban! Összefüggőek azonban ezek az utak a részgráfokon kívül.

Ennek következtében a tranzakció-menedzsernek *elég nyilvántartania a gráfon az U zárral lefoglalt részeket, továbbá egy új lock-olási kérés elbírálásánál ellenőriznie, hogy a most U zárral lefoglalandó részgráf és a gráf fennmaradó része együtt kört ad-e. Ha igen, akkor tiltani kell a zárolást, ha nem, akkor elindulhat az új tranzakció.* A lefoglalás közben természetesen a szabad élek elleni védelem miatt figyelnie kell arra, hogy ha a tranzakció megkap egy élt, akkor megkapja annak végpontjait is. Ha ez zár ütközés miatt nem lehetséges, akkor a tranzakció nem indulhat el és számára semmi sem kerül lefoglalásra.

Ez az új, mostantól A3-nak hívott zárolási kéréseket elbíráló algoritmus metanyelven leírva a következőképpen néz ki:

```
Input: Ri - List of objects that Ti intends to write in graph G
       (modify, delete)
Output: Gi sub-graph, the locking set of Ti. If Gi is empty then
        the lock request must be denied.

1) Gi = Ri
//Temporary variables:
2) edge e;  component c;  transaction T;  sub-graph Gt;
3) For (every edge of Gi) {
     a) e = current edge of Gi
     b) add e.source to Gi
     c) add e.sink to Gi
   }
```

```
4) For every component of Gi {
      g) c = current component of Gi
      h) If (c.isUserLocked()==TRUE) then
            Return Gi = {};
   }
//Creating contiguity graph:
5) graph CG = G;
6) For every running transaction of G {
      i) T = current transaction of G;
      j) Gt = T.lockedComponents();
      k) CG = graphReduce(CG, Gt);
   }
7) CG = graphReduce(CG, Gi);
//Checking for cycle:
8) If (cycleExist(CG)==TRUE) then
      Gi = {}
9) Return Gi
```

Az algoritmus a korábbiakhoz hasonló módon a szabad élek elleni védekezés miatt zárolandó részgráf kiterjesztésével kezdődik (1-3 lépés). Ez után ellenőrzi, hogy a lefoglalandó komponensek közül valamelyik zárolt-e. Ha igen akkor üres részgráffal tér vissza. Ha nem, akkor megvizsgálja a már lefoglalt, és a most lefoglalandó részgráfok gráfban betöltött helyét. Ehhez létrehozza a gráf úgynevezett *szomszédossági gráfját*. A szomszédossági gráf egy olyan gráf, amelyben

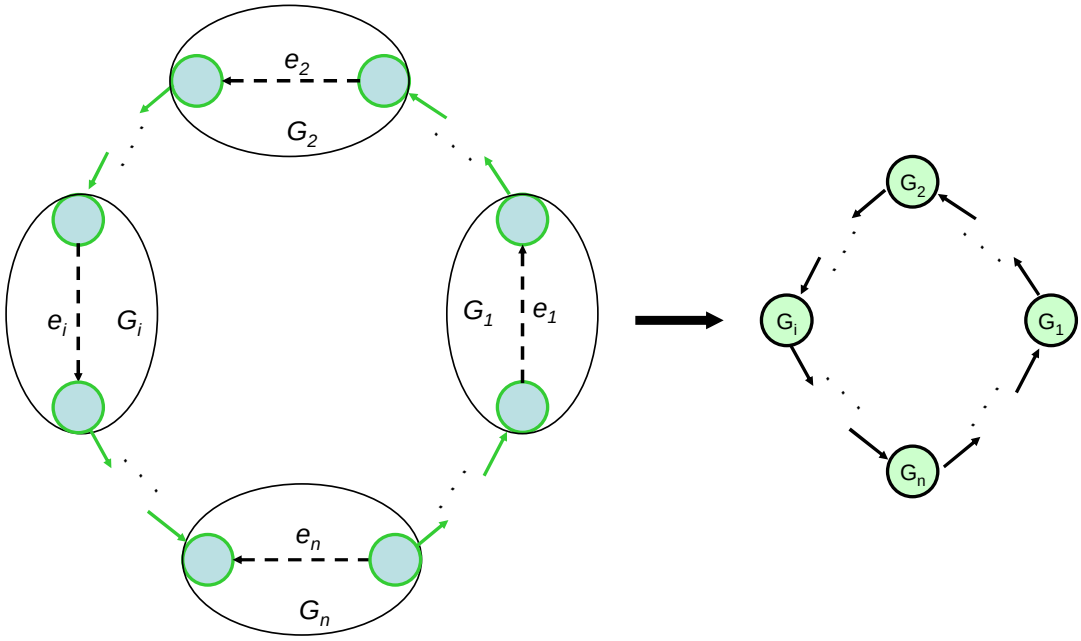
1. A gráf zárolatlan csomópontjai mind átkerülnek
2. A gráf már zárolt részgráfjai helyett egy-egy csomópont szerepel
3. A zárolatlan csomópontok között futó élek mind átkerülnek
4. A gráf egy *A* zárolatlan csomópontja és egy *B* zárolt csomópontja között futó él szerepel a szomszédossági gráfban az A csúcsnak megfelelő csomópont, és a B csúcsot magában foglaló részgráf csomópontja között futó élként.

A szomszédossági gráf kiindulópontja a teljes gráf (5. lépés), melyben 1-1 csomóponttal kerül behelyettesítésre először a már lefoglalt részgráfok (6. lépés), azután a most foglalandó részgráf (7. lépés). A szomszédossági gráf tehát normál csomópontokat és részgráfokat reprezentáló csomópontokat tartalmaz. A behelyettesítést az algoritmus egy `graph graphReduce(graph, sub-graph)` függvény segítségével végzi. A függvény metakódját nem adom meg, annak létrehozása triviális: a fenti felsorolás 2. és 4. pontja szerint le kell cserélnie a gráfban a paraméterként kapott részgráfot egyetlen csomópontra. A szomszédossági gráf létrehozása után az algoritmus megvizsgálja, hogy van-e a gráfban kör. Ha létezik kör, akkor az indulást kért tranzakció kört hozhat létre a gráfban. Ha a szomszédossági gráfban nincs kör, akkor a tranzakciónak nincs esélye kör létrehozására.

Állítás:
A fent bevezetett A3 algoritmus által engedett szerkesztési tranzakciók nem tudják megtörni egy workflow gráf szintű konzisztenciaszabályait.

Bizonyítás:
Az, hogy szabad élt és többszörös bejövő élt a tranzakciók nem tudnak létrehozni triviális, hiszen az A3 zároló algoritmus implementálja P1-et (ld. 3. lépés).

Tegyük fel, hogy egyazon gráfot szerkesztő n db egyidőben futó tranzakció T_1, T_2, … T_m. Tegyük fel, hogy T_1, T_2, … T_n $(n \leq m)$ tranzakció ezek közül kört eredményez a gráfban, és e_1 az az él, amelyet a T_1 tranzakció, e_2 az amelyet a T_2 tranzakció, … e_n pedig az amelyet a T_n tranzakció ad a körhöz. (Ld. 3-14. ábra bal oldala)



3-14 ábra: Gráfok körmentességének biztosítása szomszédossági gráf segítségével.

Ahhoz, hogy az n db tranzakció ezeket az éleket valóban hozzáadhassa a körhöz, ahhoz zárral birtokolniuk kell legalább 2-2, korábban még nem összekötött csomópontot a gráfból. Mivel az egyes részgráfokon belül hozzáadott e_1, e_2, ..., e_n élek a teljes gráfban kört eredményeznek, ezért a G_1, G_2, ..., G_n részgráfok között a tranzakciók elindulása előtt már utak kellett, hogy fussanak. Ha nem így lenne, és egy G_i és G_j részgráf közötti utat valamely T_k tranzakció hozna létre, akkor a G_k-nak G_i-vel és G_j-vel is átfedésben kell lennie. Ez csak akkor lehetséges, ha $G_i = G_j = G_k$, ugyanis minden más esetben A3 tiltaná T_k elindulását. Ha $G_i = G_j = G_k$ akkor viszont $T_i = T_j = T_k$, vagyis nem hozhatta létre egyik lefoglalt részgráf közötti utat sem futó tranzakció.

A kört eredményező n db tranzakció közül az utolsónak az indulásakor tehát a körből már léteztek a G_1, G_2, ..., G_{n-1} részgráfok, az őket összekötő utak, a G_n részgráfból legalább két csomópont, és az azokat a többi részgráfhoz kapcsoló 1-1 él. Az egyetlen esetleg hiányzó részek az e_1, e_2, ..., e_n élek. Mivel a T_1, T_2 ... , T_{n-1} tranzakciók már elindultak őket leállítani nem lehet, és így bármikor létrehozhatják az e_1, e_2, ..., e_{n-1} éleket – ha eddig még nem tették. Az A3 algoritmus ekkor a T_n elindulását tiltja, mivel a körből már meglévő komponensek a gráfból létrehozott szomszédossági gráfban kört alkotnak – ahogyan az a 3-14. ábra jobb oldalán látszik. T_n hiányában az e_n él nem jöhet létre, enélkül pedig nem alakul ki kör a gráfban.

4. Algoritmusok összehasonlítása

A dolgozat második felében értékelem az A1, A2 és A3 zárolási kéréseket elbíráló algoritmusokat, megállapítom, hogy melyik a legcélszerűbben használható, illetve hogy lehet-e még ezeknél is célszerűbb algoritmust készíteni. Egy *zárolási kéréseket elbíráló algoritmussal szemben támasztott legfontosabb kritérium, hogy minél több felhasználót engedjen az általa kezelt gráfhoz hozzáférni, minél több ember tudja a megosztott workflow-kat egyidőben szerkeszteni természetesen a konzisztencia feltételek betartatása mellett*. Mivel a gráf szintű konzisztencia kritériumok megtartása a workflow-k későbbi futtathatósága miatt alapfeltétel, ezért *az ideális zároló algoritmus csak azokat a kollaborációkat tiltja, amelyek valóban inkonzisztens gráfot eredményeznek, minden más kollaboratív szerkesztési esetet viszont enged, így eredményezve a lehető legnagyobb párhuzamosságot a rendszerben.*

Ebben a fejezetben analizálom a workflow-kon zajló lehetséges szerkesztési kollaborációkat és bevezetek olyan metrikákat, amelyek segítségével megállapítható, hogy az eddig tárgyalt, illetve később bevezetésre kerülő zároló algoritmusok milyen szintű párhuzamosságot tesznek lehetővé ezen esetek során. Megállapítom, hogy az A1, A2 és A3 algoritmusok mikor eredményeznek ideális, és mikor kevésbé ideális párhuzamosságot. Ezelőtt viszont ismertetem a kapcsolódó eredményeket csoportmunka alkalmazások értékelése és javítása témából.

4.1. Kollaboratív rendszerek értékelése – szakirodalmi áttekintés

A CSCW témakörén belül a kezdeti évtizedek munkái erősen fókuszáltak új csoportmunka alkalmazások és kapcsolódó protokollok létrehozására. A 90-es évekre ezek száma elérte azt a kritikus szintet, hogy a kutatások hangsúlya a meglévő rendszerek értékelésére és az értékelések alapján történő javítására helyeződhetett (Araujo et al, 2002). Csoportmunka környezetek értékelésének legszélesebb körben használt módszere a használat közbeni megfigyelés, és az így begyűjtött információk alapján történő elemzés. Attól függően, hogy ez a folyamat mennyire természetes vagy mesterséges körülmények között történik, Pinelle és Gutin osztályozása szerint tovább bontható (Pinelle, Gutwin, 2000) (Ld. Táblázat 2).

A kategorizálás két dimenzió mentén történik: a vizsgálat helye lehet mesterséges – vagyis laboratórium – vagy lehet természetes, amikor is a csoportmunka szoftver tesztelése a tényleges felhasználás helyén, cégnél vagy irodában történik. A másik dimenzió a tesztelési folyamat irányítottságára, a csoportmunka szoftverrel elvégzett munka meghatározottságára vonatkozik. Amennyiben az előre rögzített nagy befolyásolásról beszélünk, amennyiben egyáltalán nem, vagy csak kis mértékben rögzített, tanulmányról van szó. A valós környezetben történő tesztelés nyilvánvalóan nagyobb költséggel jár, mint a laboratóriumi körülmények között zajló, hiszen teljesen működő csoportmunka alkalmazást és annak installálását, megtanítását, fenntartását igényli. Viszont pontosabb, az adott felhasználó kör számára relevánsabb eredményt is ad.

Táblázat 2. Csoportmunka környezetek vizsgálati módszereinek osztályozása. (Pinelle, Gutwin, 2000) alapján.

		Befolyásolás mértéke	
		Nagy	Alacsony, vagy nincs
Környezet	Természetes	Terepkísérlet	Tereptanulmány
	Mesterséges	Laboratóriumi kísérlet	Laboratóriumi tanulmány

A felhasználók munka közbeni megfigyelése komoly probléma, ugyanis alapesetben több felhasználó, időben szinkronizálatlan módon használ egy csoportmunka rendszert. A befolyásolás éppen ezt a hatást hivatott csökkenteni valamilyen szinkronizációval, vagy előre megszabott felhasználói forgatókönyvekkel.

Ugyan Pinelle és Gutin osztályozása széles körben hivatkozott és elfogadott, vannak olyan munkák, amelyek vitatják a laboratóriumi kísérletek létjogosultságát (Araujo et al, 2002) (Pinelle, 2000)(Grudin, 1988). A laboratóriumi tesztek elleni legfőbb érv, hogy azok idealizált, és általános eseteket fednek le, holott a kollaboratív szoftverek használata csoportról-csoportra változik, erősen befolyásolja a munkahelyi környezet, ezért csak a célcsoport által és csak valós körülmények között végezhető. Az ilyen tesztek viszont rendkívül drágák, ráadásul hosszú átfutási idejűek, éppen ezért a csoportmunka rendszereknek csak elenyésző része esett át rajtuk.

Maga az értékelés általában valamilyen metrikákat eredményez, melyeket vagy előre rögzítenek, vagy a felmérés után szabnak meg (Neale et al, 2004). A metrikák értéke a csoportmunka szoftverek felhasználóitól kérdőívek segítségével gyűjthető be, vagy a rendszer fejlesztői is megadhatják például a felhasználókkal készített személyes interjúk vagy videós megfigyelés alapján (Neale et al, 2004)(Haynes et al, 2004). A leggyakrabban használt metrikák a csoportmunka szoftverrel elkészített entitás (dokumentum, rajz, stb.) minősége, az entitás fejlesztésével töltött idő hossza, ennek aránya az egyfelhasználós létrehozáshoz képest, mennyi rendszeren belüli, és azon kívüli kommunikációra volt szükség a fejlesztés során (Monk et al, 1996)(Nam, Wright, 2001).

Ereback és Hook munkája (Ereback, Hook, 1994) az egyfelhasználós szoftverek értékelésére használható „cognitive walkthrough” (CW) módszert terjeszti ki csoportmunka irányba. A CW módszerrel (Polson et al, 1992) egy szoftver felhasználók számára nyújtott interfészét lehet aránylag gyorsan és olcsón értékelni úgy, hogy egy csoport tapasztalt fejlesztő az interfészeken végigmegy abban a sorrendben, ahogyan azt a felhasználók munkájuk közben tennék, és minden lépés után összehasonlítják a szoftver által kínált felületeket a felhasználók céljai miatt elvártakkal. Ereback és Hook módszerüket egy találkozó-szervező csoportmunka szoftveren próbálták ki, és eredményük azt mutatja, hogy az ilyen módszer nem ad eléggé átfogó képet egy csoportmunka jóságáról, sőt, a felhasználók szinkronizálatlansága miatt kétséges, hogy minden elképzelhető együttműködési eset vizsgálatra került-e.

Baeza-Yates és Pino munkája (Baeza-Yates, Pino, 1997) az egyetlen cikk kollaboratív rendszer jóságának formális értékeléséről. A munka egy többváltozós függvényt definiál, amely egy kollaboratív csoport taglétszáma és a kapott eredmény minősége közötti kapcsolatot hivatott ábrázolni. A függvény segítségével elméletileg maximalizálni lehet a csoportmunka eredményét úgy, hogy ehhez a lehető legkevesebb emberre legyen szükség. Az egyenlet használata azonban a gyakorlatban szinte lehetetlen: szinte nincs olyan kollaboratív munkaterület ahol megadható lehet a megosztott entitás újabb és újabb verziói közötti javulás mértéke, vagy az, hogy a maga az entitás minősége mikor éri el az elfogadható szintet. Ez igaz a kollaboratívan fejlesztett workflow-kra is, és így kollaboratív workflow fejlesztés javítására sem lehet használni ezt a módszert.

4.2. Kollaboratív szerkesztési esetek analízise

A fent említett munkák mindegyike – az utolsó, Baeza-Yates és Pino cikke kivételével – mind gyakorlati oldalról közelíti meg a csoportmunka szoftver értékelést és javítást. Ugyan ilyen módon egy rendszer több oldalról is vizsgálható, a vizsgálat maga rendkívül költséges, időigényes, sok felhasználó és esettanulmány bevonását igényli. Ebben a fejezetben a zárolás

alapú, kollaboratív workflow fejlesztőrendszert szeretném célszerűség szempontjából vizsgálni, és jobbá tenni. Mikor mondhatjuk, hogy egy A zároló algoritmust használó kollaboratív szerkesztőrendszer célszerűbb, mint egy B algoritmust használó? Intuitív, informális válasz:

> *A célszerűbb, mint B, ha több felhasználót enged egyidejűleg dolgozni a megosztott gráfokon a gráf helyességének megőrzése mellett.*

Ez elég „pongyola” megfogalmazás, pontosítást igényel. Az, hogy *A*-t jobbnak tekinthetjük, mint *B*-t több mindentől függ. Elsősorban attól, hogy az *A* és *B* viselkedését milyen gráf és milyen felhasználók – pontosabban milyen szerkesztési tranzakciók – esetén hasonlítjuk össze. *Én általános célú workflow fejlesztőrendszer számára keresem a lehető legnagyobb párhuzamosságot engedő zároló algoritmust, vagyis azt, amely hosszú távon eredményezi a legnagyobb párhuzamosságot.* Mivel egy általános workflow fejlesztőrendszerben előre nem tudható hogy milyen struktúrájú gráfokat hány ember, milyen munkamegosztás szerint épít majd fel, ezért az algoritmusokat *minden elképzelhető gráf minden elképzelhető szerkesztési esetére* meg kellene vizsgálnom.

Ahogyan azt az 2.1 fejezetben és a 2-1. ábrán bemutattam, egy workflow életciklusa egy fejlesztési és egy futtatási fázisból áll. A szerkesztési fázis tovább bontható olyan szakaszokra, amelyek során valóban szerkesztik felhasználók a gráfot – ezeket hívom szerkesztési szakaszoknak – és olyanokra, amikor senki nem dolgozik a gráfon. A szerkesztési szakaszok és az azokat megszakító „szünetek” egymás után felváltva következnek és időbeli hosszúságuk előre nem ismert. Mivel a szünetek során nincs értelme párhuzamosságot vizsgálni, ezért a szerkesztési szakaszokra kell koncentrálni. Egy-egy szerkesztési szakasz független az őt megelőző szakasztól, a közös pont csak a gráf permanens része, *vagyis az a gráf, amelyet az előző szerkesztési szakasz végére a felhasználók létrehoztak. Egy V szerkesztési szakasz után következő W szerkesztési szakasz a V által előállított G gráfon indul.*

Egy szerkesztési szakasz során $n>0$ darab tranzakció kér indítást és ebből $m \leq n$ tranzakció fog valóban elindulni. Hogy pontosan mennyi és melyik, az a rendszer által használt zároló elbíráló algoritmus döntéseitől függ. Mivel előre nem ismerhető hogy a szakaszban mennyi és pontosan milyen tranzakcióindítási kérés lesz, ezért az algoritmus a döntést minden tranzakcióra külön-külön, az indulási kérés tartalma és a gráf aktuális állapota alapján hozza meg. *Egy ilyen döntésnél az számít, hogy*

- Mi a gráf permanens, vagyis a szerver oldalon mentett állapota.
- Milyen permanens komponensek vannak a már futó tranzakciók számára lefoglalva.
- Milyen komponenseket kér az éppen elbírálandó tranzakció zárolásra.

Vegyük észre hogy *a döntés független* egy sor más dologtól*:*

- A gráf temporális részétől, vagyis azoktól a még nem mentett komponensektől, amelyeket az épp futó tranzakciók hoztak létre.
- Attól, hogy mikor indultak a már futó tranzakciók.
- Attól, hogy mikor indul az most elbírálandó tranzakció.
- Attól, hogy milyen hosszan fognak futni a már futó tranzakciók.
- Attól, hogy milyen hosszan akar majd futni a most elbírálandó tranzakció.
- Attól, hogy engedés esetén pontosan milyen műveleteket fog végezni a tranzakció a gráfon.

- Attól, hogy engedés esetén a műveleteit milyen sorrendben és időzítéssel végzi majd a gráfon.
- Attól, hogy a tranzakció a módosításait végül menti (commit), vagy sem (abort).

Ezek következménye, hogy ha egy adott tranzakcióra akarjuk megállapítani, hogy egy adott zároló elbíráló algoritmus engedi-e a futását, ahhoz elegendő ismerni:

1. A gráf permanens részét.
2. A gráf permanens komponenseinek zárolási topológiáját.
3. Az indulást kért tranzakció zárolási kérését.

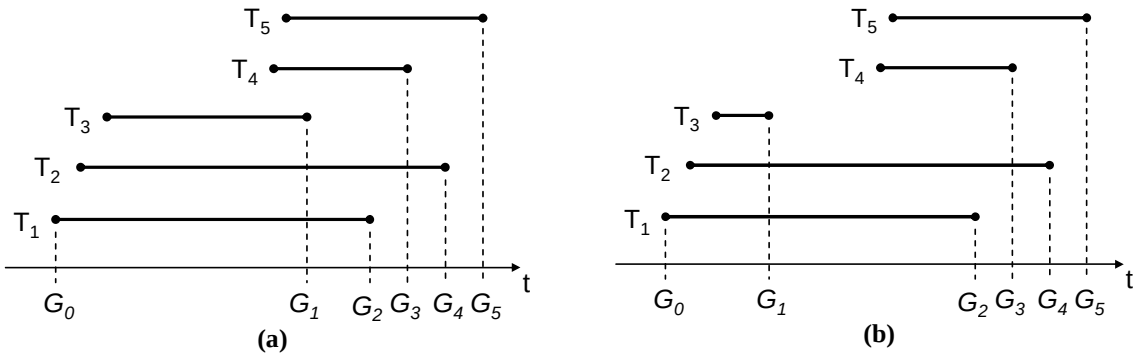
Párhuzamosság vizsgálatakor egy szerkesztési szakasz során egymás után indulást kért tranzakciók futásának lehetőségét vizsgáljuk. Ehhez a vizsgálathoz elegendő ismerni:

1. A szerkesztési szakasz elején létező gráf permanens részét.
2. A szerkesztési szakasz során indulást kért tranzakciók zárolási kéréseit.
3. Az indulási kérések beérkezési sorrendjét.

Egy szerkesztési szakasz a szerkesztendő gráf permanens részének tekintetében két féle lehet:

1. A szerkesztési szakasz tranzakcióira (T_1, T_2, T_3, ..., T_{n-1}, T_n) igaz, hogy $T_1 \parallel T_2 \parallel T_3 \parallel ... \parallel T_{n-1} \parallel T_n$. Ebben az esetben *a tranzakciók mindegyike egyidőben fut minden másikkal*, egyik tranzakció sem záródik le az utolsó tranzakció elindulása előtt. *Ekkor minden tranzakcióindulási kérés ugyanarra a permanens gráfrészre vonatkozik.* (Ld. 4-1. ábra (a) része.)

2. A szerkesztési szakasz tranzakcióira (T_1, T_2, T_3, ..., T_{n-1}, T_n) nem igaz hogy $T_1 \parallel T_2 \parallel T_3 \parallel ... \parallel T_{n-1} \parallel T_n$. Ebben az esetben $\exists T_i, T_j \mid T_i \rightarrow T_j$, vagyis legalább egy tranzakció (T_i) egy másik indulása előtt lezáródik. Ilyenkor a T_j tranzakció és az azután következő *tranzakciók indulási kérései már egy másik permanens gráfra vonatkoznak*, mint a T_i, és az azt megelőző tranzakciók indulási kérései. (Ld. 4-1. ábra (b) része)



4-1. ábra: Szerkesztési szakaszok két típusa.
Az (a) típus esetén egyetlen tranzakció sem záródik le az utolsó tranzakció indulása előtt és emiatt minden tranzakció indulási kérése a G_0 permanens gráfra vonatkozik. A (b) típusnál létezik az utolsó tranzakció elindulása előtt lezáródó tranzakció (T_3 a T_4 és T_5 előtt). Ilyenkor T_1, T_2 és T_3 kérései a G_0 gráfra, míg T_4 és T_5 kérései a G_1 gráfra vonatkoznak.

Egy szerkesztési eset párhuzamosság szempontjából történő vizsgálatának csak akkor van értelme, ha abban minden tranzakció ugyanarra a permanens gráfra vonatkozik. Csak ez esetben próbálnak ugyanis a tranzakciók egyazon gráfból komponenseket kapni, csak ekkor van értelme a kérdésnek, hogy „Egy *A* vagy egy *B* algoritmus enged-e több tranzakciót futni". Mostantól az ilyen szerkesztési esetek tranzakcióit *tranzakciósornak* hívom.

Def. 7: A $T^S = T_1, T_2, ..., T_n$ ***tranzakciósoron*** olyan tranzakciók sorozatát értem, melyek ugyanazon permanens gráfon kérnek időben egymásután zárolást, és igaz rájuk hogy $T_1 \| T_2 \| T_3 \| ... \| T_{n-1} \| T_n$. A tranzakciósorban a $T_1, T_2, ..., T_n$ tranzakciók a zárolási igényük benyújtásának időbeli sorrendjében szerepelnek, azaz $T_1 < T_2 < T_3 < ... < T_{n-1} < T_n$.

A fent 1. típusúnak hívott szerkesztési szakaszok tranzakcióira igaz a definíció, ezért *az olyan szerkesztési szakaszok, amelyekben egyetlen tranzakció sem záródik le az utolsó tranzakció elbírálása előtt, tranzakciósort határoznak meg.* A fent 2. típusúnak hívott szerkesztési szakaszok viszont csak *több szerkesztési esetre feldarabolva határoznak meg tranzakciósorokat, mivel bennük nem minden tranzakcióindítási kérés vonatkozik ugyanarra a permanens gráfra.* Az ilyen szerkesztési szakaszok tranzakciósorokra való feldarabolását úgy kell elvégezni, hogy a létrejövő szerkesztési esetek egyikében se legyen olyan tranzakció, amely valamely másik elbírálása előtt lezáródik. Ezt a következő algoritmussal lehet elérni:

1. A szerkesztési szakaszt „el kell vágni" azokon a helyeken, ahol korai tranzakciólezárások vannak, vagyis az olyan tranzakciólezárások után, amelyeket további tranzakciók indulási kérése követ. Egy ilyen vágás két darab, T^S_1 és T^S_2 tranzakciósort határoz meg.
2. Egy *t* időpillanatra vonatkozó vágás esetén egy *nem elvágott* T_a tranzakció T^S_1-be kerül, ha $t^v_a < t$ és T^S_2-be kerül, ha $t^i_a > t$. Az ilyen „nem elvágott" tranzakciók egyszerűen átkerülnek vagy T^S_1-be, vagy T^S_2-be. A szerkesztési szakasz vágása tehát első lépésben egy halmazátcsoportosítást foglal magában.
3. Egy *t* időpillanatra vonatkozó vágás esetén egy *elvágott* T_a tranzakció nem kerül sem T^S_1-be, sem T^S_2-be. Helyette
    a. egy olyan T_a' tranzakció kerül T^S_1-be, amelyre:
        i. $R_a' = R_a$ (ugyanazokat a komponenseket kéri zárolásra, mint az eredeti tranzakció)
        ii. A $t^v_a' = t + \Delta t$. (Rögtön a vágás után lezáródik. Δt a tranzakciók futási idejéhez képest valamilyen rövid időintervallumot jelöl.)
        iii. Nem okoz változást a gráf permanens részén, vagyis *rollback-el és nem commit-tal záródik le*.
        iv. Ha a vágás előtti szerkesztési szakasz egy T_b tranzakciójára igaz, hogy $T_a < T_b$ és a vágás miatt T_b vagy T_b' a T^S_1-be kerül, akkor $T_a' < T_b$ vagy $T_a' < T_b'$.
        v. Ha a vágás előtti szerkesztési szakasz egy T_b tranzakciójára igaz hogy $T_a > T_b$ és a vágás miatt T_b vagy T_b' a T^S_1-be kerül, akkor $T_a' > T_b$ vagy $T_a' > T_b'$.
    b. egy olyan T_a" tranzakció kerül T^S_2-be, amelyre:
        i. $R_a" = R_a$ (ugyanazokat a komponenseket kéri zárolásra, mint az eredeti tranzakció)
        ii. Ha a vágás előtti szerkesztési szakasz egy T_b tranzakciójára igaz hogy $T_a < T_b$ és a vágás miatt T_b vagy T_b" a T^S_2-be kerül, akkor $T_a" < T_b$ vagy $T_a" < T_b"$.

iii. Ha a vágás előtti szerkesztési szakasz egy T_b tranzakciójára igaz hogy T_a>T_b és a vágás miatt T_b vagy T_b" a T^S_2-be kerül, akkor T_a"> T_b vagy T_a"> T_b".

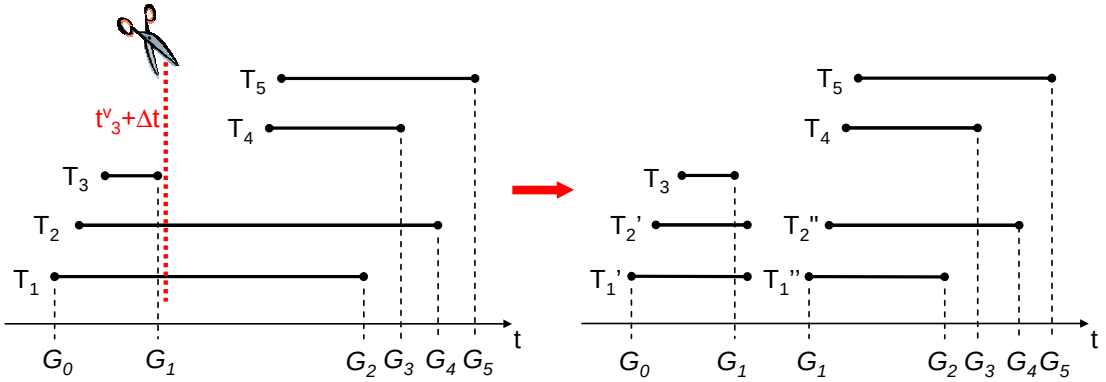
iv. Ha a vágás előtti szerkesztési szakasz egy T_b tranzakciójára igaz hogy t^v_a< t^v_b és a vágás miatt T_b vagy T_b" a T^S_2-be kerül, akkor t^v_a"< t^v_b vagy t^v_a"< t^v_b".

v. Ha a vágás előtti szerkesztési szakasz egy T_b tranzakciójára igaz hogy t^v_a> t^v_b és a vágás miatt T_b vagy T_b" a T^S_2-be kerül, akkor t^v_a"> t^v_b vagy t^v_a"> t^v_b".

vi. Ugyanazt a változást okozza a gráf permanens részén amit T_a.

A 4-1. ábra (b) részén látható szerkesztési szakaszt ezen szabály alapján a 4-2. ábrán látható módon lehet tranzakciósorokra bontani.

- Az 1. lépés értelmében annyiszor kell vágni ahány korán lezáródó tranzakció van, esetünkben egyszer. (T_3 záródik le korán.). A vágást T_3 lezárása után kell megtenni, valamilyen kis időintervallum eltéréssel ($t^v_3+\Delta t$ pillanat).
- A 2. lépés értelmében T_3 a T^S_1-be kerül, T_4 és T_5 a T^S_2-be kerül.
- A 3. lépés miatt T_1-ből és T_2-ből T^S_1 számára T_1' és T_1" jön létre, a T^S_2 számára pedig T_1" és T_2" jön létre.
- 3/a/i és 3/b/i miatt R_1=R_1'=R_1", illetve R_2=R_2'=R_2". 3/a/ii miatt T_1' és T_1" is lezáródik közvetlenül T_3 után, de 3/a/iii miatt nem eredményeznek új gráf verziót, vagyis az első tranzakciósor G_0 gráfból G_1 gráfot állítja elő. 3/a/iv lépés miatt T^S_1-ben T_1'<T_3 és T_1'<T_2' (és emiatt persze T_1'<T_3).
- 3/b/ii és 3/b/iii miatt T^S_2-ben T_1"< T_2", T_2"<T_4 és T_4<T_5.
- 3/b/iv és 3/b/v miatt T^S_2-ben t^v_1"< t^v_4, t^v_4< t^v_2" és t^v_2"< t^v_5.
- 3/b/vi miatt T^S_2 a G_1 gráfból G_5 gráfot állítja elő



4-2. ábra: Egy korai tranzakciólezárást tartalmazó szerkesztési szakasz tranzakcióinak beosztása egy T^S_1=T_1', T_2', T3 és egy T^S_2=T_1", T_2", T_4, T_5 tranzakciósorrá.

Ezzel az algoritmussal minden „korán lezáródó" tranzakciót tartalmazó szerkesztési szakaszból tranzakciósorok hozhatók létre. Mivel célom az összes szerkesztési szakaszon vizsgálni a zároló algoritmusok hatékonyságát, ezzel az eljárással a feladatot a zároló algoritmusok tranzakciósorokon való elemzésére redukáltam. Amennyiben az összes tranzakciósoron elvégzem a vizsgálatot, abból megadható a korán lezáródó tranzakciót tartalmazó szerkesztési szakaszokon (fent 2. típus) elérhető hatékonyság, és eleve közvetlenül adódik a tranzakciósort jelentő szerkesztési szakaszokon (fent 1. típus) elérhető hatékonyság.

4.3. Zárolási kéréseket elbíráló algoritmusok modellezése

A vizsgálathoz a zárolási kéréseket elbíráló algoritmusokat egy függvénnyel fogom modellezni. A függvény fejléce minden elbíráló algoritmus számára azonos, míg a függvény implementációja algoritmusról algoritmusra más és más. A függvényt „tranzakciósor elbíráló függvény”-nek fogom hívni:

Tranzakciósor elbíráló függvény bemenő adatai:

- T^S: A vizsgált tranzakciósor tranzakciói. ($T^S = T_1, T_2, ..., T_n$). R_i a T_i tranzakció indulási kérése. $T_a < T_b$.
- *G(V, E)* – A tranzakciósor tranzakciói által szerkeszteni kívánt gráf. A G gráfot meghatározó komponensek a workflow-t a múltban szerkesztő, már lezárt tranzakciók által lettek létrehozva (permanens komponensek). A gráfot nem szerkeszti a tranzakciósoron kívüli tranzakció, ezért egyetlen komponense sem zárolt.

Tranzakciósor elbíráló függvény kimenő adatai:

- *G(V, E)* = $G_1 \cup G_2 \cup ... \cup G_n \cup G^U$ ami a *G* gráf T^S tranzakciósor tranzakcióinak elbírálása után kapott, zárolásokat is tartalmazó változata. A G_i részgráf a T_i tranzakció számára lefoglalt gráf komponenseket tartalmazza. G^U a G gráf nem zárolt komponenseit tartalmazza.
  - Mivel a konzisztenciaőrzés miatt egy gráf komponens egyszerre maximum csak egy tranzakció számára lehet lefoglalva, ezért $G_i \cap G_j = \emptyset, \forall 1 \le i, j \le n, i \ne j$ esetén.
  - Mivel a gráf egy komponense vagy le van foglalva vagy nincs, ezért $G_i \cap G^U = \emptyset, \forall 1 \le i \le n$ esetén.
- *[0, 1]*^{n} bináris vektor, amelynek *i.* eleme 0 $\Leftrightarrow$ ha az algoritmus a tranzakciósor *i.* tranzakciójának elindulását nem engedte. (Ilyenkor $G_i = \emptyset$) A vektor *i.* eleme 1 $\Leftrightarrow$ ha az algoritmus T_i elindulását engedte. (Ilyenkor $G_{i+1} \ne \emptyset$) Bár ez a vektor egyértelműen meghatározható függvény első visszatérési értékének ismeretében, azért kerül külön bevezetésre, mert kifejezőértéke segít a későbbi vizsgálatok egyszerűsítésében.

A függvény segítségével vizsgálható, hogy egy adott tranzakciósor egy adott gráfon való elbírálása esetén melyik algoritmus mennyi és pontosan melyik tranzakciókat engedi, és melyeket tiltja. Mivel a fent bevezetett tranzakciósor koncepció szerint nincs tranzakcióindítási kérést megelőző tranzakciólezárás, ezért a „tranzakcióindítást” modellező függvény önmagában elegendő a párhuzamosság vizsgálatához, nincs szükség a tranzakciólezárást „modellező” további függvényre.

4.3.1. Metrikák

Egy tranzakciósorban szereplő indulási kérések elbírálására alkalmas függvény megadása után formálisan is definiálhatók azok a tulajdonságok, amelyekkel zárolási kéréseket elbíráló algoritmusok közötti különbségek objektív módon kimondhatók.

Def. 8: Egy A zárolási kéréseket elbíráló algoritmus ***jobb, mint*** egy B zárolási kéréseket elbíráló algoritmus egy *G* gráf és azon indulást kérő T^S tranzakciósorra, ha az elbíráló függvénye *több tranzakció számára enged futást,* mint a B algoritmus tranzakciósor elbíráló függvénye. Jelölése: A↖B.

A „jobb mint" reláció tranzitív, de csak rögzített T^S és *G*-re. Vagyis miután kiválasztottuk T^S-t és *G*-t, és egy A, B és C algoritmusokra igaz hogy A↖B, B↖C, akkor A↖C is igaz lesz. Ettől függetlenül még létezhet olyan *F* gráf és T^P tranzakciósor, melyre C↖A.

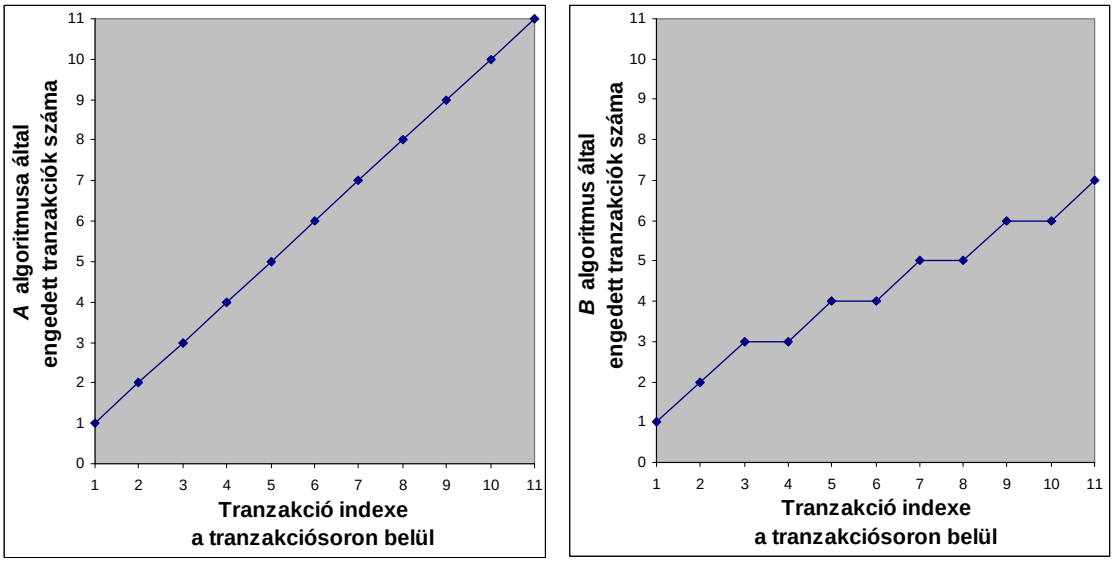
Def. 9: Egy A zárolási kéréseket elbíráló algoritmus ***ugyanolyan jó***, mint egy B zárolási kéréseket elbíráló algoritmus, ha egy rögzített T^S tranzakciósor és rögzített G gráf esetén *ugyanannyi tranzakció számára enged futást* a gráfon, mint B. Jelölése A↤B.

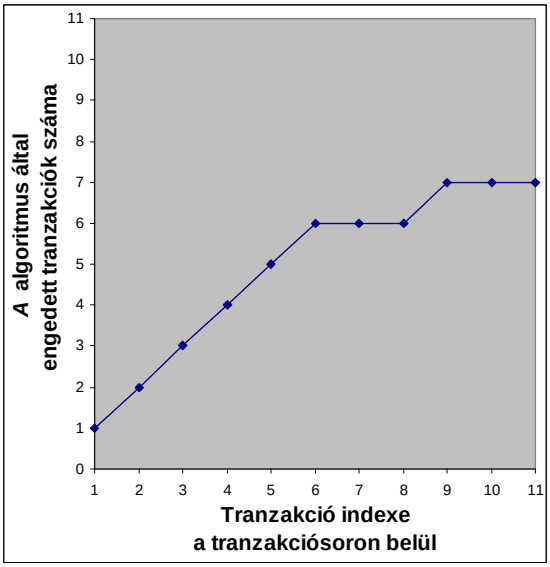
Def. 8 és Def. 9 használják a „tranzakció számára engedi a futást" szófordulatot. Mit is jelent ez pontosan? Ez azt jelenti, hogy az előre rögzített tranzakciósor elemein végighaladva azokat egyesével elbírálja, egy-egy tranzakcióra második visszatérési értékként egy 0-át vagy 1-et adva a vektorhoz. Amelyik tranzakcióra 1-et adott azt engedi elindulni, amelyikre 0-át adott azt nem. *Mivel tranzakciósorokról van szó, ezért az 1-essel elbírált tranzakciók legalább a tranzakciósor utolsó tranzakciójának az elbírálása utánig futnak.* Más szóval, nem fejeződik be tranzakció addig, amíg minden tranzakcióindítási kérés nincs elbírálva. *Az utolsó tranzakció elbírálása után lesz az az időpillanat, amikor az adott algoritmussal az adott G gráf és T^S tranzakciósor esetén a legtöbb tranzakció dolgozik egyidejűleg a gráfon. Tulajdonképpen ez az a pont, ami számunkra érdekes, ami alapján azt mondjuk két algoritmusra, hogy ugyanakkora konkurenciát engednek vagy sem, vagyis hogy egyenlők, vagy egyikük jobb, mint a másik.* Ld. 4-3. ábra.

A↤B esetén az A által visszaadott $[0, 1]^n_A$ eredményvektorban ugyanannyi 1-es van, mint a B által visszaadott $[0, 1]^n_B$ eredményvektorban. A két vektorban az 1-esek pozíciója viszont nem feltétlenül ugyanaz! A vektorokban az 1-esek pozíciójáról Def. 8 és Def. 9 semmit sem mond, így azt sem hogy A vektorában mindenhol 1-es van-e azokban a pozíciókban ahol B vektorában 1-es van, vagyis hogy A ugyanazokat a tranzakciókat engedi-e futni amelyeket B. A „jobb" és „ugyanolyan jó" értelmezések tehát csak az 1-esek számával foglalkoznak, azok elhelyezkedésével nem.

Megjegyzés:

A jósági relációk jeleit (↖, ↤) éppen a függvényekre való utalással választottam. A jobb algoritmus függvénye magasabbra jut, ezért felfelé mutató a nyíl. Az ugyanolyan jó algoritmusok függvénye ugyanolyan magasra jut, ezért vízszintes a nyíl.





- *A* függvényének maximuma nagyobb, mint *B* függvényének maximuma ezért A↖B.
- *B* függvényének maximuma ugyanannyi, mint *C* függvényének maximuma ezért B↚C

4-3. ábra: Egyazon *G* és T^S esetén egy fiktív "A", "B" és "C" zárolási kéréseket elbíráló algoritmus által engedett tranzakciók számának függvényszerű ábrázolása. Mivel a párhuzamosság vizsgálata során tranzakciólezárással nem kell foglalkozni az ilyen függvények mindig monoton növekvőek.

Def. 10: Egy A zárolási kéréseket elbíráló algoritmus **m véges egész értékkel jobb, mint** egy B zárolási kéréseket elbíráló algoritmus egy *G* gráf és azon indulást kérő T^S tranzakciósorra, ha az elbíráló függvénye *m db-bal több tranzakció számára enged futást*, mint a B algoritmus tranzakciósor elbíráló függvénye.

A definíció nyilvánvalóan feltételezi, hogy T^S legalább *m* db tranzakciót tartalmaz. (Ha pontosan *m*-et, akkor A mindet engedi, B mindet tiltja.)

Def. 11: Jelölje $\bar{a} = (a_1, a_2, ..., a_n)$ és $\bar{b} = (b_1, b_2, ..., b_n)$ az A és B zárolási kéréseket elbíráló algoritmusok által egy *G* gráf és T^S tranzakciósor esetén visszaadott eredményvektorokat. Az A algoritmus ***ugyanolyan igazságos,*** mint a B algoritmus, jelöléssel **A⇆B**, ha A↤B és $\sum_{h_a \in H_A} h_a = \sum_{h_b \in H_B} h_b$, ahol $H_A = \{h_a \mid \bar{a}[h_a] = 1\}$ és $H_B = \{h_b \mid \bar{b}[h_b] = 1\}$. (Azaz a H_A halmaz pontosan azokat az indexeket tartalmazza, amelyeken az $\bar{a}$-ban 1-es áll, és a H_B halmaz azokat indexeket tartalmazza, ahol a $\bar{b}$-ban 1-es áll.)

Def. 12: Jelölje $\vec{a} = (a_1, a_2, ..., a_n)$ és $\bar{b} = (b_1, b_2, ..., b_n)$ az A és B zárolási kéréseket elbíráló algoritmusok által egy *G* gráf és T^S tranzakciósor esetén visszaadott eredményvektorokat. Az A algoritmus ***igazságosabb, mint*** a B algoritmus, azaz **A↾ B**, ha A⇆B és $\sum_{h_a \in H_A} h_a < \sum_{h_b \in H_B} h_b$, ahol $H_A = \{h_a \mid \bar{a}[h_a] = 1\}$ és $H_B = \{h_b \mid \bar{b}[h_b] = 1\}$. (Azaz a H_A halmaz pontosan azokat az indexeket tartalmazza, amelyeken az $\bar{a}$-ban 1-es áll, és a H_B halmaz azokat az indexeket tartalmazza, ahol a $\bar{b}$-ban 1-es áll.)

Def. **11**. szerint tehát akkor ugyanolyan igazságos két algoritmus egy adott szerkesztési esetre, ha egyrészt ugyanannyi tranzakciót engednek, továbbá ha ezen felül *az engedett tranzakciók sorszámainak összege is azonos*. Ez tulajdonképpen annyit jelent, hogy van, amelyik felhasználót az egyik, és van, amelyiket a másik algoritmus részesíti előnyben, de összességében ugyanannyi előny és ugyanannyi hátrányt adnak az adott szerkesztési szakaszban részt vevő felhasználóknak. Az igazságosságot demonstrálja a következő két példa:

1. Példa:

	1-esek pozíciója
A eredményvektora: [1 0 0 1 0 0 1 0]	1 4 7
B eredményvektora: [1 0 1 0 0 0 0 1]	1 3 8

H_A elemei: 1, 4, 7; $\sum h_a = 12$

H_B elemei: 1, 3, 8; $\sum h_b = 12$

$12 = 12 \Rightarrow$ A ⇆ B

2. Példa

	1-esek pozíciója
A eredményvektora: [1 1 1 1 0 0]	1 2 3 4
B eredményvektora: [1 1 1 0 1 0]	1 2 3 5

H_A elemei: 1, 2, 3, 4; $\sum h_a = 10$

H_B elemei: 1, 2, 3, 5; $\sum h_b = 11$

$10 < 11 \Rightarrow$ A↾ B

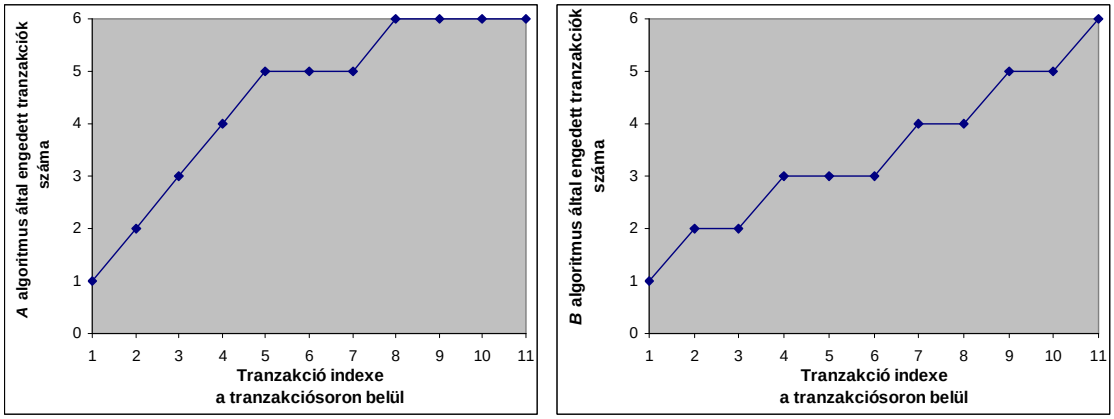
Igazságosság szempontjából csak ugyanolyan jó algoritmusok hasonlíthatók össze. Az igazságosság tulajdonképpen arról ad információt, hogy két olyan algoritmus közül, amelyik ugyanannyi tranzakciót enged futni egy adott helyzetben, melyik preferálja azon tranzakciók futását, amelyek hamarabb kérték az indulást. Amelyik tranzakciók hamarabb kérték az indulást,

azok alacsonyabb sorszámmal szerepelnek a tranzakciósorban és emiatt az elbírálásuk kimenetét jelentő értékek is alacsonyabb pozícióban vannak az eredményvektorban. (kisebb h indexszel).

Miért csak ugyanolyan jó algoritmusok esetén értelmezem az igazságossági összehasonlítást? Mert különben egy állandóan [1, 0, 0, …, 0] vektort visszaadó algoritmus lenne a lehető legigazságosabb, holott ez valójában mindig csak egyetlen tranzakciónak engedi meg a futást a gráfon, vagyis egyáltalán nem is eredményez kollaboratív szerkesztést.

Az igazságossági relációjáról elég triviálisan belátható hogy tranzitív (azaz megtartja a rendezést): Ha egy rögzített G gráf és T^S tranzakciósorra $A \upharpoonleft B$ és $B \upharpoonleft C \Rightarrow A \upharpoonleft C$. Ugyanúgy, mint a "jobb mint" reláció esetén, a tranzitivitás itt is csak rögzített gráf és tranzakciósor pár esetén igaz. Attól, hogy T^S és G esetén $A \upharpoonleft B$ és $B \upharpoonleft C$, még létezhet olyan F gráf és T^P tranzakciósor melyre $C \upharpoonleft A$.

A 4-3. ábrán látható függvények nem csak a jósági reláció megállapításához, hanem az igazságossági összehasonlításhoz is segítséget nyújtanak. Amíg a függvények végállapota a jóságról ad információt, addig a függvények tranziens része az igazságosságra vonatkozik. Azonos végértékkel rendelkező függvények közül ugyanis az az igazságosabb, amelyik hamarabb éri el a maximumát, mivel az preferálja inkább a korábban indulást kért tranzakciókat. Másképp úgy is tekinthető, hogy az az igazságosabb, amelyiknél a függvény alatti rész nagyobb területet foglal. Egy ilyen példa látható a 4-4. ábrán.



4-4. ábra: Igazságossági reláció demonstrálása függvénnyel.
Az A és B algoritmusok ugyanolyan jók. A kettő közül A az igazságosabb.
Ez a függvényekből onnan látszik, hogy A függvénye éri el hamarabb a maximumát.

Megjegyzések:

1. Az igazságossági relációk jelét ($\upharpoonleft$, $\leftrightarrows$) a függvényekre való utalással választottam. Az igazságosabb algoritmus függvénye meredekebben emelkedik, erre utal a függőlegesbe forduló nyíl. Az ugyanolyan igazságos algoritmusok függvénye azonos magasságba jut, és azonos mértékben emelkedik. Erre utalnak a vízszintes nyilak.

2. A bevezetett igazságossági összehasonlításra szolgáló metrikák várakozó sor nélküli rendszerekre vonatkoznak. Várakozó sort kezelő rendszerekre olyan értékek lehetnének metrikaként vizsgálhatók, mint például
   - Melyik rendszer késleltet darabszámra kevesebb tranzakciót egy tranzakciósoron belül.
   - Melyik rendszer eredményez *összességében* kevesebb késleltetést egy tranzakciósorban.

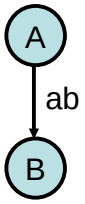
Mivel munkámban várakozó sor nélküli rendszerekkel foglalkozom, ezért maradok a fent ismertetett igazságossági értelmezésnél.

4.4. A1 és A2 algoritmusok összehasonlítása

4.4.1. Mondható olyan G, T^S pár amelyre A2 *jobb, mint* A1? Igen.

Pl.:

G:



$T^S = T_1, T_2$
$R_1=\{A\}, R_2=\{B\}$

Ennél A1 [1, 0], A2 pedig [1, 1] vektort ad vissza. A2 vektorában több az 1-es érték, vagyis A2 jobb. Mivel erre a szerkesztési esetre a két algoritmus ugyanolyan jó, ezért az igazságosság nem vizsgálható.

4.4.2. Mondható olyan G, T^S pár amelynél A1 *jobb, mint* A2? Nem.

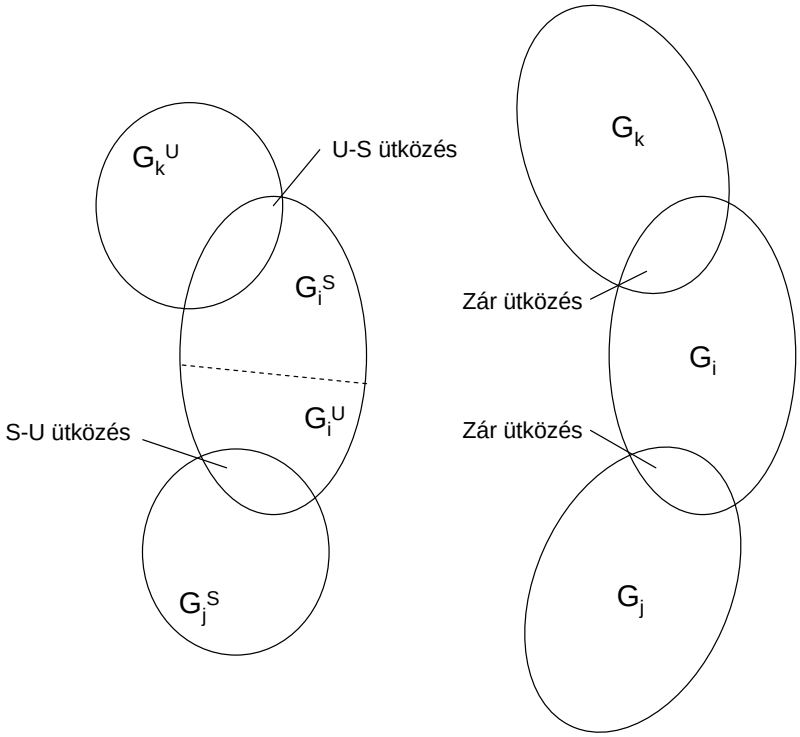
Állítás: Nincs olyan G, T^S pár amelynél A1 ↖ A2.

Bizonyítás:

Lemma: Ha A2 tiltja egy T_i tranzakció elindulását egy tetszőleges G gráfon, akkor azt A1 is tiltja.

Biz (Lemma): Ha A2 tiltja T_i elindulását, akkor T_i a G gráfon U-S és S-U ütközést is okoz egyidejűleg a már futó T_j és T_k tranzakciókkal. (T_j és T_k lehet akár ugyanaz). Emiatt tudjuk, hogy a T_i számára lefoglalandó G_i részgráf U zárral lefoglalt része (G_i^u) átfedésben van T_j részgráf S részével (G_j^S-el). Ld. 4-5. ábra.

Mivel A1 nem különböztet meg U és S lock-okat, ezért ebben a topológiában nála $G_i^U \cup G_i^S = G_i$, és $G_j^U \cup G_j^S = G_j$, továbbá G_i és G_j ugyanazokkal az elemekkel átfedésben vannak, mint G_i^u és G_j^S az A2 esetén. Az átfedés miatt A1 tiltja T_i elindítását.



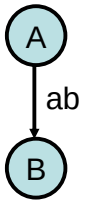
4-5. ábra: Ha A2 tiltja egy T_i tranzakció elindulását (bal oldal), akkor az A1 algoritmus is tilt (jobb oldal).

A Lemma következménye, hogy T^S egy T_i tranzakciójának elbírálásakor, ha A2 0-át ad vissza, akkor A1 is 0-át ad. Ebből adódóan A1 nem eredményezhet több 1-est semmilyen T^S tranzakciósor elbírálásakor, mint A2, vagyis nincs olyan G és T^S amelynél A1 jobb eredményt adna, mint A2.

4.4.3. Mondható olyan G, T^S pár amelynél A1 *egyenlő* A2? Igen

Pl.:

G gráf:



$T^S = T_1, T_2$
$R_1=\{A\}, R_2=\{A, B\}$

Ennél A1 és A2 is [0, 0] vektort ad vissza, vagyis egyenlők, sőt mivel az egyesek pozíciója is megegyezik a vektorokban ezért A1⇆A2.

Állítás: Minden olyan G és T^S esetén amelyre A1 és A2 ugyanolyan jó, ott ugyanolyan igazságos is.

Bizonyítás:
A1 és A2 csak akkor adhat egyenlő eredményvektorokat, ha mindkettőben ugyanannyi a 0-ás érték. Tudjuk a 4.4.2 fejezet bizonyításából, hogy ha egy adott G és T_i esetén A2 0-át ad akkor A1 is 0-át ad. Vagyis ebből következik, hogy egyenlőség esetén az eredményvektorjaikban pontosan ugyanott kell, hogy legyenek a 0-ás értékek, emiatt mindkettőben ugyanott lesznek az 1-es értékek is, vagyis a két algoritmus ugyanolyan igazságos eredményt ad.

4.4.4. Értelmezés – Az A1 és A2 algoritmusok közötti különbségről

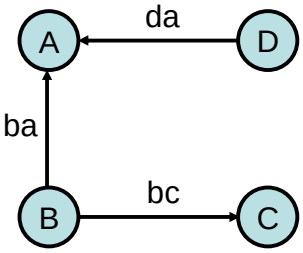
Összefoglalva elmondható hogy A2 sosem rosszabb, mint A1, vagyis az értelmezési tartomány minden pontján ugyanolyan jó, vagy jobb, mint A1. Ahol ugyanolyan jó, ott ugyanolyan igazságos is.

Vagyis semmilyen kollaboratív rendszerben nincs értelme az A2 algoritmus ellenében az A1 algoritmust beépíteni. Egy A2 algoritmust használó kollaboratív rendszer bármilyen szituációban tud legalább olyan jól teljesíteni, mint egy A1-et használó rendszer.

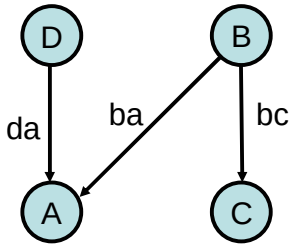
4.5. A2 és A3 algoritmusok összehasonlítása

4.5.1. Mondható olyan G, T^S pár amelyre A3 *jobb, mint* A2? Igen.

G gráf:



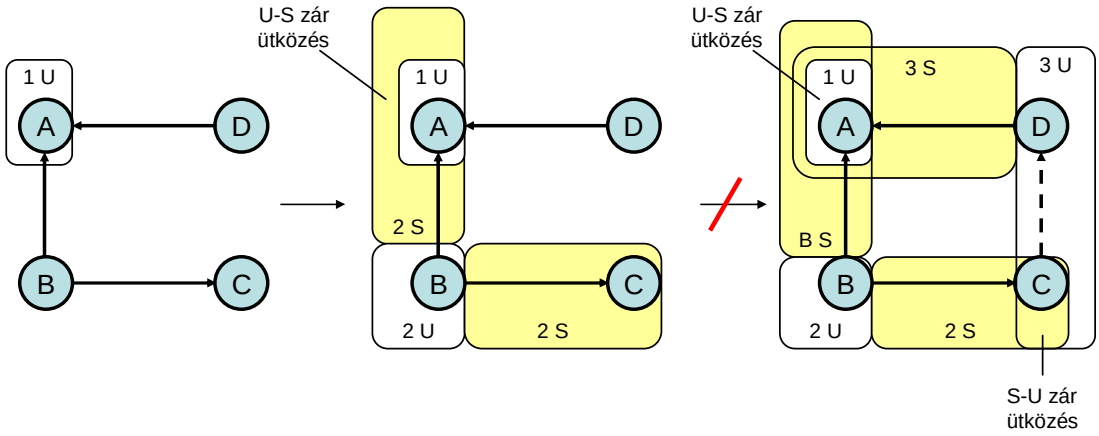
felhasználóbarátabb ábrázolásban:



$T^S = T_1, T_2, T_3$
$R_1=\{A\}, R_2=\{B\}, R_3=\{C, D\}$

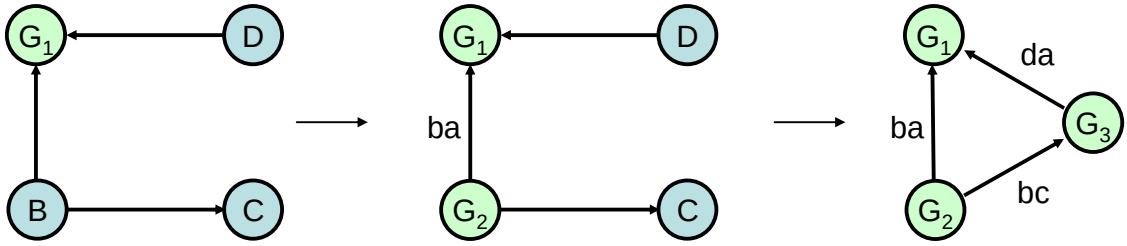
4-6. ábra: Egy olyan T^S tranzakciósor és *G* gráf, amelyre A3 ↖A2.

Az A2 algoritmus ezen *G* gráf és T^S tranzakciósor esetén T_1-et és T_2-t engedi futni, míg T_3-at nem, vagyis függvénye a [1, 1, 0] vektort fogja eredményül adni. Ennek oka, hogy míg T_1 és T_2 elindulása nem okoz egyszerre U-S és S-U ütközést, addig T_3 elindulása ezt teszi és emiatt indulása tiltott. Az A2 algoritmus által a gráfra kerülő zárolások láthatók a 4-7. ábrán.



4-7. ábra: A2 algoritmus zárjai egy A3 ↖A2 esetre.

A A3 algoritmus engedi mind T_1 mind T_2, mind T_3 elindulását, vagyis az [1, 1, 1] vektort adja eredményül. Ennek oka, hogy egyik tranzakció indulásakor sem alkotnak kört a már lefoglalt részgráfok és az újonnan lefoglalandó részgráf. T_3 elindulásának pillanatában mutatja a részgráfok kapcsolatát a 4-8. ábra, ezen látható, hogy ekkor sem (és emiatt korábban sem) létezik kör, vagyis az A3 algoritmus szerint T_1, T_2 és T_3 is indítható.



4-8. ábra: A3 algoritmus szomszédossági gráfja a 4-6. ábrán látható szerkesztési esetre.

Megjegyzés:

> A 4-7. ábra jobb oldali részén az is látható, hogy T_3 az egyetlen tranzakció, amelyik élt tudna létrehozni a gráfban (hiszen csak ő birtokolna legalább két csomópontot), viszont az ő él létrehozási kérése sem tudna kört eredményezni a gráfban, bármilyen irányba is mutasson az új, C és D csúcsok között futó él. Vagyis T_3 futása feleslegesen tiltott. Az algoritmusokat ilyen szempontból majd később, az 5.2.1 fejezetben vizsgálom.

A következőekben azok az általános tranzakciósor és gráf minták kerülnek meghatározásra, amelyekre A3 jobb, mint A2.

4.5.2. Amikor A3 nagyobb párhuzamosságot enged, mint A2

Az előbb mutattam egy példát olyan gráf-tranzakciósor párra, melyre A3 ↖ A2 egyetlen értékkel. Most azt a G, T^S mintát fogom meghatározni, amely esetén egyáltalán előfordulhat, hogy A3 ↖ A2.

Tudjuk, hogy A3 ↖ A2 esetén több olyan tranzakció van T^S-ben amelyet A3 enged és A2 tilt, mint amelyet A2 enged és A3 tilt. Nevezzük azt a tranzakciót, amelyik A3 javára billenti ezt az egyensúlyt T_i-nek. T_i-ről tudjuk, hogy *indulása nem ad kört a szomszédossági gráfban, viszont*

a már futó tranzakciókkal egyszerre S-U és U-S ütközést is okoz. Ez alapján T_i-re a következő kritériumrendszer mondható ki (ld. még 4-9. ábra):

(1) A T_i számára U lock-kal lefoglalandó részgráf (G_i) legalább egy csomópontjába út kell, hogy vezessen egy már U lock-kal lefoglalt részgráf (G_j) valamely csomópontjából: $\exists v \in G_i, u \in G_j, u \notin G_i : [u,v] \in E$

és

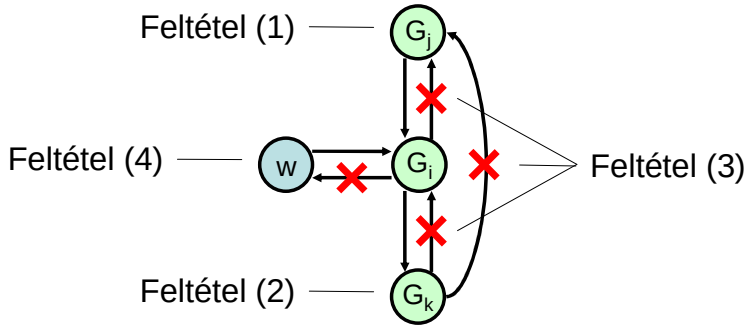
(2) A T_i számára U lock-kal lefoglalandó részgráfból (G_i) út kell, hogy vezessen egy már U lock-kal lefoglalt részgráf (G_k) valamely csomópontjába: $\exists v \in G_i, u \in G_k, u \notin G_i : [v,u] \in E$

és

(3) Nem létezik körút a T_i számára U lock-kal lefoglalandó részgráf (G_i) és semelyik már U lock-kal lefoglalt részgráf között: $\neg \exists u,v \in G_i, w,x \in G_m : [u,w] \in E, [x,v] \in E$

és

(4) Nem létezik körút a T_i számára U lock-kal lefoglalandó részgráf (G_i) és semelyik zárolatlan csomópont (W) között: $\neg \exists u,v \in G_i, w \in V, w \notin G_i, w \notin G_j : [u,w] \in E, [w,v] \in E$



4-9. ábra: Feltételrendszer, amely megadja, hogy milyen gráfstruktúra és tranzakcióesetén lehet A3↖A2.

Állítás: A fenti, 4 pontból álló kritériumhalmaz *elégséges feltételrendszer* olyan T_i tranzakcióra, melyet az A2 algoritmus tilt, az A3 algoritmus viszont enged.

Biz: A bizonyításnak be kell látnia, hogy ha a fenti kritériumok egy T_i tranzakcióra igazak, akkor a T_i tranzakciót A2 tiltja, A3 viszont engedi.

Lemma 1: A fenti kritériumok fennállása esetén A2 tiltja a T_i tranzakciót.

Biz (Lemma 1): Az (1) kritérium miatt az A2 algoritmus szerint a T_j tranzakció a G_j részgráfot a G_i részgráffal összekötő út élein és csomópontjain, ezen felül a G_i részgráf legalább egy csomópontján is S lock-kal rendelkezik. G_i ezen csomópontjára T_i elindulásakor U lock kerül, vagyis T_i indulása S-U ütközést okoz. A (2) kritérium miatt az A2 algoritmus a T_i tranzakció indulásakor a G_i részgráfot a G_k részgráffal összekötő út éleire és csomópontjaira, továbbá a G_k részgráf legalább egy csomópontjára S lock-ot tesz. Mivel a G_k ezen csomópontja már U lock-kal lefoglalt, ezért itt T_i indulása U-S ütközést okoz. Vagyis T_i elindulása mind S-U, mind U-S ütközést okoz, ezért indulása A2 szerint tiltott.

Lemma 2: A fenti kritériumok fennállása esetén A3 engedi a T_i tranzakciót.

Biz (Lemma 2): A fenti kritériumrendszer (3) pontjában szereplő G_i, G_m részgráfok és a (4) pontjában szereplő zárolatlan csomópontok éppen az A3 algoritmus szomszédossági gráfjának csomópontjainak felelnek meg. Mivel ez a két feltétel kiköti, hogy ezen komponensek között semmilyen topológiában nincs körút, ezért a szomszédossági gráfban sincs kör, vagyis a T_i tranzakció indulását engedi A3.

Állítás: A fenti, 4 pontból álló kritériumhalmaz *szükséges feltételrendszer* olyan T_i tranzakcióra, melyet az A2 algoritmus tilt, az A3 algoritmus viszont enged.

Biz: A bizonyítás megmutatja, hogy ha T_i tranzakciót az A2 algoritmus tilt az A3 algoritmus viszont enged, akkor a fenti kritériumhalmaz *minden eleme külön-külön igaz*.

Mivel az A2 algoritmus tiltja T_i elindulását, ezért annak S-U és U-S ütközést is kell okoznia.

- S-U ütközést csak úgy okozhat, hogy valamelyik számára U lock-kal lefoglalandó csomópont S lock-kal már egy másik tranzakció birtokában van. Ekkor ezen másik tranzakció által birtokolt valamely csomópontból él kell hogy fusson a T_i számára allokálandó G_i részgráfba, vagyis az (1) feltétel igaz kell hogy legyen.
- U-S ütközést csak úgy okozhat, hogy valamelyik számára S lock-kal lefoglalandó csomópont U lock-kal már egy másik tranzakció birtokában van. Ekkor ezen másik tranzakció által birtokolt részgráfba él kell hogy fusson a T_i számára allokálandó G_i részgráfból, vagyis a (2) feltétel igaz kell hogy legyen.

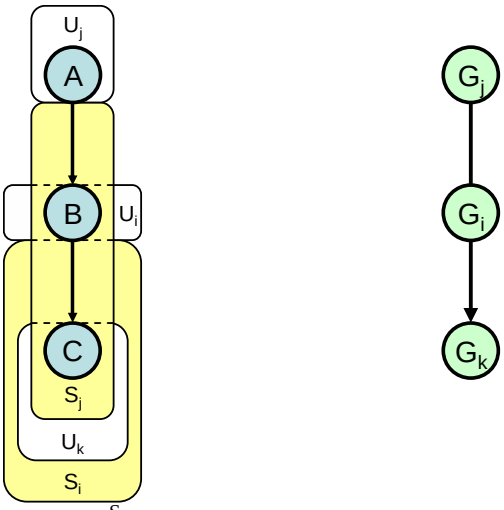
Mivel az A3 algoritmus engedi T_i elindulását, ezért a szomszédossági gráfban nem lehet kör. A fenti kritériumrendszer (3) pontjában szereplő G_i, G_m részgráfok és a (4) pontjában szereplő zárolatlan csomópontok éppen az A3 algoritmus szomszédossági gráfjának csomópontjainak felelnek meg. Ha abban nincs kör, akkor a G_i és valamely már lefoglalt részgráf, illetve a G_i részgráf és valamely zárolatlan csomópont között sem lehet. Vagyis a (3) és (4) kritériumok igazak kell hogy legyenek.

A fenti kritériumrendszer tehát szükséges és elégséges feltételeket ad olyan tranzakcióra melyet A2 tilt, A3 pedig enged. A fenti kritériumrendszer (3) pontjának egyik következménye, hogy az (1) és (2) pontokban T_j és T_k néven említett tranzakciók nem lehetnek azonosak vagyis $G_j \neq G_k$. Ha ugyanis azonosak lennének, akkor $G_j = G_k$ és emiatt a G_i és G_j részgráfok között körút adódik, aminek következtében A3 sem engedi T_i elindulását. Mivel $T_j<T_i$, $T_k<T_i$ és $T_i \parallel T_j \parallel T_k$, ezért *egy tranzakciósorban a fenti kritériumhalmaznak megfelelő T_i-nél legalább két másik tranzakció hamarabb indul*.

Egy tranzakciósor sokféleképpen tartalmazhatja a fenti kritériumoknak megfelelő T_i, T_j, T_k tranzakciókat. Vegyük észre, hogy T_j és T_k tranzakciók sorrendjére nincs megkötés, egymáshoz képest bármilyen sorrendben indulhatnak. Arra sincs megkötés, hogy T^S hány további tranzakciót tartalmaz T_j, T_k, és T_i mellett. Ha ezen három tranzakción kívül továbbiakat is tartalmaz T^S, akkor azok akár T_j, T_k, és T_i közé is ékelődhetnek *mindaddig, amíg ezen három elindulását nem gátolják*. Ha ezen további tranzakciók nem billentik vissza a tranzakciósorban T_i által A3 javára billentett egyensúlyt, akkor a teljes tranzakciósorra nézve A3 ↖ A2.

A minimális olyan implementáció, amely megfelel a T_i–vel szemben támasztott kritériumoknak, a 4-10. ábrán látható. *„Minimális" alatt azt értem, hogy a G gráf a legkevesebb*

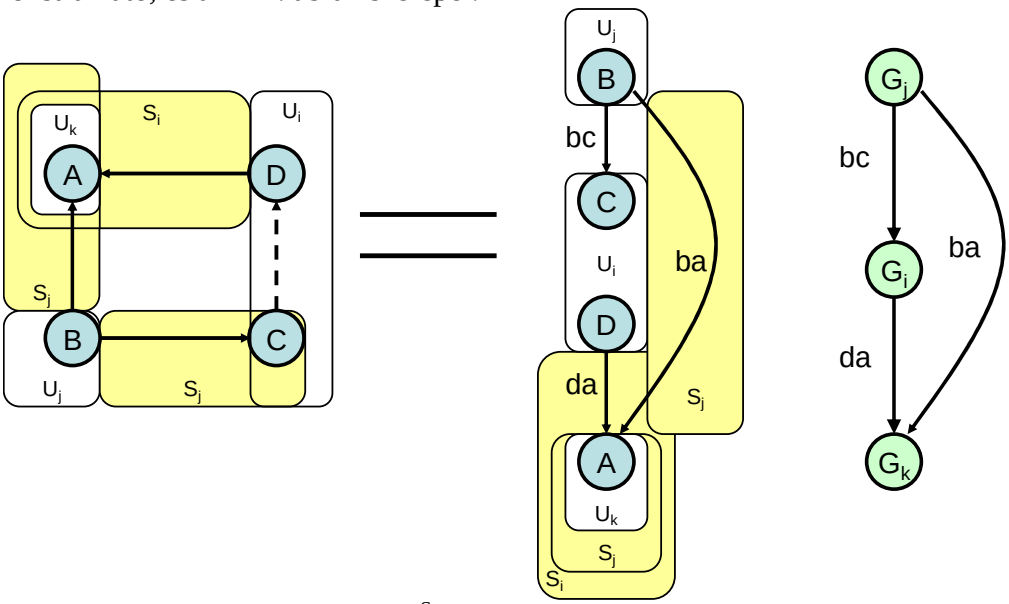
csomópontot, T^S tranzakciósor pedig a legkevesebb tranzakciót tartalmazza. A tranzakciósor elbírálásakor A2 függvénye az [1, 1, 0], míg A3 függvénye az [1, 1, 1] vektort adja vissza.



$T^S=T_j, T_k, T_i$
$R_j=\{A\}, R_k=\{C\}, R_i=\{B\}$

4-10. ábra: Egy minimális *G*, T^S pár, amelyre A3 ↖ A2.
Bal oldalon a gráf látható A2 zárjaival, jobb oldalon pedig az A3 által készített szomszédossági gráf.

Az 4.5.1 részben példaként felhozott tranzakciósor T_3 tranzakciója is a fenti kritériumrendszernek felel meg. Egy, a 4-10. ábrán látható ábrázolásmóddal ez könnyen demonstrálható, és a 4-11. ábrán szerepel.



$T^S=T_j, T_k, T_i$
$R_j=\{B\}, R_k=\{A\}, R_i=\{C, D\}$

4-11. ábra: Az 4.5.1részben példaként felhozott gráf általános ábrázolásmódban (középen).
Jobb oldalon pedig a belőle A3 által készített szomszédossági gráf.

A fent definiált kritériumhalmaz ismeretében olyan összetett gráf és tranzakciósor pár is készíthető, amelyre A3↖A2 *tetszőlegesen nagy, véges m értékkel*. A lehető legegyszerűbb ilyen példa a 4-10 ábrán látható gráf *m* esetű többszörözésével állítható elő. A gráf és a rajta A3 számára A2-vel szemben *m* előnyt adó tranzakciósor alább, a 4-12. ábrán látható.

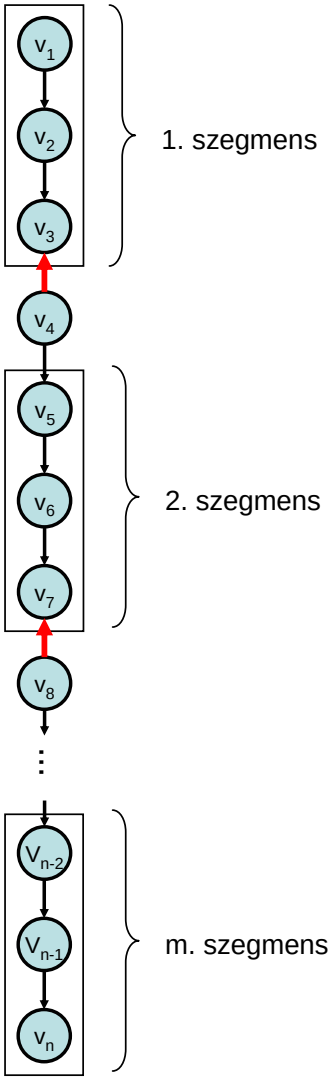
G: Csomópontjainak száma: $n = (m*3)+(m-1)= 4m-1$

Topológiája: *m* db szegmensből áll, mindegyik szegmens tartalmazza a 4-10. ábrán szereplő mintát. A vastagított éleknek nincs kitüntetett szerepe, csak azt hangsúlyozzák, hogy nem egy elágazás nélküli „pipeline” gráfról van szó.

$T^S = T_1, T_2, T_3, T_4, T_5, T_6, ..., T_{3m-2}, T_{3m-1}, T_{3m} =$

$= \{V_1\}, \{V_3\}, \{V_2\}, \{V_5\}, \{V_7\}, \{V_6\}, ..., \{V_{n-2}\}, \{V_n\}, \{V_{n-1}\}$

A2 a T^S minden harmadik elemét tiltja, míg A3 minden tranzakciót enged. Emiatt a 3*m* hosszúságú tranzakciósoron A3 éppen *m*-el lesz jobb, mint A2.

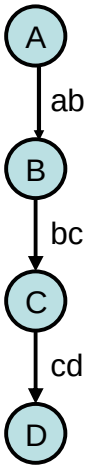


4-12. ábra: Példa olyan gráfra és tranzakciósorra, amelyre A3 tetszőlegesen nagy, véges *m* értékkel jobb, mint A2.

4.5.3. Mondható olyan G, T^S pár amelyre A2 *jobb, mint* A3? Igen.

Egy ilyen G és T^S példa:

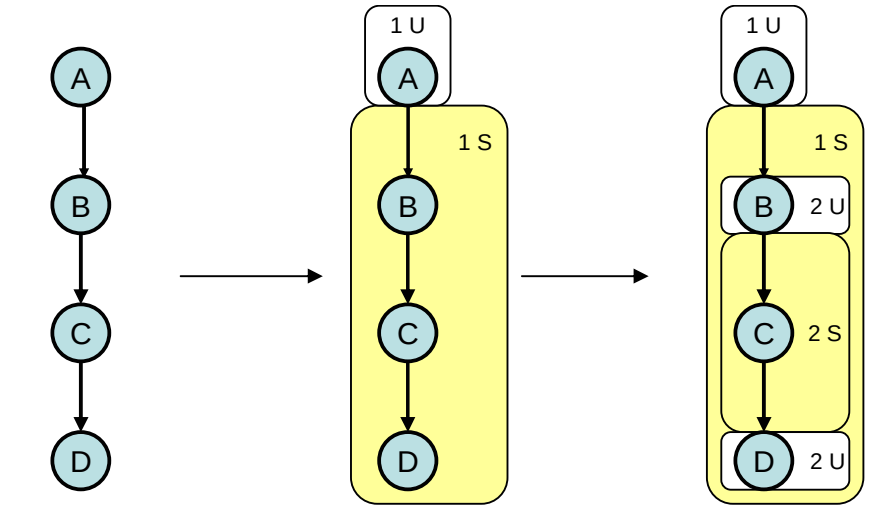
G:



$T^S=T_1, T_2$
$R_1=\{A\}, R_2=\{B, D\}$

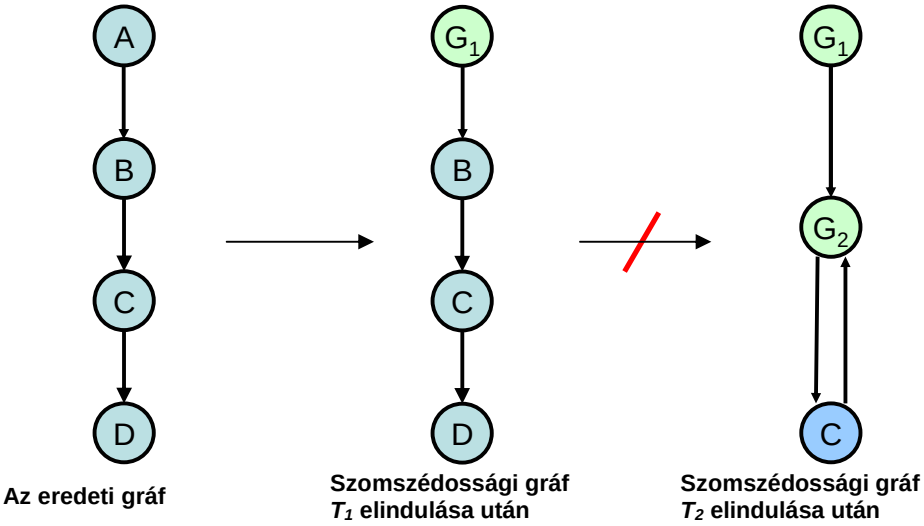
4-13. ábra: Egy olyan T^S tranzakciósor és G gráf, amelyre A2 ↖A3.

A2 algoritmussal elbírálva G-t és T^S-t a 4-14 ábrán látható zárolási mintát kapjuk. Mivel sem T_1, sem T_2 indulása nem okoz egyszerre U-S és S-U ütközést, ezért A2 engedi mindkét tranzakció elindulását. Mivel T_2 editora a saját $U \cup S$ részgráfján képes felismerni és védekezni kör ellen, ezért a T_2 által *esetlegesen* létrehozott *db* él nem kerülhet be a G_2 részgráfba, majd onnan a teljes gráfba.



4-14. ábra: A2 által elbírált tranzakciók és zárolt komponenseik

A3 algoritmussal elbírálva T^S-t a 4-15. ábrán szereplő szomszédossági gráfokat kapjuk, és ahogy látható, T_2 elindulása a szomszédossági gráfban kört hozna létre, és emiatt A3 nem engedi a tranzakciót:



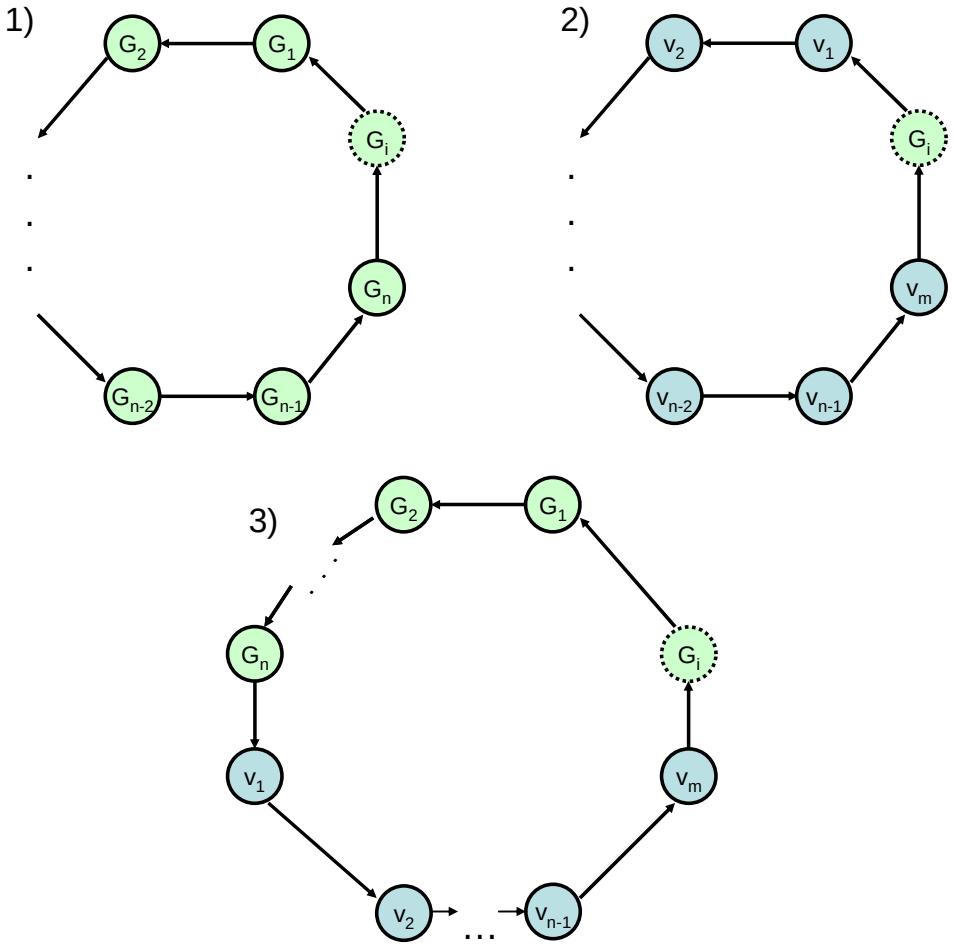
4-15. ábra: A3 által elbírált tranzakciók és a szomszédossági gráf

A példaként felhozott G és T^S esetén tehát A2 függvénye [1, 1], míg A3 függvénye [1, 0] vektort adja vissza, vagyis A2 ↖ A3.

4.5.4. Amikor A2 nagyobb párhuzamosságot enged, mint A3

Az előbb mutattam egy példát olyan gráf-tranzakciósor párra, melyre A2 ↖ A3 egyetlen értékkel. Most azt a G, T^S mintát fogom meghatározni, amely esetén egyáltalán előfordulhat, hogy A2 ↖ A3. Tudjuk, hogy A2 ↖ A3 esetén több olyan tranzakció van T^S-ben amelyet A2 enged és A3 tilt, mint amelyet A3 enged és A2 tilt. Nevezzük azt a tranzakciót, amelyik A2 javára billenti ezt az egyensúlyt T_i-nek. Mivel A3 tiltja T_i elindulását, ezért tudjuk, hogy van kör a szomszédossági gráfban. Ez három féle módon alakulhat ki (ld. még 4-16. ábra.):

1. A kört a T_i tranzakció számára zárolandó G_i részgráf és a T^S-en belül G_i előtt indult tranzakciók zárolt részgráfjai (G_1, G_2, ..., G_n) alkotják. (4-16. ábra 1. része)
2. A kört zárolatlan csomópontok (V_1, V_2, ..., V_m) és a T_i számára zárolandó G_i részgráf alkotják. (4-16. ábra 2. része)
3. A kört a T^S-en belül G_i előtt indult tranzakciók zárolt részgráfjai, zárolatlan csomópontok (V_1, V_2, ..., V_m) és a T_i számára zárolandó G_i részgráf alkotják. (Tetszőleges sorrendben) (4-16. ábra 3. része)



4-16. ábra: Kör kialakulásának módjai egy szomszédossági gráfban:
1) csak zárolt részgráfokból,
2) zárolatlan csomópontokból és egyetlen részgráfból,
3) zárolt részgráfokból és zárolatlan csomópontokból vegyesen.

1. eset vizsgálata:
   Ha ebben a szituációban T_i–t az A2 algoritmussal bíráljuk el, akkor a G_n-et birtokló T_n tranzakcióról tudható, hogy S lock-kal rendelkezik G_i legalább egy csomópontján mivel belőle él fut G_i-be. Az újonnan induló T_i viszont S lock-ot kapna G_1 legalább egy csomópontján, mivel belőle él fut G_1-be. (Szélső esetben n=1 és G_n=G_1, az U-S és S-U ütközés ekkor is fennáll.) Vagyis T_i kérése mind S-U, mind U-S ütközést okoz és emiatt *A2 nem engedi T_i elindulását*.

2. eset vizsgálata:
   Ha ebben a szituációban T_i–t az A2 algoritmussal bíráljuk el, akkor T_i a G_i részgráfon U lock-ot kap, a gráf többi, G_i-hez kapcsolódó részén S lock-ot, és ezzel sem U-S, sem S-U ütközést nem okoz, vagyis *A2 engedi T_i elindulását*.

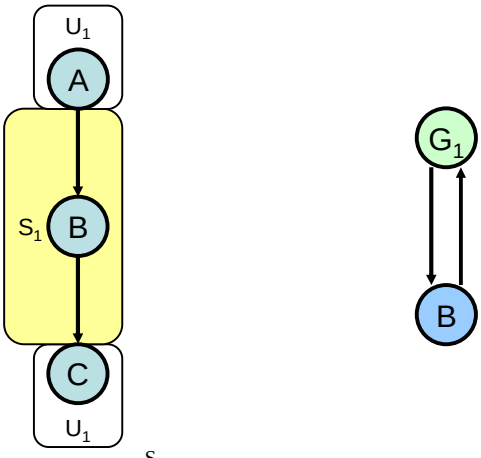
3. eset vizsgálata:
   Ha ebben a szituációban T_i–t az A2 algoritmussal bíráljuk el, akkor a G_n-et birtokló T_n tranzakcióról tudható, hogy S lock-kal rendelkezik a G_n és G_i közötti csomópontokon, és

G_i legalább egy csomópontján. Ez függetlenül igaz attól, hogy mi a G_1-G_n részgráfok és V_1-V_m csúcsok sorrendje. Az újonnan induló T_i viszont S lock-ot kapna G_1 legalább egy csomópontján. Vagyis T_i indulása, mint S-U, mind U-S ütközést okoz és emiatt *A2 nem engedi T_i elindulását.*

Vagyis egyedül a fent tárgyalt második esetben fordulhat elő az, hogy egy tranzakciót A2 enged, és A3 tilt. Ez az eset a tranzakcióra nézve a fenti bizonyítás következtében az alábbi *szükséges és elégséges kritériumrendszert támasztja*:

(1) A tranzakció a gráf legalább kettő, nem közvetlenül egymás után következő, de egymástól függő csomópontján kér zárolást[11]: $\exists U,V \in R_i : uv \notin E, vu \notin E, [uv] \in E$ *és*
(2) Ezen két csomópont között kizárólag zárolatlan csomópontok vannak és ezen csomópontokon a tranzakció nem kér zárolást: $\forall W : \exists [uw] \in E, \exists [wv] \in E \Rightarrow W \notin R_i$

Egy minimális, ennek a kritériumrendszernek megfelelő gráf és tranzakciósor a 4-17. ábrán látható. Ugyanúgy mint korábban, minimális alatt azt értem, hogy a gráf a legkevesebb csomópontot, a tranzakciósor pedig a legkevesebb tranzakciót tartalmazza – esetünkben csupán egyet. Ezt a tranzakciót A2 engedi, A3 pedig nem.



4-17. ábra: Egy olyan minimális gráf-tranzakciósor pár, amelyre A2 ↖ A3. Bal oldalon a gráf látható A2 zárjaival, jobb oldalon pedig ugyanerre az esetre az A3 által készített szomszédossági gráf.

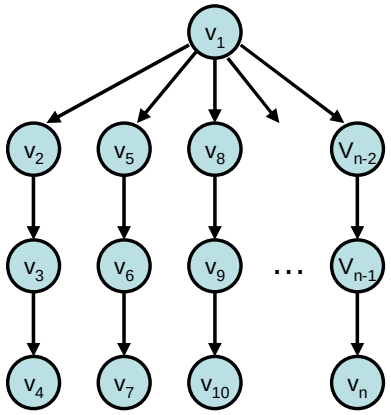
Ugyan A2 engedi a T_1 tranzakció elindulását, minden ez után jövő, T_1 által U lock-kal körül fogott elemekkel dolgozni akaró tranzakciót A2 tiltani fog. (Vagyis a példán szereplő esetben egy T_2 = {B} tranzakciót.) Az ilyen tranzakciók ugyanis mind U zárolást kérnek T_1 által S lock-kal birtokolt komponensen (példánkban B-n), és S zárolást igényelnek a T_1 által U lock-kal birtokolt részen (példánkban C-n), vagyis egyszerre U-S és S-U ütközést is produkálnak.

[11] (u, v) jelentése: u és v csomópontok között futó él; [u, v] jelentése: u és v csomópontok között futó irányított út. Az út egy vagy több él egymásutánját jelenti.

Bármilyen példát néznénk is ez mindig igaz lesz, vagyis ehhez a tranzakciósorhoz nem adható további olyan tranzakció melyet A2 enged és A3 tilt.

Mivel egy gráfban tetszőleges olyan legalább 3 csomópont hosszúságú szakasz lehet, ami a fent definiált kritériumnak megfelel, ezért *elképzelhető olyan G, T^S pár, melyre A2 ↖ A3 tetszőlegesen nagy véges m értékkel*. Egy ilyen G, T^S párra ad példát a 4-18. ábra. A példaként felhozott gráf nem a lehető legkevesebb csomópontot tartalmazó változat, viszont összefüggő.

G: Csomópontjainak száma: $n = 3m + 1$.
Topológiája: *m* db ágból áll, mindegyik ág tartalmazza a 4-17. ábrán szereplő mintát.
$T^S = T_1, T_2, T_3, ..., T_m = \{V_2, V_4\}, \{V_5, V_7\}, \{V_8, V_{10}\}, ..., \{V_{n-2}, V_n\}$
A3 a T^S minden elemét tiltja, míg A2 mindet engedi, ezért az *m* hosszúságú tranzakciósoron A2 éppen *m*-el lesz jobb, mint A3.

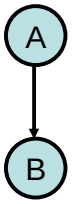


4-18. ábra: Példa olyan gráfra és tranzakciósorra, amelyre A2 tetszőlegesen nagy, véges *m* értékkel jobb, mint A3.

4.5.5. Mondható olyan G, T^S pár, amelynél A2 *egyenlő* A3? Igen

Az 4.5.2 részben bemutattam azt a mintát, amelyre illeszkedő T_i tranzakció képes egy tranzakciósorban A3 javára A2 ellenében 1 tranzakciónyi előnyt adni. Az 4.5.4 fejezetben azt a tranzakciómintát mutattam be, ami viszont A2 javára ad előnyt. Ha 0 db eltérést eredményező tranzakció van egy tranzakciósoron belül, *vagy* ha az A2 és A3 javára előnyt biztosító tranzakciók száma azonos egy tranzakciósoron belül, akkor arra a tranzakciósorra A2 ↔A3. Ez alapján könnyen készíthető ilyen, A2 és A3 által is azonos párhuzamosságot megvalósító tranzakciósor:

G:



$T^S = T_1, T_2$
$R_1 = \{A\}, R_2 = \{A, B\}$

Ennél A2 és A3 is [1, 1] vektort ad vissza, vagyis egyenlők, sőt mivel az egyesek pozíciója is megegyezik a vektorokban, ezért A2⇆A3.

Állítás: Olyan G és T^S esetén, amelyre A2 és A3 egyenlők, nem biztos, hogy ugyanolyan igazságosak.

Bizonyítás: Fentebb láttuk, hogy ha egy G és T^S párra A2 ↔A3, akkor a következő esetek valamelyike állhat fenn:

1. T^S valamely T_i tranzakcióját A2 akkor és csak akkor engedi, ha azt A3 is engedi (vagyis nincs T^S-ben valamelyik algoritmus javára előnyt biztosító tranzakció.). Ilyen esetben nyilván A2 ⇆ A3.
2. Azonos számban szerepel A2 és A3 javára előnyt biztosító tranzakció T^S-en belül. Ebben az esetben előfordulhat, hogy az eltérések éppen kiegyenlítik egymást, viszont lehet olyan tranzakciósorrend is, amelyre A2 ↼ A3, vagy amelyre A3 ↼ A2. Az 4.5.1 és 4.5.3 részekben szereplő példákban olyan G és T^S szerepelnek, melyek esetén A2 és A3 között pontosan 1 tranzakciónyi eltérés van. Ha ezeket a példákat „egybegyúrjuk”, akkor ezekből olyan gráf és tranzakciósor pár kapható, amelyekre A2↔A3 és A2 ↼ A3 vagy A3 ↼ A2. Ha például az 4.5.1 fejezetben szereplő példa gráfját és tranzakciósorát most G_1 és $T^S{}_1$-nek hívjuk, míg az 4.5.3 fejezetben szereplő példa gráfját és tranzakciósorát G_2 és $T^S{}_2$-nek hívjuk, akkor az integrált gráf lehet $G_1 \cup G_2$ (vagyis egy nem összefüggő gráf), az integrált tranzakciósor pedig A2 ↼ A3 esetén $T^S{}_2+T^S{}_1$, míg A3 ↼ A2 esetén $T^S{}_1+T^S{}_2$. (A „+” operátor egy $T^S{}_1$ és $T^S{}_2$ tranzakciósorra egy olyan új $T^S{}_3$ tranzakciósort eredményez, amelyben az $T^S{}_1$ és $T^S{}_2$ tranzakciósorok tranzakciói egymás után másolva kerülnek be úgy, hogy a $T^S{}_1$ legutoljára lezáródó és a $T^S{}_2$ legkorábban induló tranzakciója egyidőben fusson.)

4.5.6. Értelmezés – Az A2 és A3 algoritmusok közötti különbségről

Mivel az 4.5.2 részben megmutattam, hogy sokféle (és pontosan milyen) tranzakciósor esetén lehet A3 ↖ A2, az 4.5.4 részben pedig azt, hogy mikor A2 ↖ A3, ezért a két algoritmus közötti kapcsolatot nem lehet olyan egyszerűen elintézni, mint az A1 és A2 algoritmusok kapcsolatát. *Sem A2, sem A3 nem egyértelműen jobb, mint a másik, nem lehet egyiket vagy másikat minden esetben maximális párhuzamosságot eredményező „győztesnek” kinevezni és azt mondani, hogy egy kollaboratív rendszerbe csak azt érdemes a másik ellenében beépíteni. Az, hogy néha A2, néha A3 a jobb, és ez azt jelenti, hogy különböző kollaboratív szituációkban van hogy az egyik, van hogy a másik eredményez nagyobb párhuzamosságot. Hogy éppen melyik, az attól függ, hogy éppen milyen gráfról és annak milyen minta szerinti szerkesztéséről van szó.*

4.6. A1 és A3 algoritmusok összehasonlítása

A1 és A2, illetve A2 és A3 összehasonlítási eredményének ismeretében az A1-A3 összehasonlítás a korábbiaknál jóval egyszerűbb módon végezhető el. Mivel tudjuk, hogy A2 mindenhol legalább olyan jó, mint A1, továbbá ismertek azok az esetek, amelyekben az A3 algoritmus jobb, vagy ugyanolyan jó, mint A2, ezért biztosra vehető hogy ezekben az esetekben A3 van olyan jó, mint A1. A kérdés csupán az, hogy azokra a tranzakciókra, amelyekre A2 ↖ A3, vajon mi az A1 és A3 algoritmusok kapcsolata?

Az 4.5.4 fejezetben megállapítottam az A2 által engedett, A3 által tiltott tranzakciókról, hogy a gráf legalább kettő, nem közvetlenül egymás után következő, de egymástól függő csomópontján kérnek zárolást, úgy, hogy ezen zárolásra kért csomópontok között kizárólag zárolatlan csomópontok vannak, és azokon nem kérnek zárolást. *Egy ilyen tranzakciót A1 csak akkor tiltja, ha a tranzakció számára lefoglalandó részgráfból egy már lefoglalt részgráfba út fut. Ha ez a feltétel nem teljesül, akkor A1 engedi a tranzakciót. Vagyis vannak olyan szerkesztési esetek melyekben A1 nagyobb párhuzamosságot tesz lehetővé, mint A3.* Például a 4-18. ábrán látható gráf és tranzakciósor kizárólag ilyen, A1 által engedett, A3 által tiltott tranzakciókat tartalmaz és emiatt erre az esetre A1 *m* értékkel lesz jobb, mint A3. Ez alapján persze könnyen lehetne kevesebb tranzakciót tartalmazó olyan tranzakciósort mondani amelyre A1 ↖ A3.

4.7. Gyakorlati példák 1.

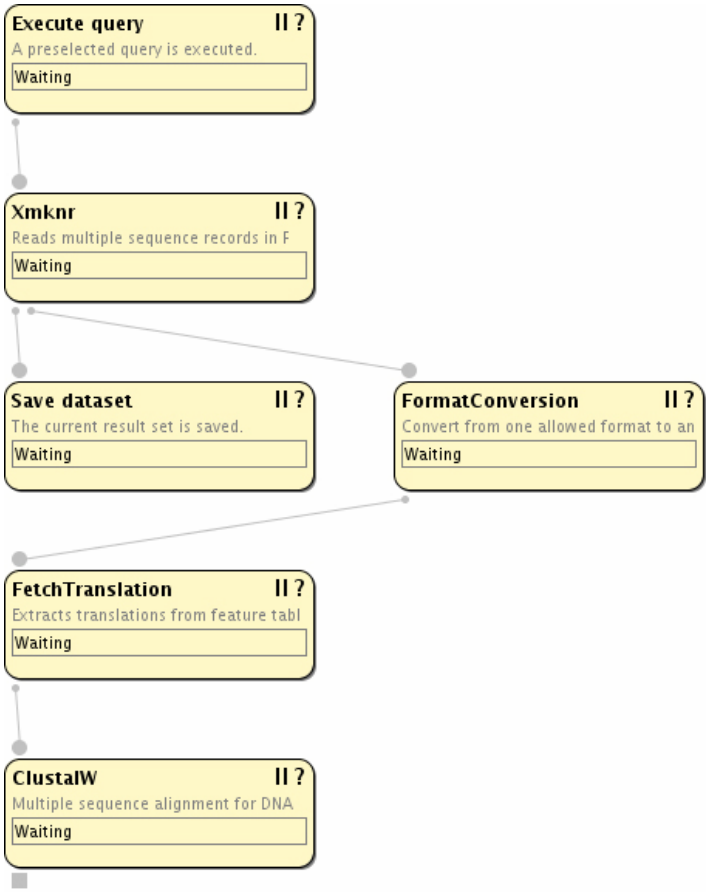
A következőekben röviden ismertetek néhány való életből vett workflow-t és demonstrálom rajtuk az A2 és A3 algoritmusok közötti különbséget. Minden példaként felhozott workflow-ra megadok egy-egy olyan tranzakciósort, amelyre vagy A2↖A3, vagy A3↖A2, vagy A2↔A3, viszont nem ugyanolyan igazságos eredményt ad a két algoritmus. *Mindegyik példaként felhozott gráfra a megadott tranzakciósorok mellett számos további olyan tranzakciósor lehet adható, amelyre az A3 és az A2 algoritmusok nem azonos eredményt adnak.*

A célom a példákkal demonstrálni, hogy valóságos workflow-k kollaboratív rendszerrel való szerkesztésekor közel sem mindegy, hogy a használt szerkesztőrendszer az A2, vagy az A3 algoritmust használja gráf felosztáshoz. A példákban használt workflow-k aránylag összetettek, ezért ilyen méretű gráfok kollaboratív szerkesztése során valóban előnyös lehet a kollaboratív szerkesztés, több felhasználó munkájának hatékony összeillesztése.

A példákat mind Grid vagy High Performance Computing területről választottam, de lehetett volna számtalan más háttérből is. Viszont ügyeltem arra, hogy a workflow-k különböző fejlesztő környezetekkel készüljenek, és arra is, hogy a felhozott tranzakciósor példák egymástól különbözzenek. A példák ismertetése szándékosan rövid, mellőzi a mély analízist, csakis a felhasználók munkájának párhuzamosíthatóságára fókuszál. A rövid példák így lehetővé teszik a workflow alkalmazások több féle, teljesen eltérő tudományterületeken való felhasználási lehetőségének bemutatását is.

Példa 1: Liliopsida Protein Alignment workflow (Lushbough et al, 2008):

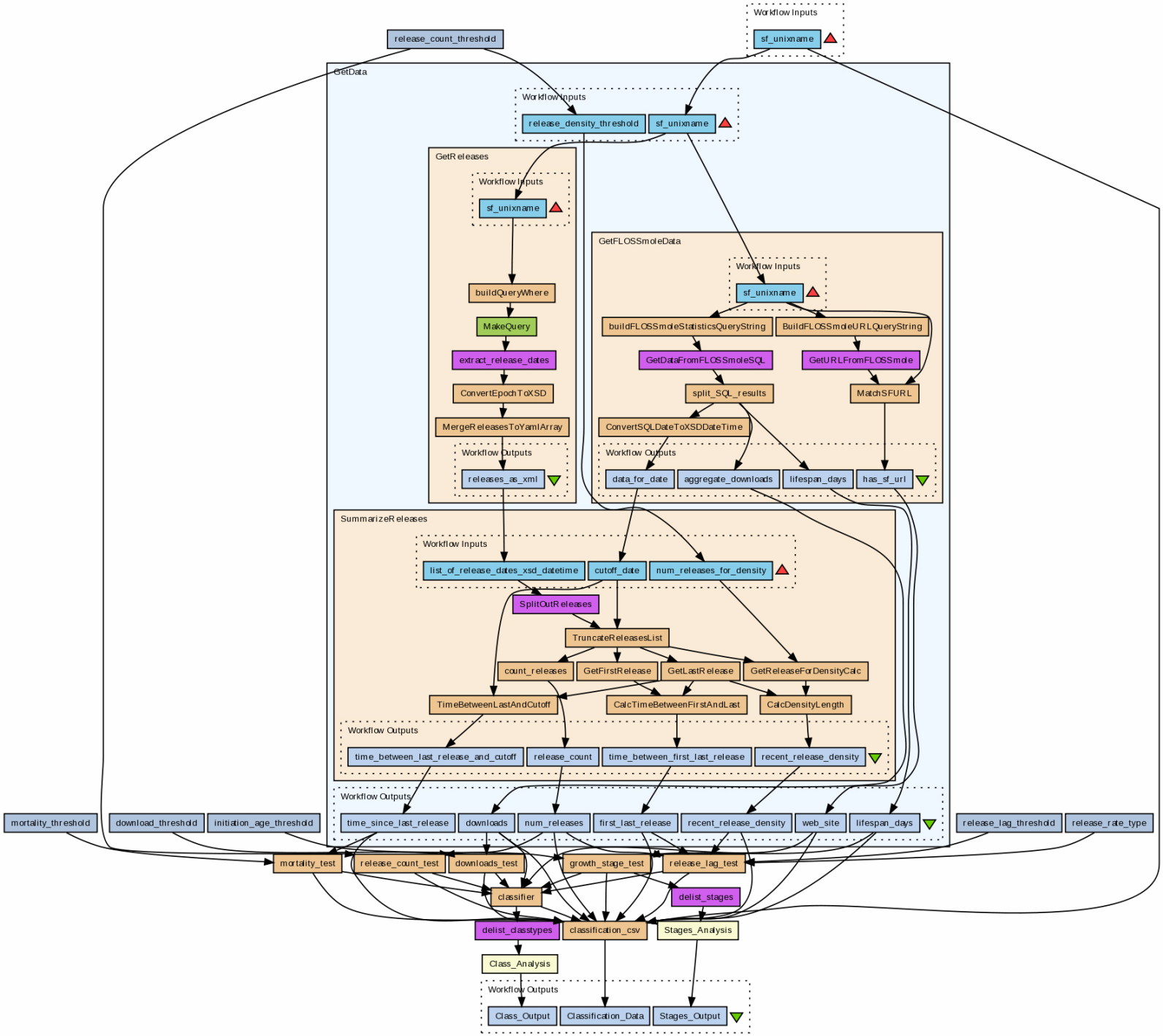
- Készítette: Carol Lushbough
- Workflow környezet: BioExtract
- Célja: A workflow az NCBI nukleotidból kinyeri a Liliopsida Chloreoplast Petb gént, eltávolítja a duplikátumokat, majd az eredményt menti a BioExtract szerverre. Ezek az eredmények azután áttranszformálásra kerülnek GenBank formátumba.
- Egy olyan tranzakciósor, amelyre A2↖A3:
  $T^S = T_1, T_2$
  R_1={Xmknr, Save dataset, FetchTranslation, ClustalW}
  R_2={Execute query}
- A2-vel elbírálva: [1, 1]; A3-al elbírálva: [0, 1]



4-19. ábra: Liliopsida Protein Alignment workflow BioExtract környezetben

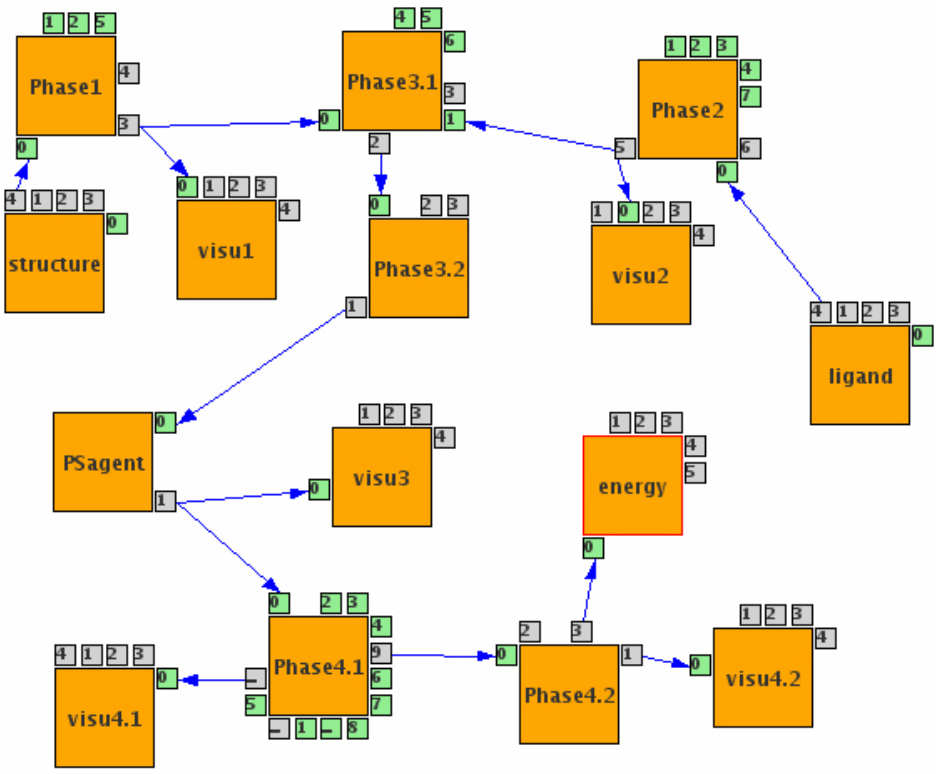
Példa 2: Success Abandonment Classification workflow (Howison et al, 2008):

- Készítette: Andrea Wiggins
- Workflow környezet: Taverna
- Célja: A FLOSSmole és Notre Dame SourceForge tárhelyeken regisztrált projektekről tölt le adatokat, melyekből azután különböző statisztikákat állít elő, úgy mint megjelenések, letöltések, életciklus. A statisztikák alapján azután a projektek kategorizálásra kerülnek.
- Egy olyan tranzakciósor amelyre A3↖A2:
  $T^S = T_1, T_2, T_3$;
  R_1={CalcDensityLength}
  R_2={GetLastRelease}
  R_3={GetReleaseForDensityCalc, CalcTimeBetweenFirstandLast}
- A2-vel elbírálva: [1, 1, 0]; A3-al elbírálva: [1, 1, 1]


Workflow Inputs
release_count_threshold
sf_unixname
GetData
Workflow Inputs
release_density_threshold
sf_unixname
GetReleases
Workflow Inputs
sf_unixname
buildQueryWhere
MakeQuery
extract_release_dates
ConvertEpochToXSD
MergeReleasesToYamlArray
Workflow Outputs
releases_as_xml
GetFLOSSmoleData
Workflow Inputs
sf_unixname
buildFLOSSmoleStatisticsQueryString
BuildFLOSSmoleURLQueryString
GetDataFromFLOSSmoleSQL
GetURLFromFLOSSmole
split_SQL_results
MatchSFURL
ConvertSQLDateToXSDDateTime
Workflow Outputs
data_for_date
aggregate_downloads
lifespan_days
has_sf_url
SummarizeReleases
Workflow Inputs
list_of_release_dates_xsd_datetime
cutoff_date
num_releases_for_density
SplitOutReleases
TruncateReleasesList
count_releases
GetFirstRelease
GetLastRelease
GetReleaseForDensityCalc
TimeBetweenLastAndCutoff
CalcTimeBetweenFirstAndLast
CalcDensityLength
Workflow Outputs
time_between_last_release_and_cutoff
release_count
time_between_first_last_release
recent_release_density
Workflow Outputs
time_since_last_release
downloads
num_releases
first_last_release
recent_release_density
web_site
lifespan_days
mortality_threshold
download_threshold
initiation_age_threshold
release_lag_threshold
release_rate_type
mortality_test
release_count_test
downloads_test
growth_stage_test
release_lag_test
classifier
delist_stages
delist_classtypes
classification_csv
Stages_Analysis
Class_Analysis
Workflow Outputs
Class_Output
Classification_Data
Stages_Output

Példa 3: Proteine molecule simulation workflow of the ProSim project (Kiss et al, 2010):

- Készítette: University of Westminster
- Workflow környezet: WS-PGRADE Portal
- Célja: A workflow in-silico modellezési módszerrel szimulálja a proteinek és ligandok összekapcsolódását. A folyamat már meglévő szimulációs csomagokból építkezik. A workflow grid infrastruktúra segítségével csökkenti a szimuláció futási idejét.
- Egy olyan tranzakciósor, amelyre A2↔A3, viszont A3↾A2:
  $T^S = T_1, T_2, T_3, T_4, T_5$;
  R_1={Phase3.1}
  R_2={Phase2}
  R_3={visu2}
  R_4={PSagent}
  R_5={ Phase4.1, visu4.2}
- A2-vel elbírálva: [1, 1, 0, 1, 1]; A3-al elbírálva: [1, 1, 1, 1, 0]
  Megjegyzés: $T^{S2} = T_1, T_2, T_3$ példa olyan tranzakciósorra, amelyre A2↖A3



4-21. ábra: Proteine molecule workflow WS-PGRADE környezetben

5. Javított zároló algoritmusok

5.1. Az eddigieknél jobb megoldás

Az előző részek vizsgálatai megállapították, hogy az A2 algoritmus minden esetben legalább olyan jó (ugyanolyan jó vagy jobb), mint az A1 algoritmus. Az A2 és A3 algoritmusok, illetve az A1 és A3 algoritmusok között viszont nem lehet egyértelmű jósági döntést hozni, egyiket a másikkal szemben egy tetszőleges gráfokat és együttműködéseket támogató rendszerbe beépíteni. Ha előre ismert lenne egy kollaboratív szerkesztés során a létrehozandó gráf felépítése és az, hogy a gráfot milyen sorrendben és mekkora darabokban kívánják szerkeszteni a felhasználók, akkor persze arra a konkrét esetre lehetne az A2 vagy az A3 algoritmus javára dönteni. De ez nyilvánvalóan abszurd feltételezés. Ha előre ismert a gráf szerkezete, akkor mi szükség kollaboratív szerkesztésre? Ha előre ismert, hogy ki mikor mit módosít majd, akkor miért kell valós idejű szerkesztőrendszer?

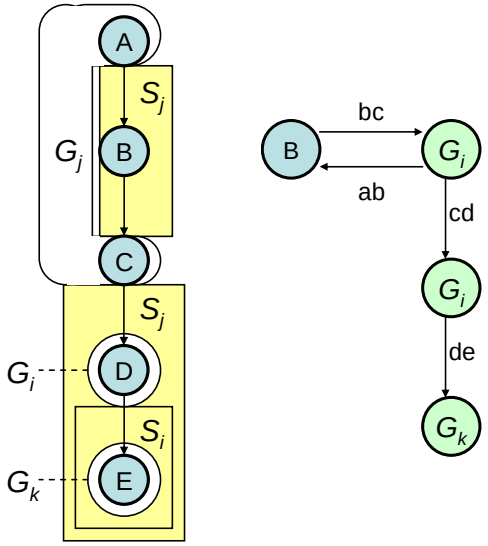
Munkámmal olyan gráf-szerkesztési módot kívánok adni, mellyel *előre nem ismert, tetszőleges struktúrájú gráfokat tetszőleges munkamenet szerint dolgozó felhasználók tudnak egyidőben szerkeszteni*. Sőt, ezen felül célom *megtalálni egy olyan eljárást, amely fejlesztés során a gráfok konzisztenciáját a lehető legtöbb felhasználó párhuzamos munkájának engedése mellett biztosítja.*

Láttuk, hogy vannak olyan szerkesztési esetek, amelyeket A2 tilt és A3 enged, és olyanok is amelyeket A3 tilt és A2 enged. Joggal merül fel a kérdés, hogy *vajon van-e olyan A4 zárolási kéréseket elbíráló algoritmus, amely jobb, mint A2 és jobb, mint A3*? A4 tetszőleges gráf és tetszőleges tranzakciósor esetén legalább annyi tranzakciót kell, hogy engedjen, mint A2 vagy A3 teszi, bizonyos esetekben pedig még náluk is többet. Amikor többet enged azoknál, akkor A4 jobb, mint A2 és A3. Ha ugyanannyit enged, akkor legalább olyan igazságos kell hogy legyen, mint azok. A4 létrehozására a következő 3 mód adódik:

1. *A4 létrehozására az A2 és A3 algoritmusok „sorbakötésével”:* Ez a mód hasonló ahhoz, ahogy A2-t a 3.6.1 részben létrehoztam eredetileg két önállónak kezelt zároló algoritmusból. Az A2 és A3 integrációjaként létrejövő A4 először az egyik, majd a másik algoritmust hívná meg, az ő együttes segítségükkel bírálva el a beérkező tranzakcióindítási kéréseket. Akár az egyik, akár a másik algoritmus engedi egy tranzakció indulását, A4 is engedi azt. A4 csak akkor tiltana egy tranzakciót, ha A2 *és* A3 is tiltja azt. Engedés esetén természetesen annak az algoritmusnak a zár topológiáját használva kell A4-nek lefoglalnia a gráf komponenseket, amelyik algoritmus az engedélyt megadta. Az A4-be integrált A2 és A3 algoritmusok azonban ettől még nem működhetnek egymástól elszigetelten: *fel kell ismerniük a saját maguk által adott zárak mellett a másik által adott kizárólagos foglalásokat.* (Vagyis A3-nak fel kell ismernie A2 U lock-ját, de nem kell felismernie A2 S lock-ját, míg A2-nek fel kell ismernie A3 U lock-ját.)
2. *A4 létrehozására az A2 algoritmus kiterjesztésével:* Ez a módszer A2 algoritmust terjeszti ki A4-gyé oly módon, hogy az felismerje és engedje az A2 által tiltott, A3 által engedett eseteket. Az ilyen esetek az 4.5.2 részben definiált kritériumrendszerre illenek, és annak segítségével ismerhetők fel.
3. *A4 létrehozására az A3 algoritmus kiterjesztésével:* Ez a módszer az előzőhöz hasonló, viszont itt A4 az A3 algoritmust terjeszti ki olyan módon, hogy az felismerje és engedje az A3 által tiltott, A2 által viszont engedett eseteket. Az ilyen esetek az 4.5.4 részben definiált kritériumrendszer segítségével ismerhetők fel.

A4-ben azért elegendő A2 és A3 felhasználása és A1 kihagyása, mert a korábbiakból ismert, hogy nincs olyan kollaboratív szerkesztési eset, amely során A1 A2-nél és A3-nál is jobb. Emiatt viszont a kizárólag A2-t és A3-at használó A4 algoritmus biztos, hogy lesz olyan jó, mint A1, így annak felhasználására nincs szükség.

Könnyű lenne belátni, hogy a fenti módszerek mindegyike helyes és jó eredményt adó A4-et hoz létre: a kapott algoritmus megőrzi a gráfok konzisztenciáját és van legalább olyan jó, mint A2 és A3. Sőt, az is belátható lenne, hogy bármelyik módszerrel is hozzuk létre A4-et az ugyanazt az eredményt fogja adni: ugyanazokat a tranzakciókat fogja tiltani, ugyanazokat a tranzakciókat fogja engedni. Az 5-1. ábra mutat egy példát olyan gráf-tranzakciósor párra, amelyre egy ilyen A4 algoritmus nagyobb párhuzamosságot enged, mint akár A2, akár A3.



$T^S=T_j, T_k, T_i$

$R_j=\{A, C\} \Rightarrow G_j=\{A, C\}, S_j=\{B, D, E, ab, bc, cd, de\}$
$R_k=\{E\} \Rightarrow G_k=\{E\}, S_k=\varnothing$
$R_i=\{D\} \Rightarrow G_i=\{D\}, S_i=\{E, de\}$

A2 engedi T_j-t és T_k-t, de tiltja T_i–t, mivel $S_j \cap G_i \neq \varnothing \wedge S_i \cap G_k \neq \varnothing$.
A3 tiltja T_j-t, de engedi T_k-t és T_i-t.
A4 engedi T_j-t, T_k-t és T_i-t is.

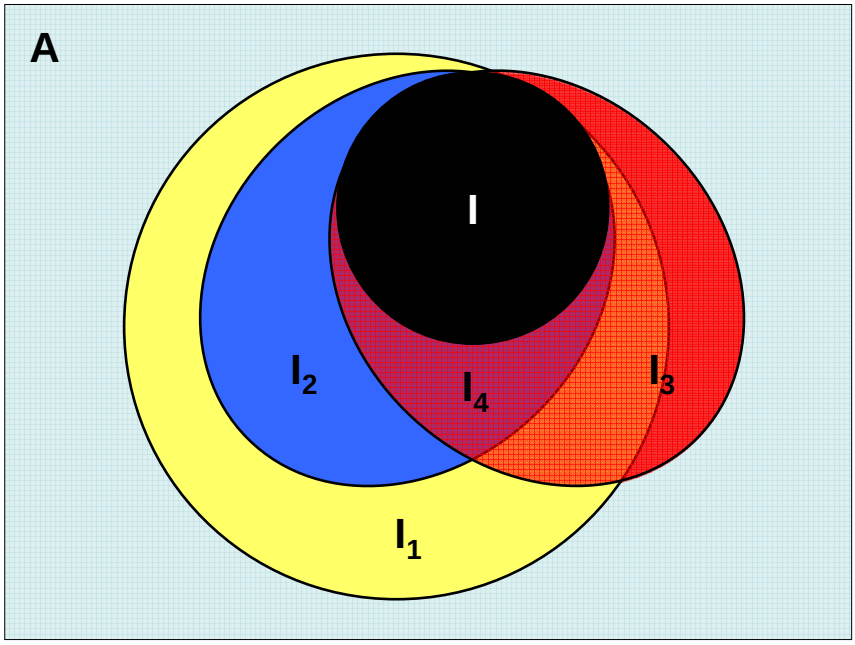
5-1. ábra: Példa olyan szerkesztési esetre, melyben az A2 és A3 integrációjaként kapott A4 azoknál jobb eredményt ad. A bal oldalon a gráf, mellette pedig az ebből A3 által készített szomszédossági gráf látható.

Mivel A4-et az A2 és A3 integrációjaként hoztam létre, ezért az 4.7 fejezetben ismertetett szerkesztési esetekben mind legalább olyan jó eredményt ad, mint A2 vagy A3. Ez különösen a harmadik példánál hasznos, ott ugyanis A2 és A3 is tilt olyan tranzakciót, amelyet A4 enged, emiatt erre az esetre A4↖A2 és A4↖A3, vagyis az [1, 1, 1, 1, 1] eredményvektort adná.

5.2. A „tökéletes” algoritmus

Van lehetőség A4-nél is jobb algoritmust létrehozni? Esetleg egy olyan „tökéletes” algoritmust, amelynél nem lehet nagyobb párhuzamosságot biztosító változatot csinálni? Elképzelhető, hogy egy ilyen, maximális párhuzamosságot biztosító algoritmus több módon is megadható. A dolgozatban eddig bemutatott munkából kiindulva, most a v4 algoritmus továbbfejlesztésével szeretnék egy ilyen algoritmust megadni. Ehhez először fel kell térképezni azokat a szerkesztési eseteket, amelyeket A4 tilt, és meg kell érteni, hogy melyek azok az esetek melyeket jogosan, és melyek amiket „jogtalanul” tilt. *Jogosan tiltott tranzakciósor esetén valóban sérülne a gráf konzisztenciája, ha az összes tranzakció lefut.* Jogtalan tiltáskor viszont a tranzakciósor összes tranzakciójának lefutása és eredményének mentése esetén sem keletkezne inkonzisztens gráf, legalább egy tranzakció viszont mégis tiltott. (A későbbiekben ezeket az eseteket „jogtalanul tiltott”, vagy „feleslegesen tiltott” esetnek hívom. A két fogalom alatt ugyanazt értem.)

Az 5-2 ábrán látható Venn-diagramos ábrázolásban az összes lehetséges gráf és a rajtuk futó összes lehetséges tranzakciósor, mint az *A* alaphalmaz elemei láthatók. *Az A halmaz egy-egy eleme egy gráf és egy azon értelmezhető tranzakciósor kettőse*: $a \in A, a = \{G, T^S\}$. Mivel bármilyen gráfot lehet szerkeszteni, ezért egy tranzakciósor kiindulási pontjaként végtelen különböző gráf létezik. Egy gráfon véges számú – a gráf összetettségétől függően több vagy kevesebb – különböző módosítási utasítássort lehet végrehajtani, vagyis véges számú, de általában nagyon nagy számú szerkesztési eset társítható ugyanahhoz a kiindulási gráfhoz. Összességében kijelenthető, hogy *A elemeinek száma végtelen*.



Jelmagyarázat:

A: Alaphalmaz. Az összes elképzelhető szerkesztési eset ({*G, T*^{S}} párok).
I: Inkonzisztens gráfot eredményező szerkesztési esetek.
I$_1$: A1 algoritmus által tiltott szerkesztési esetek.
I$_2$: A2 algoritmus által tiltott szerkesztési esetek.
I$_3$: A3 algoritmus által tiltott szerkesztési esetek.
I$_4$: A4 algoritmus által tiltott szerkesztési esetek.

5-2. ábra: Szerkesztési esetek és algoritmusok tiltásainak összevetése

Az *I* halmazba olyan elemek tartoznak, amelyekben az adott gráfon az adott tranzakciósor összes tranzakcióját végrehajtva inkonzisztens gráfot kapunk. *Ilyen esetben az tranzakciók utasításai együttesen vagy szabad élt, vagy többszörös bejövő élt, vagy körutat eredményeznek a gráfban, de az is előfordulhat, hogy a tranzakciósorban több tranzakció módosításai egymás munkáját felülírják.* Ez utóbbi felülírási eset akkor áll fenn, ha több tranzakció dolgozik a gráf ugyanazon komponensével és ilyenkor a komponens végállapota bizonytalan kimenetelű. (Természetesen szinkronizációs védelem nélkül! A komponens végállapota ilyenkor a később lezáródó tranzakció változtatásaitól függ.) Nyilvánvaló, hogy inkonzisztens gráfot eredményező {*G, T*^{S}} párból szintén végtelen számú létezik, vagyis *I* elemeinek száma is végtelen. Az 5-2. ábrán az egyes halmazok egymáshoz viszonyított mérete nem próbál utalni azok elemszámára.

I$_1$, *I$_2$*, *I$_3$*, és *I$_4$* halmazai rendre azokat a szerkesztési eseteket tartalmazzák, amelyek végrehajtása ellen egy A1, egy A2, egy A3, illetve egy A4 algoritmust használó kollaboratív

szerkesztőrendszer védekezik. *Egy-egy, ezen halmazok valamelyikébe tartozó elemet az adott algoritmust használó rendszer konzisztenciát sértő módosításnak ítél és emiatt az abban szereplő tranzakciósor legalább egy tranzakcióját tiltja*. (Lehet, hogy több tranzakciót is!) *A tiltott tranzakció(k) lefutása nélkül a tranzakciósor lefutása már nem sérti a gráf konzisztenciáját, vagyis a gráf végállapotában sem kört, sem problémás élt nem tartalmaz és minden komponensének végállapota kiszámítható, nincs több tranzakció általi felülírás.*

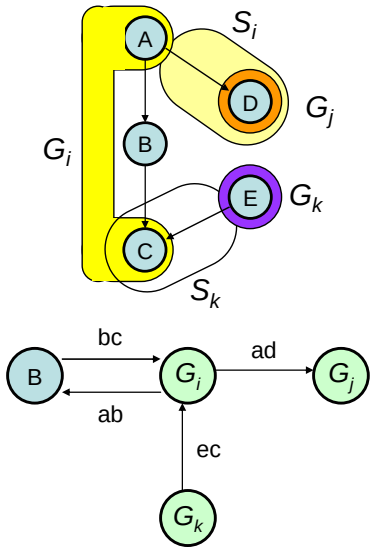
A 3.6 illetve a 3.7 fejezetekben bizonyítottam, hogy A1, A2 és A3 is bármilyen tetszőleges gráf és azon futó tetszőleges tranzakciósor esetén megőrzi a gráf konzisztenciáját. Bár ugyanezt az előbbi részben definiált A4 algoritmusra külön nem bizonyítottam, de mivel az az A2 és A3 integrálásaként jön létre, ezért ő is rendelkezik a konzisztenciőrzés tulajdonságával. *Ezekből következően a halmazábrában* $I \subset I_1$, $I \subset I_2$, $I \subset I_3$ *és* $I \subset I_4$, *vagyis minden valóban inkonzisztenciát okozó szerkesztési esetet minden algoritmus felismer és tilt.*

Az I_1, I_2, I_3 halmazainak egymáshoz viszonyított kapcsolata az előző, 4. fejezet konklúzióból adódik:

- Mivel nincs olyan szerkesztési eset amelyet A1 enged, A2 pedig tilt, viszont fordított eset létezik, ezért $I_2 \subset I_1$ (4.4.4 rész alapján).
- Mivel vannak olyan esetek amelyeket A2 tilt és A3 enged ezért $I_2 \not\subset I_3$ (4.5.1 rész alapján). Az ábrán ezek az esetek az $(I_2 \setminus I_3)$ részhalmazban vannak.
- Mivel vannak olyan esetek amelyeket A3 tilt és A2 enged ezért $I_3 \not\subset I_2$ (4.5.3 rész alapján). Az ábrán ezek az esetek az $(I_3 \setminus I_2)$ részhalmazban vannak.
- Mivel vannak olyan esetek amelyeket A1 tilt és A3 enged ezért $I_1 \not\subset I_3$ (4.6 rész alapján). Az ábrán ezek az esetek az $(I_1 \setminus I_3)$ részhalmazban vannak.
- Mivel vannak olyan esetek amelyeket A3 tilt és A1 enged, ezért $I_3 \not\subset I_1$ (4.6 rész alapján). Az ábrán ezek az esetek az $(I_3 \setminus I_1)$ részhalmazban vannak.
- Mivel vannak olyan esetek amelyeket A1 és A3 is tilt, viszont A2 enged, ezért $(I_1 \cap I_3) \not\subset I_2$ (Például a 4-14. ábra mutat egy ilyen esetet.) Az ábrán ezek az esetek az $(I_1 \cap I_3) \setminus I_2$ részhalmazban vannak.
- Mivel van olyan eset amelyet A3 tilt, viszont A1 és A2 is enged, ezért $I_3 \not\subset (I_1 \cup I_2)$ (Például a 4-6. ábra mutat egy ilyen esetet.) Az ábrán ezek az esetek szintén az $(I_3 \setminus I_1)$ részhalmazban vannak (Mivel $I_2 \subset I_1$).

Mivel A4 „örökölte" A2-t és A3-at, ezért $I_4 = I_2 \cap I_3$. *I és I_4 kapcsolata viszont eddig még nem tisztázott. Vajon* $I \subset I_4$ *vagy* $I = I_4$, *másként megfogalmazva van olyan eset amelyet A4 feleslegesen tilt?* Ha van ilyen eset, akkor azt nyilvánvalóan mind *A2*, mind *A3* tiltja és ekkor $I \subset I_4$. Ebben az esetben I_4 – az 5-2. ábrán is látható módon – túlnyúlik az *I* halmazon. Ha viszont nincs ilyen eset, akkor $I = I_4$ és I_4 szerkesztési párhuzamosság tekintetében a lehető legjobb algoritmus. ($I_4 \subset I$ nem lehet, mivel tudjuk, hogy I_4 minden valóban inkonzisztenciát okozó szerkesztési esetet felismer.)

Bizonyíték arra, hogy valóban van A4 által feleslegesen tiltott eset – és emiatt az 5-2. ábrán *I* és I_4 kapcsolata helyesen ábrázolt – az 5-3. ábrán látható. Ebben a gráf-tranzakciósor párban a tranzakciók nem dolgoznak a gráf azonos komponensével, sőt, nem is eredményeznek kört vagy szabálytalan élt. Az A4 *algoritmus mégis tiltja a tranzakciósor utolsó tranzakcióját.*



T^S=T_j, T_k, T_i = {m(D)}, {m(E)}, {m(A), m(C)}
R_j={D} $\Rightarrow$ G_j={D}, $S_j = \varnothing$
R_k={E} $\Rightarrow$ G_k={E}, S_k={C, ec}

R_i={A, C} $\Rightarrow$ G_i={A, C}, S_i={D, ad}
$S_i \cap G_j \neq \varnothing \wedge S_k \cap G_i \neq \varnothing$ emiatt A2 tiltja T_i elindulását.

T_i elbírálásakor a szomszédossági gráfban kör van, emiatt A3 is tiltja T_i elindulását.

Mivel A2 és A3 is tiltja T_i elindulását ezért A4 is tiltja azt.

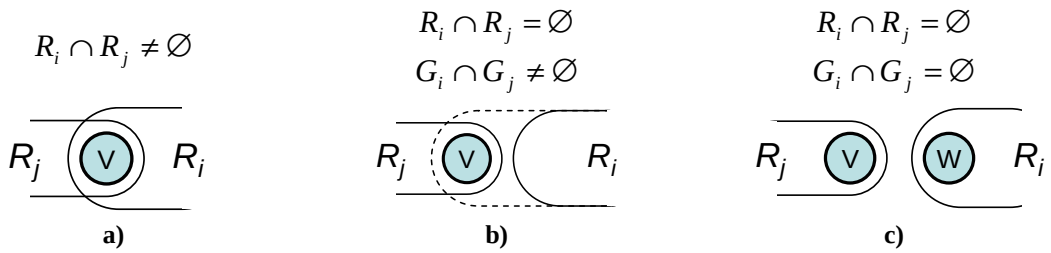
5-3. ábra: Példa inkonzisztenciát nem okozó, viszont A4 által tiltott szerkesztési esetre.

Az 5-3. *ábrán mutatott példa üzenete, hogy lehet A4-nél is jobb algoritmust készíteni.* Egy ilyen algoritmus legalább egy, az A4 által feleslegesen tiltott esetet felismer. *Egy, A4 továbbfejlesztéseként létrehozott legideális algoritmus viszont az összes A4 által feleslegesen tiltott esetet felismeri és kizárólag az inkonzisztens gráfot eredményező szerkesztési együttműködéseket tiltja. Következő célom egy ilyen, A4 továbbfejlesztéseként létrehozott „ideális" algoritmusnak a definiálása.*

5.2.1. Feleslegesen tiltott esetek feltérképezése

Maximális párhuzamosságú szerkesztést biztosító algoritmusból akár több is lehet. Vizsgálataim célja nem megmondani hogy hány ilyen algoritmus van, hanem megadni egyetlen ilyen algoritmust. Ehhez az A4 algoritmusból fogok kiindulni, azt fogom továbbfejleszteni. Először kategorizálni fogom az A4 által tiltott eseteket. Azután a kategóriák vizsgálatával eldöntöm, hogy mely elemek vannak jogosan, és melyek jogtalanul tiltva. Az A4 algoritmusnak a jogtalanul tiltott elemekre való felkészítésével azután meghatározok egy új, A5 algoritmust. *Ha az A4 által jogtalanul tiltott esetek teljes terét sikerül bejárni és ezek mindegyikének felismerése és engedése sikerül A5-ben, akkor az A5 algoritmus egy maximális párhuzamosságú szerkesztést biztosító algoritmus.*

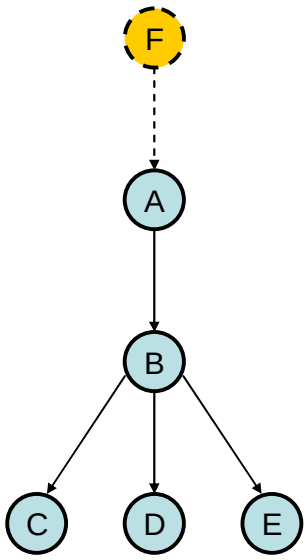
Ha egy konzisztenciát nem sértő T^S tranzakciósor valamely T_i tranzakcióját az A4 algoritmus tiltja, akkor annak a következő okai lehetnek (ld. még 5-4. ábra.): (Ebben a szakaszban csak az egyes típusokat írom le, később pedig minden típust külön-külön megvizsgálok).



5-4. ábra: A4 által tiltott, konzisztenciát nem sértő tranzakció kategorizálása. (Két tranzakció gráfot érintő módosításának átfedési lehetőségei.)

a) A tiltott tranzakció (T_i) azonos komponenst (V csomópont) akar módosítani a tranzakciósor valamely másik tranzakciójával (T_j).
b) A tranzakció nem akar azonos komponenst módosítani semelyik másik tranzakcióval, viszont az algoritmus a konzisztenciaőrzés miatt egy másik tranzakció részgráfjában lévő komponenst is akar számára zárolni.
c) A tiltott tranzakció és a tranzakciósor más tranzakcióinak zárolásra kapott részgráfjai között nincs átfedés.

1. A T_i tranzakció valamely olyan komponenst akar módosítani, amelyet a tranzakciósor egy már futó tranzakciója (T_j) kizárólagos zárolással birtokol. Ilyenkor előfordulhat, hogy a két tranzakció a komponens ugyanazon paraméterét változtatná meg és valóban kivitelezhetetlen a párhuzamos szerkesztés. *Viszont lehet olyan eset is, hogy T_i és T_j munkája a komponens különböző paramétereit érinti. Ilyen esetben T_i és T_j elvileg futhatna egyidőben és így T_i feleslegesen tiltott.* Az eddig tárgyalt rendszerek – beleértve A4-et is – az egyidejű futást ilyen esetben sem teszik lehetővé, ugyanis *a kollaboratív szerkesztőrendszer gráf komponens szinten kezeli a zárakat, nem képes ennél kisebb granularitással dolgozni. Ennek következménye, hogy két tranzakció indulási kérése között akkor is átfedés van, ha azok munkája egyazon komponens különböző paramétereire irányul*: $R_i \cap R_j \neq \emptyset$, $G_i \cap G_j \neq \emptyset$. Ilyenkor a később indulást kért tranzakció – esetünkben T_i – tiltást kap. Egy ilyen esetre mutat példát az 5-5. ábra.



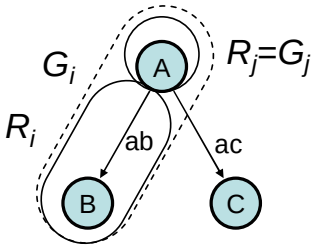
- $T^S = T_k, T_j, T_i$
- T_k új input fájlt akar definiálni az A csomópont számára, munkája az A csomóponthoz egyik paraméterének módosítását jelenti:
  $T_k = \{m(A)\} \Rightarrow R_k = G_k = \{A\}$
- T_j módosítani akarja az A csomópont futtatásához szükséges erőforrás-paramétereket (mást, mint T_k):
  $T_j = \{m(A)\} \Rightarrow R_j = G_j = \{A\}$
- T_i egy új csomópontot akar létrehozni és hozzákapcsolni az A csúcshoz (F csomópont). Munkája az A csomóponthoz egy új paramétert rendelne (mást, mint T_k vagy T_j):
  $T_i = \{m(A)\} \Rightarrow R_i = G_i = \{A\}$

$R_k \cap R_j \neq \emptyset \Rightarrow T_j$-t A2 és A3 is tiltja ezért A4 is tiltja.
$R_k \cap R_i \neq \emptyset \Rightarrow T_i$-t A2 és A3 is tiltja ezért A4 is tiltja.

5-5. ábra: Példa az A4 által feleslegesen tiltott olyan szerkesztési esetre, melyben több tranzakció ugyan azonos gráfkomponenssel dolgozik, viszont munkájuk a komponens különböző paramétereit érinti.

2. A tranzakciósorban nincsenek olyan tranzakciók, amelyek ugyanazokkal a gráf komponensekkel akarnak dolgozni és emiatt a tranzakciók által igényelt részgráfok sincsenek átfedésben: $R_i \cap R_j = \emptyset$. A zárolási kéréseket elbíráló algoritmus viszont a konzisztenciaőrzés miatt az igényeltnél nagyobb részgráfokat akar valamely tranzakció

számára lefoglalni, és a kiterjesztett részgráf már zár ütközést okoz: $G_i \cap G_j \neq \emptyset$. Egy ilyen esetre mutat példát az 5-6 ábra.

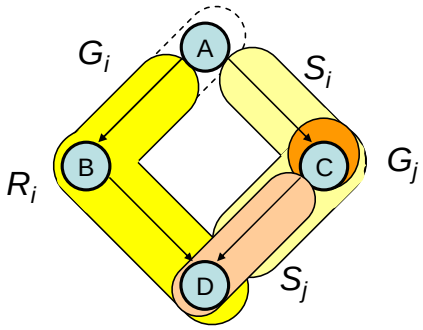


T^S=T_j, T_i = {m(A)}, {m(ab), m(B)}
(T_j az A csomópontot, T_i az ab élt és a B csomópontot módosítaná.)
R_j={A} $\Rightarrow$ G_j={A}
R_i={B, ab} $\Rightarrow$ G_i ={A, B, ab}
$G_i \cap G_j \neq \emptyset$ $\Rightarrow$ T_i-t A2 és A3 is tiltja ezért A4 is tiltja.

5-6. ábra: Példa az A4 által feleslegesen tiltott olyan szerkesztési esetre, melyben a zároló algoritmus gráfkiterjesztése egymással átfedésben levő részgráfokat, és emiatt indulási tiltást eredményez.

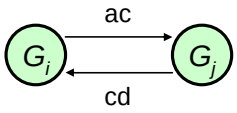
Megjegyzés:

Ebbe a kategóriába csak azok az esetek tartoznak, amelyekben a T_i tranzakció részgráfjának kiterjesztése U-U ütközést okoz valamely, már futó tranzakció részgráfjával. Létezik azonban olyan eset is, amelyben a kiterjesztés nem U-U, hanem U-S vagy S-U ütközést okoz, viszont ez egy már meglévő S-U illetve U-S ütközéssel együtt szintén tranzakciótiltást eredményez (ld. 5-7. ábra.). Az ilyen esetek a következő, a 3. kategóriába tartoznak, ugyanis ezeknél nincs a kizárólagosan lefoglalandó részgráfok között átfedés ($G_i \cap G_j = \emptyset$).



T^S=T_j, T_i = {m(A)}, {m(ab), m(B)}
R_j={C} $\Rightarrow$ G_j={C}
R_i={B, D, ab, bd} $\Rightarrow$ G_i={A, B, D, ab, bd}
S_j={D, cd}
S_i={C, ac, cd}

$G_i \cap G_j = \emptyset$, de
$S_i \cap G_j \neq \emptyset \wedge S_j \cap G_i \neq \emptyset$ emiatt A2 tiltja T_i elindulását.



T_i elbírálásakor a szomszédossági gráfban kör van, emiatt A3 is tiltja T_i elindulását.

5-7. ábra: Példa az A4 által feleslegesen tiltott olyan szerkesztési esetre, melyben a zároló algoritmus gráfkiterjesztése olyan U-S ütközést okoz, amely már meglévő S-U ütközéssel együtt tranzakciótiltást eredményez.

3. A T_i tranzakció nem akar olyan komponenst módosítani, amely már kizárólagosan zárolt, és a zárolási kéréseket elbíráló algoritmus esetlegesen részgráf-kiterjesztése sem okoz a kizárólagosan lefoglalandó gráfban átfedést: $R_i \cap R_j = \emptyset$ és $G_i \cap G_j = \emptyset$. Egy ilyen esetre mutatott példát az 5-3. ábra.

A következőekben azt fogom megvizsgálni, hogy mi alapján lehet a fenti három kategóriába eső szerkesztési eseteket felismerni, és hogyan, illetve milyen kompromisszumokkal lehetne az A4 zárolási kéréseket elbíráló algoritmust az engedésükre felkészíteni.

5.2.1.1. Átfedésben levő zárolási kérések

Ahogy az 5.2.1 fejezetben szereplő felsorolás 1. pontjában írtam, előfordulhat, hogy a felesleges tiltásnak az oka, hogy a kollaboratív szerkesztőrendszer *gráf komponens szinten kezeli a zárakat (csomópont, él a zárolható egység) és nem tud ennél alacsonyabb szemcsézettséggel dolgozni. Emiatt két felhasználó egyazon komponensre vonatkozó változtatásai nem folyhatnak egyidőben még akkor sem, ha a felhasználók munkája valójában a komponens különböző paramétereit érintik.*

Ez a probléma feloldható, de csak alacsonyabb szemcsézettségű zárolással. Alacsonyabb szemcsézettségű zároláskor a rendszernek nem csomópont és él, hanem azon belüli paraméter szinten kell nyilvántartani a zárakat. Hogy a rendszer tudja mikor mit kell lock-olni, a felhasználóknak szintén paraméter szinten kell megadniuk a rendszer számára, hogy a gráf mely komponensének milyen paramétereit akarják módosítani. *Ennek az információnak a felhasználótól kell származnia, magától a rendszer nem képes előállítani.* Ez a megadás történhet például úgy, hogy a grafikus felületen a csomópont vagy él paraméterlistában a felhasználó bejelöli, hogy az *x*, *y* és *z* paramétereket akarja módosítani.

A finomabb szemcsézettségű megadás az elbíráló algoritmus számára lehetővé teszi komponensek részbeni foglalását azokban az esetekben, ha a felhasználó munkája nem érinti a komponens egészét, csak annak bizonyos paramétereit. Ez igaz akkor, ha csak 1-2 paramétert akar a felhasználó változtatni egy csomóponton vagy egy élen, és akkor is, amikor egy új élt akar egy csomóponthoz kötni. (A 3.4.2. részben részletesen tárgyaltam, hogy az él létrehozás miért számít paraméterváltoztatásnak.) Ha csomópontot vagy élt töröl, akkor nem élhet ezzel a lehetőséggel, mivel a komponens törlés implicit módon az összes paraméter törlését jelenti.

Annak ellenére, hogy az alacsonyabb szemcsézettségű zárolás segítene feloldani az ilyen típusú felesleges tiltásokat, megjelenése jelentős bonyodalmat hoz a felhasználók életébe:

- Jóval pontosabban kell szervezniük a munkájukat, ugyanis előre kell jelezniük, hogy módosítani, vagy törölni akarnak-e egy komponenst. Módosítás esetén ráadásul azt is jelezniük kell, hogy milyen paramétereket érint a munka.
- Jóval nehezebb áttekinteniük, hogy ki milyen változtatásokat végez épp a gráfon. Amíg a csomópont és él szintű zárolás esetén ránézésre látni lehetett, hogy ki melyik részt szerkeszti, addig az alacsonyabb szemcsézettség miatt akár több személy között is meg lehet osztva egy komponens. Ha csak egy személy dolgozik egy komponensen, akkor sem látható egyből, hogy csak paramétereket módosít és be lehet még társulni, vagy törölni fog és kizárólag ő érheti el a komponenst.

Emiatt a nehézségek miatt nem javaslom az A4 zárolási kéréseket elbíráló algoritmus alacsonyabb szemcsézettség felé történő kiterjesztését.

5.2.1.2. Részgráf-kiterjesztés miatti átfedések

Ahogy az 5.2.1 fejezetben szereplő felsorolás 2. pontjában írtam, előfordulhat, hogy a felesleges tiltás oka valamely olyan gráf komponens (vagy több komponenst tartalmazó gráfrész) amelyet a konzisztenciaőrzés miatt a zároló algoritmus ad az ütközésben részt vevő valamely

tranzakció U lock-kal lefoglalandó részgráfjához. Ha a hozzáadott komponens valamely futó tranzakció számára már U lock-kal le van foglalva, akkor ütközés történik.

A lefoglalandó részgráf kiterjesztésére a zároló algoritmusokban a szabad él és a többszörös bejövő él elleni védelem miatt van szükség. Az A4 algoritmusban (és A1, A2, és A3-ban is) gráf kiterjesztés a 3.4.1 részben ismertetett P1 szerint történik akkor, ha egy tranzakció (T_i) olyan élt kér szerkesztésre, amelynek végpontját (vagy végpontjait) nem akarja szerkeszteni. Ilyen esetben az elbíráló algoritmus ezen végponto(ka)t is T_i U zárral lefoglalandó, G_i részgráfjához adja. Felesleges tiltás ebből akkor származik, ha

1. A hozzáadott csomópont zárolása ütközést okoz egy már futó tranzakció (T_j) által birtokolt részgráffal. Ez esetben a zárolás nem végezhető el, T_i tranzakció nem indítható. Mivel komponens szintű zárak ismeretében pontosan nem tudható, hogy T_i és T_j tranzakciók mit is csinálnak a lefoglalt részgráfjaikkal (csak az, hogy milyen komponensekkel dolgoznának), ezért a kiterjesztett komponensek kizárólagos zárolására szükség van, az nem hagyható el.
2. T_i részgráfjához adott csomópont lefoglalása T_i tranzakció indulásakor nem okoz ütközést már futó tranzakcióval, viszont később, ugyanazon a tranzakciósoron belül jön egy T_h tranzakció, amely azért nem indítható, mert részgráfja ütközik a zárolási kéréseket elbíráló algoritmus által T_i részgráfjához adott komponenssel. Mivel T_i elbírálásakor előre nem tudható, hogy milyen tranzakcióindítási kérések fognak a tranzakciósoron belül jönni, ezért akár olyan is jöhet, amelyik törölni akarja a T_i által szerkesztett él végpontját, T_i értékes munkáját semmisítve meg. T_i gráfjának kiterjesztése és a kiterjesztett részgráf kizárólagos zárolása tehát ekkor sem hagyható el.

Mindkét variáció esetén hasonló helyzet áll fenn, mint az előző, 5.2.1.1 részben tárgyalt eset: két tranzakció ugyanazt a csomópontot kéri zárolásra, viszont munkájuk nincs feltétlenül ütközésben, a lefoglalásra kért komponenseken belül nem feltétlenül ugyanazokat a paramétereket akarják módosítani. *A megoldás ennek következtében itt is hasonló lehet, mint előbb: be kell vezetni az alacsonyabb, paraméter szintű zárolás lehetőségét. A hátrányok szintén ugyanazok és emiatt továbbra sem javaslom az A4 zárolási kéréseket elbíráló algoritmus alacsonyabb szemcsézettség felé történő kiterjesztését.*

5.2.1.3. Nem felismert párhuzamosíthatóság

Ahogy az 5.2.1 fejezetben szereplő felsorolás 3. pontjában írtam, előfordulhat, hogy a felesleges tiltás azért van, mert ugyan T_i engedése nem okozna inkonzisztens gráfot, az A4 algoritmus által használt „inkonzisztencia előrejelzési módszerek” ezt jelzik. Tudjuk, hogy ilyen esetben

(1) A T_i tranzakció indulásakor a G gráf szomszédossági gráfjában kör van. (emiatt A3 tilt)
és
(2) A T_i tranzakció az indulásakor a G gráfban U-S és S-U ütközést okoz. (emiatt A2 tilt)
és
(3) A T_i tranzakció számára U lock-kal lefoglalandó részgráf (G_i) nincs átfedésben egyetlen, már futó tranzakció által zárolt részgráffal sem. (Különben az 5.2.1.1 vagy az 5.2.1.2 fejezetekben tárgyalt okok miatt tiltaná A4 a T_i elindulását.)

Ezek a feltételek elég tágak, többféle gráf konfigurációban is fennállhatnak. Pontosabb meghatározására van szükség. Ezt a pontosabb meghatározást az (1) kritérium alapján történő[12] alkategóriák felállításával fogom elvégezni. Az (1) pont értelmében tehát tudjuk, hogy amikor A4 tilt, akkor a szomszédossági gráfban kör van. Ez a kör a következő módok valamelyikén jöhet létre:

1. A kör kialakításában a G_i részgráf és még nem lefoglalt csomópontok vesznek részt.
2. A kör kialakításában a G_i részgráf mellett további részgráf(ok) is részt vesz(nek). Ebben az esetben biztos, hogy a kör megtörik valamelyik részgráfon belül, különben valamely, a T_i-t megelőző tranzakció indulása már kört hozott volna létre a szomszédossági gráfban és nem futhatna. A kör megtörhet:
    i. A G_i részgráfon belül
    ii. A G_i részgráfon kívül, másik részgráf(ok)on belül
    iii. A G_i részgráfon belül és másik részgráf(ok)on belül is

Ez a négy eset (1, 2i, 2ii, 2iii) már külön-külön vizsgálható, és a gráf felépítésére és T_i-re a következő feltételrendszereket adja (Ld. még 5-8. ábra.):

1. eset (fenti 1.):
   - A G gráfban kört alkot a G_i részgráf (csomópontként tekintve) és még nem lefoglalt csomópontok.
   - A G_i részgráfon belül megtörik a kör.
   - A G_i részgráfhoz biztosan kapcsolódik még legalább egy bejövő zárolt részgráf (G_k) és egy kimenő zárolt részgráf (G_j), különben nem teljesülne a fenti (2)-es kritérium.

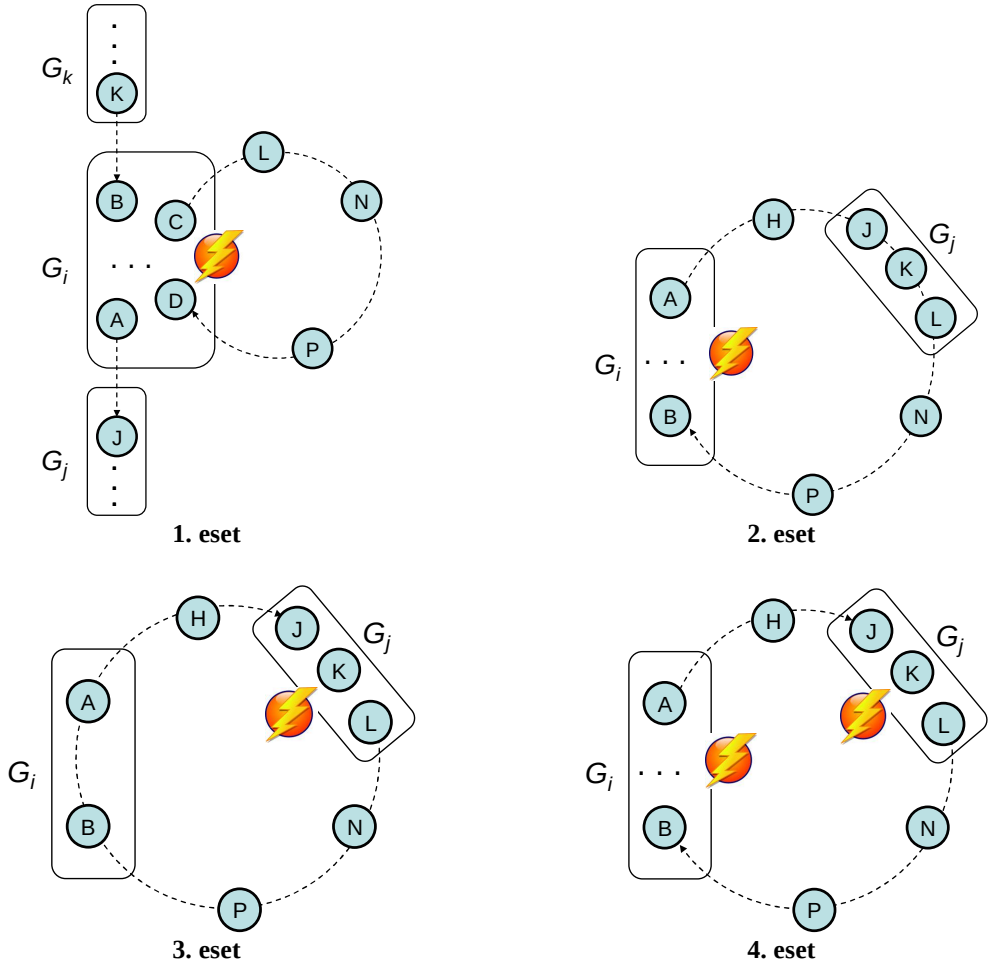
2. eset (fenti 2i.):
   - A G gráfban kört alkot a G_i részgráf (csomópontként tekintve) és zárolt részgráf vagy részgráfok (azokat szintén csomópontként tekintve), és 0 vagy több zárolatlan csomópont.
   - A G_i részgráfon belül megtörik a kör.
   - A G_i mellett a kör kialakításában résztvevő zárolt részgráfokon belül *nem törik meg* a kör.
   - Ebben az esetben nem szükséges hogy G_i részgráfhoz a körön kívüli zárolt részgráfok kapcsolódjanak, az előző oldalon szereplő (2)-es kritérium a G_j és G_i U és S zárjai miatt egyébként is teljesül.

3. eset (fenti 2ii.):
   - A G gráfban kört alkot a G_i részgráf (csomópontként tekintve), még legalább egy zárolt részgráf (G_j, szintén csomópontként tekintve), és 0 vagy több még nem zárolt csomópont.
   - A G_i részgráfon belül *nem törik meg* a kör.
   - A kör kialakításában G_i mellett részt vevő, zárolt részgráfok közül pontosan egy részgráfon belül (G_j) *megtörik* a kör.
   - Nem szükséges, hogy a G_i részgráfhoz a körön kívüli zárolt részgráfok kapcsolódjanak, az előző oldal (2)-es kritériuma egyébként is teljesül.

4. eset (fenti 2iii.):

[12] Lehetne a (2) és (3) pontban kimondott tulajdonságok alapján is felállítani az alkategóriákat, de én az (1) pont szerinti felállítást találtam a legszemléletesebb megoldásnak.

- A G gráfban kört alkot a G_i részgráf (csomópontként tekintve), még legalább egy zárolt részgráf (G_j, szintén csomópontként tekintve), és 0 vagy több zárolatlan csomópont.
- A G_i részgráfon belül megtörik a kör.
- A kör kialakításában G_i mellett résztvevő zárolt részgráfok közül *legalább egy részgráfon belül (G_j) megtörik a kör*.
- Nem szükséges, hogy a G_i részgráfhoz a körön kívüli zárolt részgráfok kapcsolódjanak, az előző oldal (2)-es kritériuma egyébként is teljesül.



5-8. ábra: A4 által fel nem ismert párhuzamosíthatóság esetei.
Az 1., 2. és 3. eset engedhető egy, a felhasználók számára csak U lock-ot láthatóvá tevő A5 algoritmussal, a 4. eset kezelése viszont csak egy további, látható zár típus bevezetésével lehetséges. A villámok a részgráfokon belüli kör megtöréseket jelölik.

Az 1. esetben a T_i tranzakció ugyan a *dc* él létrehozásával meg tudná törni a gráf konzisztenciáját, viszont az ilyen törést a kliens oldali szerkesztőrendszer képes felismerni és megakadályozni. Ilyenkor ugyanis a *kört alkotó összes él ismert a T_i-t futtató kliens oldali szerkesztőrendszer számára*: az élek ugyanis vagy már léteznek T_i elindulásakor, vagy T_i hozza létre őket. *Vagyis a T_i tranzakció engedése, és a dc él létrehozásának kliens oldali megakadályozása járható út. Egy A4-nél jobb A5 algoritmus tehát az 1. típusú esetek engedésére is felkészülhet a fent „1. eset” alatt felsorolt kritériumok figyelésével.*

Ez az eset egyébként is hasonló ahhoz a 4-16. ábra 2. részén látható és A4 által engedett esethez, amely az 5-8 ábra 1. részéből a T_j és T_k tranzakciók eltávolításával áll elő. A 4-16. ábra 2. részén látható esetben azért enged A4, mert a kliens oldali szerkesztőrendszer tiltja a *dc* él létrehozását. Ugyanakkor egy T_i-hez hasonló, viszont pl. a *cd* élt definiálni akaró T_i tranzakció futhat. A T_j és T_k tranzakciók hozzáadása ezen a helyzeten nem változtat.

A 2. esetben (5-8. ábra 2. része) A4 szintén feleslegesen tilt, hiszen a T_i tranzakció ugyan a *dc* él létrehozásával itt is meg tudná törni a gráf konzisztenciáját, viszont az ilyen törést a kliens oldali szerkesztőkörnyezet felismerheti és megakadályozhatja. Teljesen mindegy, hogy T_j pontosan milyen műveleteket hajt végre a saját gráfrészén, ha kör alakulna ki a gráfban, akkor azért most is egyedül T_i felelős. *A tranzakció engedése és az él létrehozás kliens oldali megakadályozása ezért itt is ésszerű és kivitelezhető megoldás. Egy A4-nél jobb A5 algoritmus tehát a 2. típusú esetek engedésére is felkészülhet a fent „2. eset" alatt felsorolt kritériumok figyelésével.*

A 3. esetben (5-8. ábra 3. része) szintén feloldható a tiltás és engedhető a T_i tranzakció. Mivel ekkor a T_i tranzakció által kért gráfrész már T_i indulásakor részt vesz a körben, T_i munkáján semmi nem múlik, tehát T_i engedhető. Ebben az esetben T_i indításakor a kliens oldali kör-figyelés akár a T_i részgráfon belül is elegendő. *Egy A4-nél jobb A5 algoritmus tehát a 3. típusú esetek engedésére is felkészülhet a fent „3. eset" alatt felsorolt kritériumok figyelésével.*

A 4. esetben (5-8. ábra 4. része) a kör létrehozása több tranzakció együttes munkáján múlik – és ez jelentős eltérés az előző három kategóriához képest! Az ábrán szereplő példában csak akkor jöhet létre kör, ha T_i létrehozna egy *dc* élt (D csúcsból C csúcsba futó élt), T_j pedig létrehozna egy *jn* élt (J csúcsból N csúcsba futó élt), vagyis *a kör kialakulásáért több, egyidőben futó tranzakció a felelős*. Mivel a komponens szintű zárolás miatt sem T_i-ről, sem T_j-ről nem tudható előre, hogy pontosan milyen éleket akarnak a saját részgráfjaikon belül létrehozni, ezért tényleg van esélyük akár kört is létrehozni, ezért egyiküket tiltani kell. Az eddig tárgyalt tranzakciók mind a körben utolsónak induló, esetünkben T_i tranzakciót tiltják. *Ha mindkettőt engednénk, és azok valóban létrehoznák a kört kialakító éleket, akkor nem lehetne objektív módon megállapítani a felelősséget és valamelyik tranzakciót úgy kéne abortálni, hogy az konzisztenciatörésért nem egyedül ő a felelős.* Hogy ilyen esetekben engedhessük a párhuzamos szerkesztést, ahhoz szintén alacsonyabb szemcsézettségű zárolást kéne bevezetni. Ez az előzőekhez hasonló problémákat jelentene a felhasználóknak, emiatt *továbbra sem javaslom az A4 zárolási kéréseket elbíráló algoritmus alacsonyabb szemcsézettség felé történő kiterjesztését.*

5.2.2. Egyféle zár segítségével elérhető legjobb algoritmus: A5

Az előző részekben ugyan több ponton is felvetettem az alacsonyabb szemcsézettségű zárak használatát azonban ezt mindig elvetettem azért, mert bevezetésük nem járna annyi előnnyel, hogy kompenzálja a használatukból adódó kellemetlenségeket. Az alacsonyabb szemcsézettségű zárak egyébként is csak akkor jelentenének valóban javulást a párhuzamosságban, ha a felhasználók tényleg a lehető legkevesebb, csak a munkájukhoz szükséges paramétereken kérnének zárakat. Amint valaki többet lock-olna mint amire valóban szüksége van, a párhuzamos szerkesztés lehetősége lecsökken, közelít a komponens szintű lock-olás által biztosítotthoz.

Az alacsonyabb szemcsézettségű zárakról való lemondás viszont egyben azt is jelenti, hogy lemondunk a maximális párhuzamosságot eredményező algoritmusról. Alacsonyabb szemcsézettség nélkül ugyanis nem oldható meg az 5.2.1.1 részben, az 5.2.1.2 részben, illetve az 5.2.1.3 rész 4. alkategóriájában tárgyalt esetek engedése. Ez a kijelentés akkor is igaz, ha nem az A4-ből kiindulva vizsgáljuk a feleslegesen tiltott eseteket. Ha maximális párhuzamosságú algoritmus létrehozása komponens szintű zárolással nem is lehetséges, azért az A4-hez képesti javításra van mód, hiszen az 5.2.1.3 rész 1., 2. és 3. alkategóriájában tárgyalt esetek finomabb

szemcsézettség nélkül, az eddig is használt User lock zárolással is engedhetők. Az A5 algoritmust tehát az A4 olyan kiterjesztéseként fogom megadni, amely képes felismerni és engedni az 5.2.1.3 rész 1., 2. és 3. alkategóriájában tárgyalt eseteket, viszont továbbra is csak U zárat tesz a felhasználók számára láthatóvá.

A A4 algoritmus metakódját korábban nem adtam meg, viszont az 5.1 részben ismertettem, hogy három féle módon képezhető. Most ezek közül bármelyiket vehetem A5 kiindulási pontjaként:

1. A2 és A3 algoritmusok egymás után kötése.
2. A2 algoritmus felkészítése az A3 által engedett esetekre.
3. A3 algoritmus felkészítése az A2 által engedett esetekre.

Mivel az A2 és A3 algoritmusokat már korábban leírtam, ezért az 1. módszert követve definiálható legegyszerűbben A4, majd belőle A5. Ehhez először az A2-t kell lefuttatni, majd az A3-at és ha mindkettő tilt, de a következő feltételek fennállnak, a tranzakció indulását akkor is engedni kell (Ezen feltételek az 5.2.1.3/1, 5.2.1.3/2 és 5.2.1.3/3 részekből vannak idemásolva):

1. eset:
   - A *G* gráfban kört alkot a G_i részgráf (csomópontként tekintve) és még nem lefoglalt csomópontok.
   - A G_i részgráfon belül megtörik a kör.
   - A G_i részgráfhoz biztosan kapcsolódik még legalább egy bejövő zárolt részgráf (G_k) és egy kimenő zár részgráf (G_j), különben nem teljesülne a fenti (2)-es kritérium.

vagy

2. eset:
   - A *G* gráfban kört alkot a G_i részgráf (csomópontként tekintve) és zárolt részgráf vagy részgráfok (azokat szintén csomópontként tekintve), és 0 vagy több még nem lefoglalt csomópont.
   - A G_i részgráfon belül megtörik a kör.
   - A G_i mellett a kör kialakításában résztvevő zárolt részgráfokon belül *nem törik meg* a kör.
   - Ebben az esetben nem szükséges hogy G_i részgráfhoz a körön kívüli zárolt részgráfok kapcsolódjanak, a (2) kritérium a G_j és G_i U és S zárjai miatt egyébként is teljesül.

vagy

3. eset:
   - A *G* gráfban kört alkot a G_i részgráf (csomópontként tekintve) és még legalább egy zárolt részgráf (G_j, szintén csomópontként tekintve), és 0 vagy több zárolatlan csomópont.
   - A G_i részgráfon belül *nem törik meg* a kör.
   - A kör kialakításában G_i mellett résztvevő zárolt részgráfok közül pontosan egy részgráfon belül (G_j) *megtörik* a kör.
   - Nem szükséges hogy a G_i részgráfhoz a körön kívüli zárolt részgráfok kapcsolódjanak, a (2) kritérium egyébként is teljesül.

Az első és második esetek algoritmikus felismerését megkönnyíti, hogy azok összevonhatók. Vegyük észre ugyanis, hogy ez a két eset akkor és csak akkor áll fenn (így, *vagy* kapcsolattal!), ha kör jön létre a *G* gráfban a G_i részgráf csomópontra való lecserélése után. Az, hogy vannak-e

a körnek zárolt komponensei (a második eset áll fenn), vagy nincsenek (az első eset áll fenn), tulajdonképpen mellékes. Akár az egyik, akár a másik is a helyzet, a tranzakciót akkor is engedni kell. Abban biztosak lehetünk, hogy az első esetben szereplő G_k és G_j részgráfokra vonatkozó kritériumok teljesülnek, ugyanis A2 már tiltotta az indulást. (A2 lefuttatása megelőzi a speciális esetek felismerését.) Az, hogy a második esetben nem törik meg a kör a már zárolt részgráfokon belül szintén biztosra vehető, hiszen csak G_i-t cseréltük csomópontra és úgy kaptunk kört.

A harmadik eset felismeréséhez először le kell generálni a G gráfhoz azt a „majdnem-szomszédossági" gráfot, amelyben minden, már zárolt részgráf csomópontként jelenik meg, a most foglalásra kért részgráf komponensei viszont változatlanul élekként és csomópontokként[13]. Ha ebben a „majdnem-szomszédossági" részgráfban kör van, akkor engedélyezhető a tranzakció indulása, ellenkező esetben nem. Ezek ismeretében A5 a következő módon írható le algoritmikus formában:

```
Input:  Ri – the set of objects that Ti intends to write from G
Output: Ui - the sub-graph that must be locked for Ti with user lock.
        If U is empty then lock must be denied.
Output: Si - the sub-graph that must be locked for Ti with system
        lock.

1) Ui = {}; Si = {}

//A2-vel való foglalás:
2) (Ui, Si) = A2(Ri)
3) If ( Ui <> {} ) then Return Ui, Si

//A3-mal való foglalás:
4) Ui = A3(Ri)
5) If ( Ui <> {} ) then Return Ui, Si

//Kivételfelismerésre való felkészülés:
6) Ui = Ri
7) For every edge of Ui {
      a) X = current edge of Ui
      b) Add X.source() to Ui
      c) Add X.sink() to Ui
8) For every component of Ui {
      a) X = current component of Ui
      b) If (X.isUserLocked()==TRUE) then {
            i)  Ui = {}
            ii) Return Ui, Si
         }
   }

//1. és 2. típusú kivételek felismerése:
9) If (cycleExist(replace(Ui, G))==TRUE) then {
      a) For every node of Ui {
            i)  X = current node of Ui
            ii) c = current component of G
            iii)if ((pathExists(X, c)==TRUE) and (c∉Ui)) then
```

[13] Azért hívom „majdnem-szomszédossági"-nak ezt a gráfot, mert a szomszédossági gráfhoz képest csak annyi az eltérés, hogy a most indulást kért tranzakció komponensei eredeti formában szerepelnek benne és nem egyetlen csomópontként összevonva.

```
                    add c to Si
            }
        b) Return Ui, Si
    }

//3. típusú kivételek felismerése:
10) Graph preGraph = createPreContiguityGraph(G)
11) If (cycleExists(preGraph)) then {
      a) Return Ui, Si
    }

//Ha tényleg jogos a tiltás:
12) Ui = {}; Si = {}
13) Return Ui, Si
```

A A5 algoritmus először meghívja A2, majd A3 algoritmusokat. Ha mindkettő tiltja a foglalást, akkor előállítja az U lock-kal foglalandó részgráfot (6 és 7 pont) (a 7. pontban alkalmazva a szabad élek és a többszörös bejövő élek elleni védelem miatti részgráf-kiterjesztést), majd ellenőrzi, hogy a korábbi tiltás nem U-U zár ütközés miatt következett-e be (8 pont). Ha nem, akkor lecseréli a gráfban az U lock-kal foglalandó G_i részgráfot egy csomópontra, és megnézi, hogy létrejön-e kör. (9 pont). Ha létrejön kör, akkor megállapítja az S lock-kal foglalandó részt és engedi T_i indulását. Az indulást kért tranzakció részgráfjának csomópontra való lecserélése a `replace` függvény segítségével történik. Ennek a függvénynek a metanyelves leírása:

```
//Lecseréli a G gráfban a Gi részgráfot egyetlen csomópontra.
Graph replace(SubGraph Gi, Graph G)
   1) Graph G2 = {}
   2) New node gi
   3) Add gi to G2
   4) For every node of G {
        a) X = current node of G
        b) If (X∉Gi) then add X to G2
      }
   5) For every edge of G {
        a) X = current edge of G
        b) Node source = X.source()
        c) Node sink = X.sink()
        d) If (source∈Gi and sink∉Gi) then
                add new Edge(gi, sink) to G2
        e) If (source∉Gi and sink∈Gi) then
                add new Edge(source, gi) to G2
        f) If (source∉Gi and sink∉Gi) then
                add new Edge(source, sink) to G2
      }
   6) Return G2
```

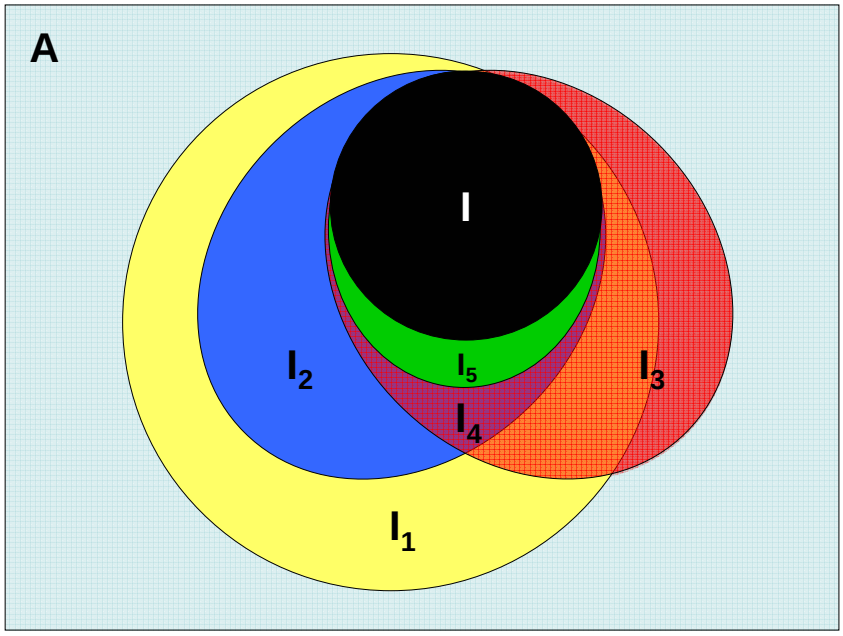
A A5 algoritmus a 10. pontban a 3. típusú eset felismerése miatt létrehozza a „majdnem-szomszédossági gráfot”, a 11. pontban megnézi, hogy van-e benne kör. Ha van, akkor engedi a tranzakció elindulását (hiszen az már nem hozhat létre kört), ha nincs, akkor tiltja az indulást (12 és 13 pontok). A „majdnem-szomszédossági gráf” létrehozása a lenti algoritmussal történik. Ez nagyon hasonló az A3 algoritmusnál már ismertetett szomszédossági gráf létrehozó

algoritmushoz, kivéve, hogy az aktuálisan elbírálás alatt lévő tranzakció által kért részgráfot nem cseréli le csomópontra:

```
// Előállít a G gráfból egy olyan majdnem-szomszédossági gráfot,
// amelyben a most zárolást kért tranzakció részgráfja nem kerül
// lecserélésre, a már zárolt részgráfok viszont igen.
boolean createPreContiguityGraph(Graph G)
   1) Graph G2 = {}
   2) A már zárolt részgráfok mindegyikének megfelelő, egy-egy
      csomópont kerül G2-be
   3) A már zárolt részgráfokon belül futó élek nem kerülnek át G2-be
   4) Ha egy G1 es G2 zárolt részgráf között irányított út fut G-ben,
      akkor a nekik megfelelő V1 es V2 csúcsok között irányított él fut
      G2-ben
   5) A nem zárolt elemek átkerülnek G-ből G2-be
   6) Return G2
```

A A5 algoritmus tehát javítás az A4-hez képest (ld. 5-9. *ábra). Segítségével még több szerkesztési tranzakció engedhető a gráfok helyességének megőrzése mellett. A fent definiált A5 algoritmus a komponens szintű zárakat használó algoritmusok között a lehető legnagyobb párhuzamosságot engedő, nála nagyobb párhuzamosságot biztosító algoritmus nem készíthető*[14]. Ez annak a következménye, hogy az 5.2.1 fejezetben az A4 algoritmus által tiltott összes kategórián végigmentünk, a lehetséges tiltások fájának mindhárom ágát bejártuk. Ez a fa első körben három irányba ágazott el: (1) átfedésben lévő kérések miatti tiltások, (2) részgráf kiterjesztés miatti tiltások, (3) fel nem ismert párhuzamosíthatóság esetei (más szavakkal A4 működéséből adódó problémák). Az első két ágról az 5.2.1.1 és 5.2.1.2 részekben beláttam, hogy csak alacsonyabb szemcsézettségű zárolással feloldhatók. A harmadik ágat viszont egyben nem tudtam kezelni, további bontásra volt szükség. A bontás az 5.2.1.3 részben tárgyalt 4 további ágat eredményezte. Ezek közül egy, a 4. szintén csak alacsonyabb szemcséjű zárolással kezelhető. A másik háromról viszont bemutattam, hogy komponens szintű zárolás esetén is felismerhetők és engedhetők. Az A5 algoritmus ezzel elért az egyféle zárat használó algoritmusok határáig – nála jobbat csak a feltárt ágak engedésével, alacsonyabb zárolás bevezetésével lehetne létrehozni. *A5 ilyen továbbfejlesztése egy olyan A6 algoritmust adna, amely kizárólag az inkonzisztens gráfot eredményező kollaborációkat tiltja, és minden más együttműködést enged: $I_6=I$ (ld.* 5-9. *ábra).* Az A6 algoritmusnak viszont már jelentős ára van: eléréséhez nem elég azt tudni, hogy a felhasználók milyen komponensekkel szeretnének dolgozni, azt is meg kell tőlük kérdezni, hogy melyik komponensen milyen műveleteket akarnak végrehajtani: mit akarnak módosítani, mit akarnak törölni, melyik komponensek között akarnak élt létrehozni. Ez a fajta előregondolkodás ellehetetlenítené a felhasználók együttműködését, a workflow fejlesztése közbeni kreatív tevékenységeket.

[14] Vele azonos párhuzamosságot biztosító, szintén egyetlen felhasználó által is látható zárat használó algoritmus ettől még lehet, hogy készíthető.



Jelmagyarázat:
A: Alaphalmaz. Az összes elképzelhető szerkesztési eset ({*G, T*^{S}} párok).
I: Inkonzisztens gráfot eredményező szerkesztési esetek.
$\mathbf{I_1}$: A1 algoritmus által tiltott szerkesztési esetek.
$\mathbf{I_2}$: A2 algoritmus által tiltott szerkesztési esetek.
$\mathbf{I_3}$: A3 algoritmus által tiltott szerkesztési esetek.
$\mathbf{I_4}$: A4 algoritmus által tiltott szerkesztési esetek.
$\mathbf{I_5}$: A5 algoritmus által tiltott szerkesztési esetek

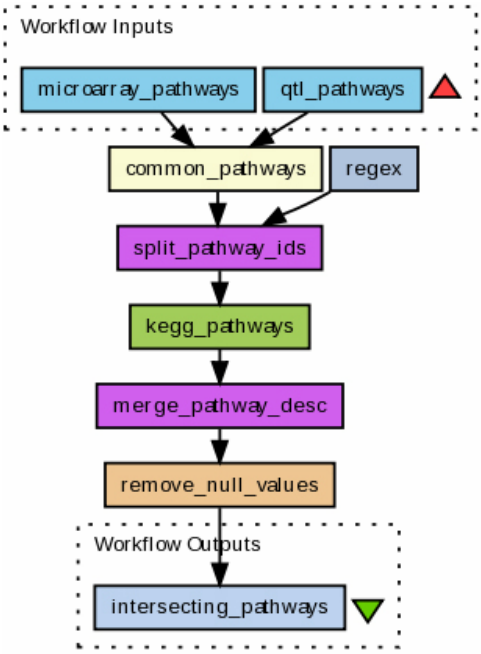
5-9. ábra: Szerkesztési esetek és algoritmusok tiltásainak összevetése 2.

5.3. Gyakorlati példák 2.

Ebben a részben további való életből vett workflow-t és rajtuk futtatható olyan tranzakciósorokat mutatok be azzal a céllal, hogy demonstráljam az eddig tárgyalt algoritmusok egymáshoz képesti erősségeit, és hogy bemutassak olyan szerkesztési eseteket, amelyekben az A5 algoritmus A2, A3 és A4 algoritmusoknál is nagyobb párhuzamosságot enged. Ahogy korábban, most is törekedtem arra, hogy különböző rendszerekből és különböző tudományterületekről származzanak a példák, ezzel is hangsúlyozva a dolgozat eredményeinek általánosságát.

Példa 1: KEGG pathways to QTL and microarray based investigations workflow (Fisher et al, 2007):

- Készítette: Paul Fisher
- Workflow környezet: Taverna
- Célja: A workflow
- A workflow „microarray” és minőségi „trait loci” adatokat kombinál, majd hasonlít össze a szürkeegér génjeiből származó mintákkal. Az egyezések és különbségek alapján tesz következtetéseket a pheno-típusra. A szimulációk a szürkeegér African trypanosomiasis-al szembeni ellenállásának mértékére és módjára adhatnak válaszokat. A fejlesztés során genetikus-biológusok és informatikusok együttműködésére van szükség, mivel a workflow nagy adathalmazokkal dolgozik, párhuzamosított adatfeldolgozást igényel.
- Egy olyan tranzakciósor, amely az 5-8 ábra 1. esetét valósítja meg, és amelyre emiatt A5↖A2 és A5↖A3 és A5↖A4:
  $T^S = T_1, T_2, T_3$
  R_1={common_pathways} *(fehér komponens)*
  R_2={remove_null_values} *(barna komponens)*
  R_3={split_pathway_ids, merge_pathway_desc } *(lila komponensek)*
- Tranzakciósort
  A2-vel elbírálva: [1, 1, 0];
  A3-mal elbírálva: [1, 1, 0] ;
  A4-gyel elbírálva: [1, 1, 0] ;
  A5-tel elbírálva: [1, 1, 1]



5-10. ábra: KEGG pathways to QTL and microarray based investigations workflow Taverna környezetben

Példa 2: Merge Sort Merge Join workflow (Oliveira, 2010):

- Készítette: Daniel de Oliveira
- Workflow környezet: Kepler
- Célja: A workflow a CAPES projekt keretében készített MiningFlow middleware tesztelésére készült. A MiningFlow rendszer workflow menedzser rendszerek szövegekben való adatbányászati képességekkel egészíti ki. A workflow a Kepler workflow rendszer adatbányászattal kiegészített változatának új funkcióit mutatja be.
- Egy olyan tranzakciósor, amely az 5-8 ábra 2. esetét valósítja meg, és amelyre emiatt A5↖A2 és A5↖A3 és A5↖A4:

  $T^S = T_1, T_2$

  R_1={Selection cmd} *(első zöld komponens)*

  R_2={Filter Client, Selection, Merge Join}

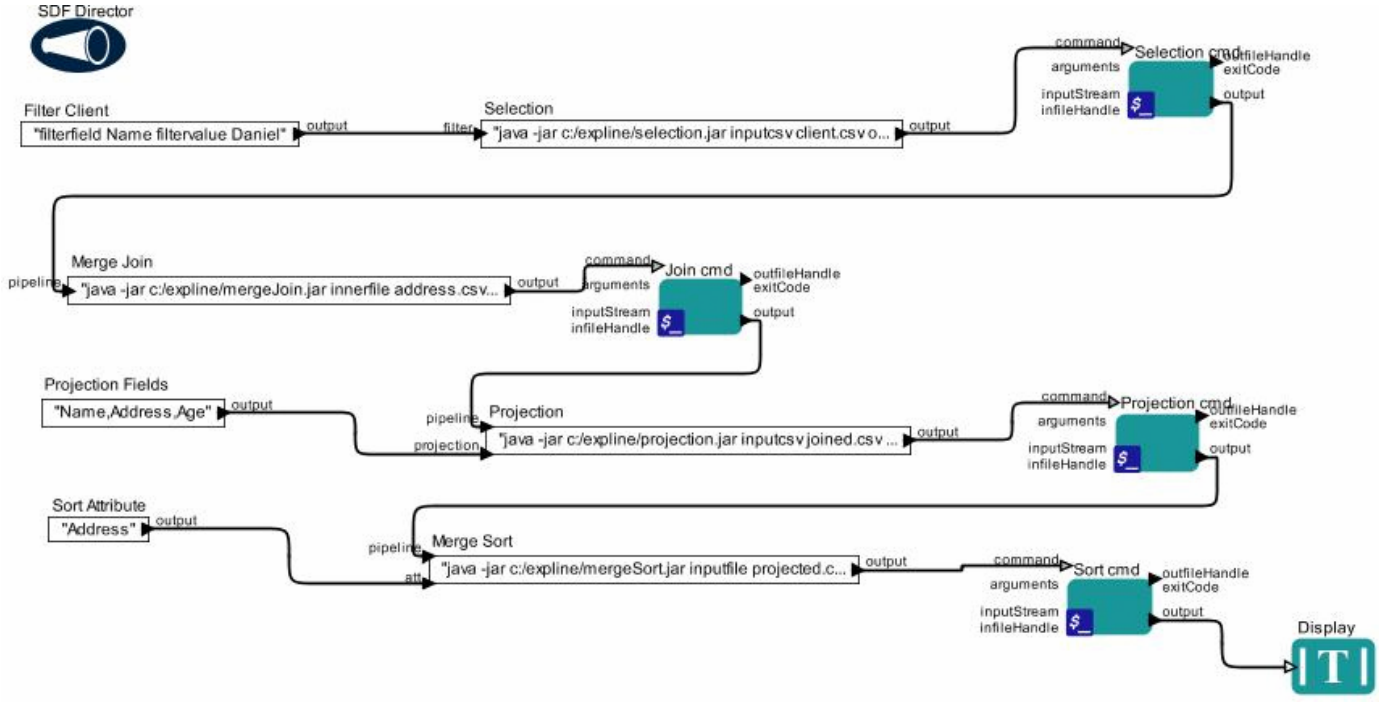
  (1., 2. és 3. fehér komponensek)
- Tranzakciósort

  A2-vel elbírálva: [1, 0];

  A3-mal elbírálva: [1, 0] ;

  A4-gyel elbírálva: [1, 0] ;

  A5-tel elbírálva: [1, 1]



5-11. ábra: Merge Sort Merge Join workflow Kepler környezetben

Példa 3: Molecular Dynamics Simulation by CHARMM (WGRASS, 2010):

- Készítette: Westminster Grid Application Support Service
- Workflow környezet: WS-PGRADE
- Célja: Molekuláris rendszerek dinamikus modellezése lehetséges Newton második törvényét leíró egyenletek atomi szinten való alkalmazásával. A CHARMM szoftver (Chemistry at HARvard Macromolecular Mechanics) ezen atomi modellekre ad implementációt és széles körben használt molekulák és DNS-ek modellezésére. Egy-egy protein molekula vagy DNS leírása nagyszámú, több tízezer atom néhány nanoszekundumos időközönként való szimulálását igényli. A WS-PGRADE környezetben megvalósított CHARMM workflow grid erőforrások bevonásával képes a nagyméretű szimulációs futást számítógépeken szétosztani. A workflow a Westminster egyetem informatikusainak és a bioinformatikusokból álló CHARMM alkalmazói közösség együttműködéseként született.
- Egy olyan tranzakciósor, amely az 5-8 ábra 3. esetét valósítja meg, és amelyre emiatt A5↖A2 és A5↖A3 és A5↖A4:

  $T^S = T_1, T_2$

  R_1={Equilibrium, ana1.1} *(kép közepén)*

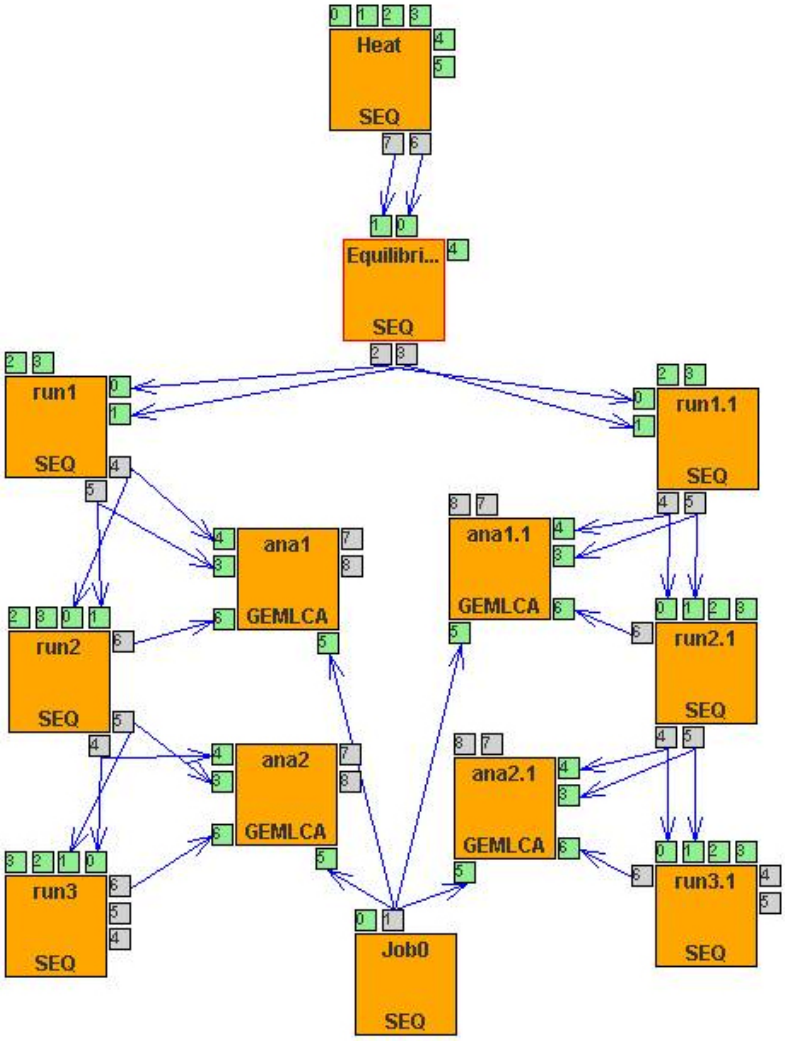
  R_2={run1.1} *(kép jobb oldalán)*
- Tranzakciósort

  A2-vel elbírálva: [1, 0];

  A3-mal elbírálva: [1, 0] ;

  A4-gyel elbírálva: [1, 0] ;

  A5-tel elbírálva: [1, 1]



5-12. ábra: CHARMM workflow WS-PGRADE környezetben

5.4. Alternatív részgráfok felajánlása

Ahogy az előző fejezetben írtam, A5 az A4 algoritmus továbbfejlesztéseként hozható létre úgy, hogy felkészítjük az 1., 2. és 3. néven tárgyalt szerkesztési esetekre. A5 használatakor viszont még mindig maradnak tiltott tranzakciók. Ez olyankor fordul elő ha: (i) a felhasználó már eleve zárolt komponenst kér (5.2.1.1 rész), (ii) ha konzisztenciaőrzés miatt zárolt komponenst kéne kapnia (5.2.1.2 rész), illetve (iii) amikor nem tudható pontosan előre, hogy milyen éleket is akar a szerkesztésre kért komponensek között definiálni. (5.2.1.3/4. eset) Hogy ilyen esetekben se kelljen a tiltott tranzakció tulajdonosának a munkáját teljesen felfüggesztenie amíg a kért komponensek szabaddá válnak, a szerkesztőrendszer felajánlhatna számára *egy olyan részgráfot, amely abban a pillanatban foglalható, és ami a kért komponensek közül a lehető legtöbbet tartalmazza*. Ennek a részgráfnak a zárolásával a felhasználó egyből elkezdhetne dolgozni, és noha nem is tudná az összes tervezett változtatását a gráfon végrehajtani, azért a lehető legtöbb munka elvégezhető. Ennek az éppen foglalható, lehető legnagyobb részgráfnak a kiválasztása a következő algoritmussal történhet:

```
Input:  Ri – the request that A5 tried, but failed to allocate
         sub-graph for.
Output: Ui - the alternative sub-graph that can be locked for Ti with user
         lock. If Ui is empty then no alternative sub-graph can be found.

1) Gi = Ri;
2) For every component of Gi {
      a) C = current component of Gi
      b) If (X.isUserLocked()==TRUE) then Gi.remove(C)
   }
3) For every edge of Gi {
      a) E = current edge of Gi
      b) Node source = E.source()
      c) Node sink = E.sink()
      d) If (source∉Gi or sink∉Gi) then Gi.remove(E)
   }
4) Hi = A5(Gi)
5) If (Hi <> ∅) then Return Hi
6) Cycles[] = identifyCycles( replace(Gi, G) )
7) For every cycle of Cycles[] {
      a) Component[] in = findIncomingComponents(Gi, cycle[i])
      b) Component[] out = findOutgoingComponents(Gi, cycle[i++])
      c) totalIncoming.add(in)
      d) totalOutgoing.add(out)
   }
8) If (totalIncoming > totalOutgoing) then {
      Gi.remove(totalOutgoing) else
      Gi.remove(totalIncoming)
   }
9) Return Gi, Si
```

A tranzakciótiltás leggyakoribb oka, hogy a felhasználó számára olyan komponenst kell zárolni, amely már eleve zárolt (vagy, mert ilyen komponenst kért, vagy ilyennel lett részgráfja kiterjesztve). Emiatt a 2. lépésben ki kell venni a zárolandó részgráfból a már zárolt komponenseket. (Amelyeket az 5.2.1.1 és 5.2.1.2 részekben tárgyaltak szerint csak finomabb zárolással lehetne több ember között megosztani.) Előfordulhat, hogy így olyan csomópontot távolítottunk el, amelyhez kapcsolódó él a zárolandó részgráf része. Az ilyen éleket szintén el kell távolítani az alternatív részgráfból, különben szabad élként maradhatnak hátra. (3. lépés)

Ha az így kapott részgráf lefoglalását már engedi az A5 algoritmus, akkor elértük a végeredményt (4-5. lépés). Ha nem engedi A5 a foglalást, akkor a részgráf további csökkentésére van szükség. Ezen a ponton a tiltás oka biztosan az, hogy a kért részgráf egy, vagy több már futó tranzakcióval együtt kört hoz létre a szomszédossági gráfban, méghozzá úgy, hogy a kör egynél több részgráfon belül megtörik. (5.2.1.3/4 részben tárgyalt eset) A feladat az ilyen kör-kezdeményekben való részvétel megszüntetése.

Az algoritmus a 6. lépésben megkeresi azokat a kör-kezdeményeket, amelyekben a kért részgráf részt vesz. Meg kell szabadulni ugyanis *vagy* azoktól a csomópontoktól, *amelyekbe* a kör-kezdeményből élek futnak, *vagy* azoktól a csomópontoktól, *amelyekből* a kör-kezdeménybe futnak élek. Ezzel a lépéssel biztosítható, hogy a részgráf nem kapcsolódik ki- és bejövő élekkel is a kör többi részéhez és így bármilyen élek is kerülnek majd létrehozásra a részgráfon belül, azok nem tudnak a részgráfon kívüli élekkel együtt kört alkotni. Az algoritmus a 7. lépésben megállapítja, hogy a kimenő, vagy a bejövő komponensek eltávolítása jár-e kevesebb költséggel, és a 8. lépésben a kevesebb komponens kidobását eredményező irányt választja.

Ha az ezután kapott, és végeredményként visszaadott (10. lépés) részgráf nem üres, akkor annak lefoglalását már engedi az A5 algoritmus. (Bizonyítást ld. a fejezet végén.)

A 7. pontban az algoritmus egy-egy függvényt használt, hogy összegyűjtse a totalIncoming halmazban azokat a komponenseket, amelyek bejövő élekkel kapcsolódnak a kör-kezdeményekhez, a totalOutgoing-halmazban pedig azokat, amelyek kimenő élekkel kapcsolódnak a kör-kezdeményekhez. A `findIncomingComponents` függvény megkeresi az első paraméterben kapott részgráfból azokat a komponenseket, amelyek a második paraméterként kapott kör-kezdeményben úgy vesznek részt, hogy belépjük a kör-kezdemény részgráfon kívüli részéből út vezet be:

```
Input:  subGraph – a subgraph in which the components must be found.
Input:  preCycle – a cycle in which subGraph participates in.
Output: inComing – set of components from subGraph which can be reached in
         preCycle from outside of subGraph + edges that are connected to
         these inside subGraph

1) ComponentSet inComing = ∅
2) ComponentSet external = preCycle.remove(subGraph)
3) Node N = (pick any node from external)
4) For every component of subGraph {
      a) Component c = current component of subGraph
      b) If (pathExists(N, c)==TRUE) then inComing.add(c)
   }
5) For (every edge of subGraph) {
      a) e = current edge of subGraph
      b) If (e.source∉subGraph or e.sink∉subGraph) then
         inComing.add(e)
   }
6) Return inComing
```

A függvény először elkülöníti a kör-kezdeményben részt vevő, de a részgráfba nem tartozó komponenseket (2. lépés). Ezek közül kiválaszt egy tetszőleges csomópontot (3. lépés), és minden, a részgráfban lévő komponensre megvizsgálja, hogy elérhető-e ebből a csomópontból. Amelyik elérhető azt elkülöníti (4. lépés). Ebben a 4. lépésben használt `pathExists` függvény ugyanaz, mint amit már az A1 algoritmusban, a 3.5 fejezetben is használtam: megvizsgálja, hogy az első paraméterből kapott csomópontból (esetünkben a részgráfon kívüli csomópontból) elérhető-e a második paraméterként kapott komponens. Az 5. lépésben megnézi, hogy a fennmaradó részgráfban kialakult-e olyan él, amelynek valamelyik végpontja nincs a részgráfban. Erre az ellenőrzésre azért van szükség, mert a 4. pontban lehet, hogy egy csomópont úgy lett kivéve a részgráfból, hogy a hozzá kapcsolódó él ott maradt. Az élek csomópont nélküli foglalása ellentmondana a P1-ben definiált szabálynak. (Ld. 3.4.1 fejezet.)

A `findIncomingComponents` függvényhez hasonló `findOutgoingComponents` függvény a foglalandó részgráf azon komponenseit keresi meg, amelyek kimenő éleken keresztül kör-kezdeményekbe futnak. Az algoritmus az előzőhöz nagyon hasonló, az egyetlen változás, hogy a 4. lépésben a részgráfból induló utat kell keresni az azon kívüli csomópontba.

```
Input:  subGraph – a subgraph in which the components must be found.
Input:  preCycle – a cycle in which subGraph participates in.
Output: outGoing – components of subGraph from which components of
         preCycle can be reached + edges that are connected to these
         inside subGraph
```

```
1) ComponentSet outGoing = ∅
2) ComponentSet external = preCycle.remove(subGraph)
3) Node N = (pick any node from external)
4) For every component of subGraph {
      a) Component c = current component of subGraph
      b) If (pathExists(c, N)==TRUE) then outGoing.add(c)
   }
5) For (every edge of subGraph) {
      c) e = current edge of subGraph
      d) If (e.source∉subGraph or e.sink∉subGraph) then
         outGoing.add(e)
   }
6) Return outGoing
```

Egy konkrét példán keresztül szeretném bemutatni az alternatív, zárolandó részgráfot kereső algoritmus működését. Tegyük fel, hogy a felhasználó az 5-13. ábrán látható gráfban az R_i komponenshalmazt kéri zárolásra: (R_i={*A*, *B*, *C*, *D*, *E*, *F*, *G*, ab_1, ab_2, ab_3, *ac*, *de*, *df*, *bk*}). Az A5-tel való tranzakcióelbírálás során a kért komponensek halmaza a *K* csomóponttal ki lesz terjesztve, és a tranzakció elbírálását az A5 algoritmus tiltja. (A K csomóponttal való kiterjesztésre azért van szükség, mert különben a *bk* él szabad élként maradhat hátra.)

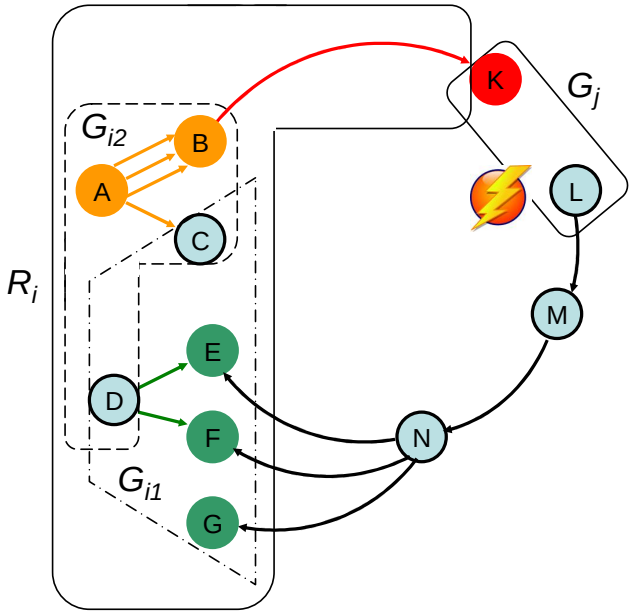
A fenti algoritmus a kiterjesztett komponenshalmazt tekinti lehetséges G_i-nek. Ebből az 1) lépésben kiveszi a G_j-vel átfedésben lévő komponenst, a *K* csomópontot. Mivel ez szabad élként hátrahagyja a *bk* élt, ezért a 2) lépésben ez kerül ki G_i-ből. Mivel az így kapott részgráf lefoglalását sem engedi A5, ezért a 3.a) lépésben meghatározásra kerülnek a gráfban lévő olyan kör-kezdemények amelyekben G_i részt vesz. 3 db ilyen körkezdemény van. (Az elsőbe az *ne* él, a másodikba az *nf* él, a harmadikba az *ng* él tartozik, a kör többi része ugyanaz.)

A G_i részgráf határán (a G_i-nek nem részeként) egy kimenő él, és három bejövő él kapcsolódik ilyen kör-kezdeményekhez: a *bk* él kimenő, az *ne*, *nf*, *ng* élek pedig bejövők. Ekkor két lehetősége van az algoritmusnak: vagy a kimenő él, vagy a bejövő élek felszámolása.

A kimenő él felszámolása magában foglalja a *B* csomópont, és az *A* csomópont G_i-ből való eltávolítását is, hiszen csak a *B* csomópont eltávolítása esetén az A csomópont kerülne a *B* helyére, ő lenne a határvonal a kör-kezdeményben. Az *A* és *B* csomópontok eltávolítása viszont magával vonzza az ab_1, ab_2, ab_3, *ac* élek eltávolítását is. Összesen tehát 6 db komponenst kellene eltávolítani (*A*, *B*, ab_1, ab_2, ab_3, *ac*).

A bejövő élek felszámolása az *E*, *F* és *G* csomópontok eltávolítását igényli. Ez viszont magával vonzza a *de* és *df* élek eltávolítását is, így összesen 5 db komponens eltávolítására van szükség (*E*, *F*, *G*, *de*, *df*). Ha tehát a cél a lehető legtöbb komponens megtartása, akkor a bejövő élektől érdemes megszabadulni, és zárolásra így a G_{i1} részgráf marad.

Elképzelhető olyan eltávolítási politika is, amely az éleket és csomópontokat nem tekinti azonos fontosságúnak. Ez felfogható úgy, hogy egy él eltávolítása *x*, egy csomópont eltávolítása *y* költséget jelent, és *cél a minél alacsonyabb összköltség elérése*. A fent ismertetett, G_{i1}-et eredményező algoritmus a csúcsokat és az éleket is 1 költségűnek veszi. Egy gyakorlatban hasznos másféle súlyozás lehet az élek 0, a csúcsok 1 értékkel való szerepeltetése. Ilyen esetben az élek eltávolítása nem jelent költséget, a cél minél kevesebb csúcs eltávolítása. Egy ilyen politika az 5-13. ábrán látható gráfban a G_{i2} részgráfot eredményezné, mivel ilyenkor érdemesebb lenne a (*A*, *B*, ab_1, ab_2, ab_3, *ac*) komponenseket eltávolítani az (*E*, *F*, *G*, *de*, *df*) halmaz helyett, *az első csoportban ugyanis kevesebb csomópont van, mint a másodikban*.



5-13. ábra: A5 által tiltott kérés helyett felajánlható egy alternatív zárolható részgráf. A súlyozási függvénytől függően vagy a G_{i1}, vagy a G_{i2} részgráf ajánlható fel az R_i zárolási kérést beküldő felhasználó számára.

Állítás: A feni algoritmus minden esetben ad eredményül vagy egy A5-tel zárolható részgráfot, vagy egy üres részgráfot.

Bizonyítás:
Indirekt úton: Tegyük fel hogy az algoritmus ad egy $G_i \neq \varnothing$ részgráfot, amely A5-tel nem foglalható le. Mivel A5 tilt, ezért a következő esetek valamelyike *kell, hogy fennálljon*:

1. $\exists G_j \mid G_i \cap G_j \neq \varnothing$
2. A G gráfban kört alkot a G_i részgráf (csomópontként tekintve) és még legalább egy zárolt részgráf (G_j, szintén csomópontként tekintve), és 0 vagy több zárolatlan csomópont.
   és
   A G_i részgráfon belül megtörik a kör.
   és
   A kör kialakításában G_i mellett résztvevő zárolt részgráfok közül *legalább egy részgráfon belül (G_j) megtörik a kör*.

Az algoritmus az 1) lépésben el kellett hogy távolítson minden már lefoglalt komponenst G_i-ből, ezért az 1. pont feltétele nem lehet igaz.
Az algoritmus a 4) pontban eltávolította vagy azokat a csomópontokat, amelyek bejövő éllel kapcsolódnak a teljes gráfban kialakuló kör-kezdeményezéshez, vagy azokat a csomópontokat, amelyek kimenő élekkel kapcsolódnak ezen kör-kezdeményekhez. Vagyis a 2. eset sem állhat fenn, és így ha az algoritmus nem üres részgráfot eredményez, akkor az a gráf A5-tel foglalható kell hogy legyen.

6. További kutatási feladatok

Kutatási témám folytatására több lehetőség kínálkozik. Egy olyan témát azonosítottam, amely az általam elért eredmények szorosan vett továbbfejlesztését jelenti, továbbá egy olyat, amely tágabb értelemben kapcsolódik a kollaboratív workflow alkalmazásokhoz, és azok futtatására fókuszál.

6.1. Beágyazást tartalmazó workflow-k kollaboratív szerkesztése

Egy beágyazást tartalmazó workflow (angolul *nested workflow*) olyan gráf, melynek egy, vagy több csomópontja szintén gráf. Workflow alkalmazások beágyazása lehetővé teszi az alkalmazáson belül különböző absztrakciós szintek definiálását, melyre tipikusan akkor van szükség, ha túl sok csomópontot vagy élt tartalmaz a munkafolyamat, vagy ha a munkafolyamat eleve egymástól logikailag jól elkülöníthető részekből épül fel.

Számos workflow fejlesztő rendszer ad támogatást beágyazott workflow-k létrehozásához (Pl. Kepler, WS-PGRADE). Ezekben a rendszerekben egy gráf azon csomópontjai, melyek további gráfokat reprezentálnak speciális típusú csomópontként jelennek meg. Az ilyen csomópontba befutó, illetve onnan kimenő élek olyan adatcsatornák, melyek a beágyazott workflow csomópontjaihoz kapcsolódnak. Mind a beágyazó, mind a beágyazott workflow-t szerkeszteni ugyanúgy kell, mint az eddig tárgyalt egyszintű workflow-kat. Egy beágyazott workflow módosítása a beágyazó szinten az adott csomópont paramétereinek megváltoztatását jelenti. A normál workflow-khoz képest egy plusz lépés, hogy itt a beágyazó workflow szabadon maradt adatcsatornáit az őt befoglaló workflow adatcsatornáihoz kell kötni.

Ha az eddig ismertetett kollaboratív fejlesztési koncepciót használnánk beágyazást tartalmazó workflow-kra, akkor ugyan a gráfok konzisztenciája biztosítható, viszont az egyszintű workflow-khoz képest nagyban romlana a kollaborációs teljesítmény. Ennek okai:

- Egy beágyazott workflow-t reprezentáló csomópont módosításához (pl. a komponenshez a külső szinten egy új élt akarunk kapcsolni) zárolni kell az adott komponenst. Hogy a zár megszerezhető legyen, ahhoz meg kell várni, hogy a csomópontba ágyazott teljes gráf-hierarchia felszabaduljon, vagyis hogy senki ne dolgozzon a beágyazott gráfon. Ez nagyon gyenge kollaboratív teljesítményt ad, különösen nagy méretű, vagy több szintű beágyazott gráfok esetén.
- Mivel egy beágyazott workflow szerkesztése a beágyazó workflow szintjén csomópont paraméter módosításként jelenik meg, ezért egy beágyazott workflow módosításához zárolnia kell a beágyazó workflow-ban az azt reprezentáló csomóponton. Ha kizárólagos zárolást használunk, akkor a megszerzés után azt más felhasználó már nem kaphatja meg, vagyis egy beágyazott workflow-n egyszerre csak egy felhasználó dolgozhatna. Ez gyakorlatilag egyfelhasználós szerkesztéssé degradálja a beágyazott workflow-n a kollaborációt.

Ezeknek a problémáknak a kiküszöböléséhez finomabb szemcsézettségű zárolásra van szükség. Azonban ezek a zárak nem azonosak az előző fejezetben ismertetett M és L lock-okkal, ugyanis azok nem képesek hierarchiában működni. További kutatási téma lehet ezeknek az új zártípusoknak és használatuk szabályainak a meghatározása annak érdekében, hogy beágyazott workflow gráfokat is hatékonyan lehessen valós idejű kollaboratív fejlesztőrendszerrel módosítani.

6.2. Kollaboratív workflow alkalmazások végrehajtása

Kollaboratív módon összeállított workflow-k ugyanúgy rendelkeznek futtatási fázissal, mint a normál, egyfelhasználós workflow-k. Előfordulhat, hogy a kollaboratív futtatást már csak egyetlen személy végzi – akár a fejlesztői csoport tagja, akár a fejlesztői körön kívüli személy. Ekkor ügyelni kell arra, hogy a személy rendelkezik a megfelelő szintű hozzáférésekkel a futtatáshoz. Workflow esetén hozzáférés több szinten is jelentkezik (ld. még1-1. ábra): a workflow által meghívott szolgáltatásokban és alkalmazásokban, ezen alkalmazások által használt erőforrásokon és adatbázisokban. Amennyiben a futtató személy nem rendelkezik megfelelő hozzáférési jogosultságokkal, akkor ezt számára valahogy delegálni kell. Ennek a delegációnak a kérdésköre – mikor, milyen jogosultságot, kitől, kinek és hogyan – további kutatásokra ad lehetőséget. Grid workflow alkalmazások kapcsán magam is végeztem a témában kutatást, melyről egy publikáció is született (Sipos et al, 2005c).

Amennyiben egy kollaboratív módon fejlesztett workflow alkalmazás futtatása is kollaboratívan, vagyis egy többfős csoport által vezérelve történik, akkor a futtatás alatt is szükség lehet valamilyen konkurenciakezelési módszer használatára. Konkurenciakezelés nélkül ugyanis előfordulhat, hogy a különböző résztvevők által adott utasítások egymásnak ellentmondó irányítást jelentenek, mely inkonzisztens eredményt, a workflow több irányba elváló nézeteit adja. A helyzet hasonló ahhoz, amit például Mauve (Mauve, 2000) publikációja több résztvevős számítógépes játék kapcsán említ. Ott egy több személy által kormányozható vonatról van szó, amely a váltókkal számos különböző vágányra irányítható. Amennyiben valaki épp egy váltón való áthaladás előtt módosít a váltókon, akkor a vonat más vágányra juthat az ő saját, mint a többi játékos nézetében. Amennyiben ezt az ellentmondást utólag oldjuk fel, akkor a vonat valaki képernyőjén hirtelen egy másik vágányra „repül át”.

Léteznek olyan workflow-k amelyek futás közben adatokat olvasnak be a felhasználóktól, és futásukat ennek megfelelően változtatják, módosítják. A kapott információ alapján például bizonyos csomópontokat kihagyhatnak, több ág közül csak egy megadottat futtatnak, stb. Egy workflow futás közbeni irányítása nagyon hasonlít egy, a fenti példában leírt vonat irányításához, és valószínű, hogy az interaktív hálózati játékokban alkalmazott módszerek itt is használhatóak konkurenciakezelésre. Ennek megvizsgálása jövőbeni kutatási téma lehet.

7. Összefoglalás és az eredmények hasznosíthatósága

Workflow-k kollaboratív módon történő fejlesztése ugyanazokat az általános előnyöket hordozza magában, amelyek más valós idejű csoportmunka környezetre is jellemzőek (Baeza-Yates, Pino, 1997):

- *A végeredmény jobb minőségű lesz*: jobb workflow, több funkcióval, hatékonyabb működéssel, gyorsabb futással, futás közben kevesebb erőforrás-használattal, stb.
- *A fejlesztésben részt vevő személyek egyéni, és csoportos fejlődése*: a workflow fejlesztésben részt vállaló személyek tanulnak a közös munkából, legközelebb egyedül is képesek lesznek hasonlóan magas színvonalú workflow létrehozására. A csoportmunka során ráadásul a többi tagot is jobban megismerik, szociális hálózatuk is megerősödik.
- *A fejlesztésben részt vevők időt és energiát spórolnak:* A kollaboratív környezeten keresztül végzett fejlesztés gyorsabb, ráadásul alacsonyabb erőfeszítést igényel, mint más, egyfelhasználós rendszereken, VCS rendszereken át történő együttműködés.

A tárgyalt zár alapú konkurenciakezelés, és intelligens gráf felosztáson alapuló konzisztenciaőrzés különösen akkor eredményez hatékony munkát, ha a gráf csomópontok létrehozása sok kézzel végzendő módosítást igényel, amikor ezen fejlesztések utólagos kompenzációja, esetleges eldobása nagy idő- és energiakiesést jelent a felhasználóknak. A számítógépes szolgáltatásokból, Web illetve Grid szolgáltatásokból építkező workflow-k szinte kivétel nélkül ide tartoznak. Ezeknél ugyanis egy-egy csomópont bekonfigurálása számos paraméter beállítását, akár új kódok, szolgáltatások fejlesztését is magában foglalja. *A dolgozatban ismertetett módszerekkel a beállított paraméterek, a kifejlesztett kódok workflow-ba kerülése garantált.*

Az irányítatlan gráfként ábrázolható hierarchikus struktúrák (például dokumentumok, vagy XML állományok), CAD dokumentumok, multimédia tartalmak (rajzok, filmek, zenék) kollaboratív szerkesztése eléggé széles körben kutatott terület. Ezzel szemben az irányított gráfokként ábrázolható struktúrák eddig elhanyagolt téma volt csoportmunka alkalmazás fejlesztői körökben. Az értekezés irányított gráfok valós idejű kollaboratív szerkesztőrendszerekkel való szerkesztésének témakörében hozott új eredményeket. Konkrét felhasználási területként a dolgozatban végig dominált a workflow alkalmazások fejlesztése, azon belül is gyakorlati példákon keresztül a Grid workflow alkalmazások részterület. *Az ismertetett eredmények azonban bármely olyan felhasználási körben hasznosnak bizonyulhatnak, ahol a megosztott objektum irányított gráfként ábrázolhatóak, például „mind-mapping" eszközökben, tudásreprezentációs környezetekben, termékfejlesztő eszközökben* (Huang et al, 2007). Az egyre szélesebb körben használt workflow rendszerek mellett tehát ezen további témakörök is profitálhatnak az értekezés eredményeiből.

Az MTA SZTAKI Párhuzamos és Elosztott Rendszerek Laboratóriumának és a University of Reading (Egyesült Királyság) munkatársainak a közreműködésével létrehoztuk a P-GRADE Grid Portál környezet csoportmunkát támogató kiterjesztését (Sipos et al, 2005)(Sipos, Kacsuk, 2006). A P-GRADE Grid Portál nagy adattároló és nagy számítási kapacitást igénylő kalkulációk workflow alapú megfogalmazásához és Grid rendszereken való végrehajtásához nyújt segítséget. A kiterjesztett változatban a felhasználók a kliens gépeken futó interaktív fejlesztőkörnyezetükben kérhetnek gráf részeket zárolásra. A P-GRADE Portál központi szervere a hálózaton keresztül fogadja, majd elbírálja ezeket a kéréseket. A dolgozat 2.4. részében ismertetett zár alapú megoldással a szerver képes felosztani a Grid workflow-kat a

felhasználók között, akik ezután a saját szerkesztőkörnyezetükben módosíthatják saját részgráfjaikat. A környezet képes megjeleníteni, hogy a workflow mely részei milyen felhasználók számára vannak lefoglalva, sőt – amennyiben a felhasználók kívánják – időről időre értesülhetnek az egymás által a workflow-n végzett, de még nem elmentett módosításokról. A rendszer ezáltal segíti a csoportmunka hatékonyságát, a résztvevők csoportba való beilleszkedését.

2010-ben indult „Sharing Interoperable Workflows for Large-Scale Scientific Simulations on Available DCIs" (röviden SHIWA) néven egy kétéves, az EU 7. kutatási keretprogramja által támogatott projekt. A SHIWA projekt célja a P-GRADE Grid Portál kiterjesztésével létrehozni a ma legelterjedtebbnek számító számítási és grides workflow-kat egymásba ágyazni képes rendszert (SHIWA, 2010). A workflow-k ilyen módon történő integrálása lehetővé teszi a workflow nyelvek közötti együttműködést az eddig kifejlesztett workflow-k átírása, illetve új workflow nyelvek kifejlesztése és megtanulása nélkül. A meglévő workflow-kat egyesíteni képes rendszer a már meglévő, és különböző eszközökkel és nyelveken létrehozott workflow-kat egy-egy csomópontként veszi be magasabb szintű „szuper-workflow"-kba, így rejtve el azok belső felépítését. Mivel ezeknek a „szuper-workflow"-knak a létrehozása több, már meglévő workflow ismeretét feltételezi, legideálisabb módon valamilyen csoportmunkát támogató környezetben történhet. Az MTA SZTAKI Párhuzamos és Elosztott Rendszerek Laboratóriuma, a SHIWA projekt vezetője, meg fogja vizsgálni, hogy a dolgozatban tárgyalt kollaboratív workflow szerkesztés lehetősége milyen formában segíthetné a SHIWA workflow fejlesztőrendszert. Ennek első lépése a EuroPar 2010 konferencián publikált cikk (Sipos, Kacsuk, 2010b).

8. Az értekezés témakörében készített saját publikációk

(Kacsuk, Kiss, Sipos, 2008) Péter Kacsuk, Tamás Kiss, Gergely Sipos: Solving the grid interoperability problem by P-GRADE portal at workflow level. Journal of Future Generation Computing Systems, Volume 24, Issue 7, pp. 744 751, July 2008.

(Lewis et al, 2005) Gareth J. Lewis, Gergely Sipos, Florian Urmetzer, Vassil N. Alexandrov, Péter Kacsuk: The Collaborative P-GRADE Grid Portal. Proc. of International Conference on Computational Science (ICCS), LNCS 3516, Atlanta, USA, pp. 367-375, 2005.

(Sipos et al, 2005) Gergely Sipos, Gareth J. Lewis, Péter Kacsuk, Vassil N. Alexandrov: Workflow-oriented Collaborative Grid Portals. Advances in Grid Computing, Proc. of European Grid Conference, EGC 2005, LNCS 3470, Amsterdam, The Netherlands, pp. 434-443, 2005.

(Sipos et al, 2005b) Gergely Sipos, Csaba Németh, Tamás Boczkó, Péter Kacsuk: Providing a Multi-Grid Access Mechanism by the P-GRADE Portal. Proc. of Microcad 2005 International Conference, Miskolc, Hungary, pp. 359-365, 2005.

(Sipos et al, 2005c) Gergely Sipos, Csaba Nemeth, Gareth J. Lewis, Vassil N. Alexandrov, Peter Kacsuk: Executing Workflow-Based Grid Applications with the Collaborative P-GRADE Portal. Proc. of UK e-Science All Hands Meeting, Nottingham, UK, 2005.

(Sipos, Kacsuk, 2005) Gergely Sipos, Péter Kacsuk: Collaborative Workflow Editing in the P-GRADE Portal. Proc. of Microcad 2005 International Conference, Miskolc, Hungary, pp. 353-358, 2005.

(Sipos, Kacsuk, 2005b) Gergely Sipos, Péter Kacsuk: Classification and Implementations of Workflow-Oriented Grid Portals. Proc of The 2005 International Conference on High Performance Computing and Communications, HPCC-05, Sorrento, Italy, pp. 684-693, 2005.

(Sipos, Kacsuk, 2006) Gergely Sipos, Péter Kacsuk: Multi-Grid, Multi-User Workflows in the P-GRADE Portal. Journal of Grid Computing, Volume 3, Issue 3-4, Kluwer Academic Publisher, pp. 221-238, 2006.

(Sipos, Kacsuk, 2009) Gergely Sipos, Péter Kacsuk: Maintaining Consistency Properties of Grid Workflows in Collaborative Editing Systems, Proc. of Eighth International Conference on Grid and Cooperative Computing (GCC09), IEEE-publishing, Lanzhou, China, pp.168-175., 2009.

(Sipos, Kacsuk, 2010) Gergely Sipos, Péter Kacsuk: Efficient Partitioning of Graphs in Collaborative Workflow Editor Systems, Proc. of IADIS International Conference on Collaborative Technologies 2010, Freiburg, Germany, pp. 69-76, 2010.

(Sipos, Kacsuk, 2010b) Gergely Sipos, Péter Kacsuk: Workflow Support for the Effective Collaboration of Grid Application Developers, P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 50–61, 2010.

(Sipos, Kacsuk, 2010c) Sipos Gergely, Kacsuk Péter: Munkafolyam alkalmazások szerkesztésének támogatása csoportmunka módszerekkel, Híradástechnika folyóirat, 2010/5. szám, 21-24. oldal, 2010, június.

9. Irodalomjegyzék

(Aalst et al, 2003) Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Mathias Weske, Business Process Management: A Survey, Lecture Notes in Computer Science, Volume 2678, Springer Berlin, Heidelberg, pp 1-12, 2003.

(Araujo et al, 2002) Araujo, R.M., Santoro, F.M., Borges, M.R.S, The CSCW Lab for groupware evaluation. In: Groupware: design, implementation and use, Proceedings of the 8th International Workshop, CRIWG 2002, La Serena, Chile, September 2002, LNCS 2440, Haake, J., Pino, J. (eds)., pp. 222-231.

(Araujo et al, 2002) Renata Mendes de Araujo, Flavia M. Santoro and Marcos R. S. Borges, The CSCW Lab for Groupware Evaluation, in Groupware: Design, Implementation, and Use, Springer Berlin / Heidelberg, pp. 385-400, 2002.

(Baecker, 1995) Baecker, R.M. et al, Readings in human-computer interaction: toward the year 2000. Morgan Kaufmann Publishers, 1995.

(Baeza-Yates, Pino, 1997) Ricardo Baeza-Yates , Jos, A. Pino, A first step to formally evaluate collaborative work, Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge, pp.56-60, November 16-19, 1997, Phoenix, Arizona, USA

(Baker et al, 1999) D. Baker, D. Georgakopoulos, D. Schuster, A.R. Cassandra, A. Cichocki, Providing customized process and situation awareness in the collaboration management infrastructure, in: Proceedings of the Conference on Cooperative Information Systems, 1999, pp. 79-91.

(Barker, Hemert, 2008) Adam Barker and Jano van Hemert, Scientific Workflow: A Survey and Research Directions, Lecture Notes in Computer Science, Volume 4967, Springer Berlin, Heidelberg, pp. 746-753, 2008.

(Bernstein, Goodman, 1987) Bernstein, P., Goodman, N. and Hadzilacos, V. Concurrency control and recovery in database systems, Addison-Wesley., 1987.

(Bhola et al, 1998) Bhola, S., Mukherjee, B., Doppapaneni, S., Ahamad, M.: Flexible Batching and Consistency Mechanisms for Building Interactive Groupware Applications, proc. of ICDCS'98, Amsterdam, 1998.

(Bollacker et al, 2008) K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In SIGMOD Conference, pages 1247-1250, 2008.

(Cellary et al, 1988) Cellary W., Gelenbe E., Morzy T. Concurrency Control in Distributed Database Systems. North-Holland, Amsterdam, 1988

(Chen, 2006) Chen, D. A Survey of Real-Time Collaborative Editing Systems. Proceedings of the Eighth International Workshop on Collaborative Editing Systems. ACM CSCW 2006, Banff, Canada. November 4, 2006.

(Connolly, Begg, 2004) T. Connolly, C. Begg, Database Systems: A Practical Approach to Design, Implementation and Management, Addison Wesley, 2004.

(Cummings et al, 2008) J. Cummings et al., "Plasma Edge Kinetic-MHD Modeling in Tokamaks Using Kepler Workflow for Code Coupling, Data Management and Visualization", Commun. Comput. Phys., vol. 4(3), pp. 675-702, September 2008.

(Curcin, Ghanem, 2008) Curcin, V., Ghanem, M.: Scientific workflow systems-can one size fit all? In: Proceedings. Cairo International Biomedical Engineering Conference, 2008.

(Deelman et al, 2003) E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. Journal of Grid Computing, 1(1):25-39, 2003.

(Dewan, Riedl, 1993) P. Dewan, J. Riedl, Toward computer-supported concurrent software engineering, In IEEE Computer, vol. 26, no. 1, pp. 17-27., 1993.

(Ehrlich, 1987) Ehrlich, SF., Social and psychological factors influencing the design of office communication systems. Proc. CHI+GI '87 Human Factors in Computing Systems (Toronto, April S-9. 1987), pp. 323-329.

(Ellis et al, 1991) Ellis, C,A., Gibbs, S.J., and Rein, G.L. Groupware: some issues and experiences. Communications of the ACM 34, 1 (January, 1991), 9-28.

(Ellis, Gibbs, 1989) Ellis, C.A., and Gibbs, S.J. Concurrency control in groupware systems. Proc. of the ACM SIGMOD '89 Conference on the Management of Data ACM, New York, pp. 399-407.

(Ereback, Hook, 1994) Ereback A.L. and Hook, K. (1994). Using Cognitive Walkthrough for Evaluating a CSCW Application. Proc ACM CHI'94, pp.91-92.

(Filho, Hirata, 2002) H,lio A. S. Lima Filho, Celso M. Hirata, GroupGraph: A Collaborative Hierarchical Graph Editor Based on the Internet, Proceedings of the 35th Annual Simulatin Symposium, 2002.

(Fisher et al, 2007) Fisher P, Hedeler C, Wolstencroft K, Hulme H, Noyes H, Kemp S, Stevens R, Brass A, A systematic strategy for large-scale analysis of genotype phenotype correlations: identification of candidate genes involved in African trypanosomiasis., Nucleic Acids Research, 20 August 2007, http://nar.oxfordjournals.org/cgi/content/full/35/16/5625

(Foster, Kesselman, 1998) Ian Foster, Carl Kesselman (eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998.

(Fox, Gannon, 2006) Geoffrey C. Fox , Dennis Gannon, Special Issue: Workflow in Grid Systems: Editorials, Concurrency and Computation: Practice & Experience, v.18 n.10, p.1009-1019, August 2006

(Friese et al, 2006) T. Friese, M. Smith, B. Freisleben, J. Reichwald, T. Barth, M. Grauer, Collaborative Grid Process Creation Support in an Engineering Domain, Proc. of the 13th International Conference on High Performance Computing, 2006, IEEE Press

(Fuh, Li, 2004) Fuh, J.Y.H., Li. W. D., 2004, Advances in collaborative CAD: the-state-of-the art, Computer-Aided Design vol 37, issue 5. pp 571-581.

(Fujimoto, 1990) Fujimoto, R.M., Parallel discrete event simulation. Communications of the ACM, 33(10), pp. 31-53, 1990.

(Galli, Lou, 2000) Ricardo Galli , Yuhua Luo, Mu3D: a causal consistency protocol for a collaborative VRML editor, Proceedings of the fifth symposium on Virtual reality modeling language (Web3D-VRML), p.53-62, February 20-24, 2000, Monterey, California, United States

(Georgakopoulos et al, 1995) Diimitrios Georgakopoulos, Mark Hornick, Amit Sheth: An overview of workflow management: From process modeling to workflow automation infrastructure, Journal of Distributed and Parallel Databases, Volume 3, Number 2 / April, 1995, pp. 119-153.

(Goble, De Roure, 2007) Goble C.A., De Roure D.C., 2007, MyExperiment: Social Networking for Workflow-Using E-scientists, Proc. of the 2nd workshop on Workflows in support of large-scale science, Monterey, California, USA. ACM, New York, USA

(Greenberg, Bohnet, 1991) Greenberg, S. and Bohnet, R. GroupSketch: a multi-user sketchpad for geographically distributed small groups. Proc. of Graphics Inte@ace 1991, pp. 207-215.

(Greenberg, Marwood, 1994) Saul Greenberg , David Marwood, Real time groupware as a distributed system: concurrency control and its effect on the interface, Proceedings of the

1994 ACM conference on Computer supported cooperative work, Chapel Hill, North Carolina, United States, pp.207-217, October 22-26, 1994.
(Grudin, 1988) Jonathan Grudin, "Why CSCW applications fail: problems in the design and evaluation of organization of organizational interfaces". Proceedings of the 1988 ACM conference on Computer-supported cooperative work. ACM Press New York, NY, USA. pp. 85-93., 1988.
(Grudin, 1994) Jonathan Grudin. "Computer-Supported Cooperative Work: History and Focus". Computer 27 (5): 19-26., 1994.
(Gu, Yang, Zhang, 2005) Ning Gu , Jiangming Yang , Qiwei Zhang, Consistency maintenance based on the mark & retrace technique in groupware systems, Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work, November 06-09, 2005, Sanibel Island, Florida, USA
(GWF, 2010) Grid Workflow Forum: www.gridworkflow.org Elérés: 2010.02.25.
(Haake, Wilson, 1992) Jörg M. Haake, Brian Wilson, Supporting collaborative writing of hyperdocuments in SEPIA, Proceedings of the 1992 ACM conference on Computer-supported cooperative work, p.138-146, November 01-04, 1992, Toronto, Ontario, Canada
(Haynes et al, 2004) Steven R. Haynes , Sandeep Purao , Amie L. Skattebo, Situating evaluation in scenarios of use, Proceedings of the 2004 ACM conference on Computer supported cooperative work, November 06-10, 2004, Chicago, Illinois, USA.
(Held, Blochinger, 2008) M. Held and W. Blochinger. Collaborative BPEL Design with a Rich Internet Application. Proc. of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid), pages 202-209. IEEE Computer Society, 2008.
(Held, Blochinger, 2009) Held, M., Blochinger, W., 2009, Structured Collaborative Workflow Design, In. Future Generation Computer Systems, Vol. 25, Issue 6, Pages 638-653
(Hill, Gutwin, 2004) J. Hill, C. Gutwin, The MAUI Toolkit: Groupware Widgets for Group Awareness, Journal of Computer Supported Cooperative Work (CSCW), Vol. 13 No. 5-6, pp. 539-571, 2004.
(Howison et al, 2008) Howison, James, Andrea Wiggins, & Kevin Crowston (2008). eResearch workflows for studying free and open source software development. Proceedings of the Fourth International Conference on Open Source Software (IFIP 2.13)
(Huang at al, 2007) Huang, C.J. Liao, L.M., An intelligent agent-based collaborative workflow for inter-enterprise PCB product design, Proc. of the IEEE International Conference on Industrial Engineering and Engineering Management, Singapore, pp. 189-193. 2007.
(Ignat, Norrie, 2008) Claudia-Lavinia Ignat , Moira C. Norrie, Multi-level Editing of Hierarchical Documents, Computer Supported Cooperative Work, vol.17 no.5-6, pp.423-468, 2008.
(Jefferson, 1985) Jefferson, D.R. (1985), Virtual time, A C M Transactions on Programming Languages and Systems, 7(3), pp. 404-425, July.
(Kacsuk et al, 2008) Péter Kacsuk, Krisztián Karóczkai, Gábor Hermann, Gergely Sipos, József Kovács: WS-PGRADE: Supporting parameter sweep applications in workflows. Published by: 3rd Workshop on Workflows in Support of Large-Scale Science, In conjunction with SC 2008, Austin, TX, USA, 2008.
(Kamath et al, 1996) Kamath, M., G. Alonso, R. Guenthor and C. Mohan (1996). Providing High Availability in Very Large Workflow Management Systems. Proceedings of the 5th International Conference on Extending Database Technology, Avignon. pp. 427-442.
(Kammer et al, 2000) P. J. Kammer, G. A. Bolcer, R. N. Taylor, and M. Bergman, Techniques for Supporting Dynamic and Adaptive Workflow, Computer Supported Cooperative Work: The Journal of Collaborative Computing, vol. 9, pp. 269-292, 2000.

(Karsenty, Beaudouin-Lafon, 1993) Karsenty, A. and Beaudouin-Lafon, M. (1993) An algorithm for distributed groupware applications. Proc. of the 13th International Conference on Distributed Computing Systems ICDCS'93, Pittsburgh, May 25-28.
(Khetawat et al, 1997) A. Khetawat, H. Lavana and F. Brglez. Collaborative Workflows: A Paradigm For Distributed Benchmarking and Design on the Internet. NCSU Technical Report, February 1997.
(Kiss et al, 2010) Tamas Kiss, Pamela Greenwell, Hans Heindl, Gabor Terstyanszky and Noam Weingarten: Parameter Sweep Workflows for Modelling Carbohydrate Recognition. Submitted to the Philosophical Transactions of the Royal Society, Series A
(Krajcek, Kiss, Sipos, 2006) Ondrej Krajcek, Tamas Kiss, Gergely Sipos: Building biomedical Grid infrastructure with P-GRADE portal and GEMLCA. Proc. of Coregrid Integration workshop, Integrated research in Grid computing (eds. S. Golratch, M. Bubak), Krakow, pp. 237-248, 2006.
(Kukla et al, 2008) T. Kukla, T. Kiss, G. Terstyanszky, and P. Kacsuk. A general and scalable solution for heterogeneous workflow invocation and nesting. In Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on, pages 1-8, Nov. 2008.
(Lamport, 1978) Lamport, Leslie (1978). "Time, Clocks and the Ordering of Events in a Distributed System", Communications of the ACM, 21(7), 558-565.
(Li, Li, 2005) Li, R. and D. Li (2005): Commutativity-Based Concurrency Control in Groupware. Proc. of the IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'05), pp. 1-10. San Jose, California, USA: IEEE Computer Society, December.
(Lushbough et al, 2008) Lushbough CM, Bergman M.K, Lawrence C J, Jennewein D, Brendel V, BioExtract Server - An Integrated Workflow-Enabling System to Access and Analyze Heterogeneous, Distributed Biomolecular Data, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 11 September 2008, http://doi.ieeecomputersociety.org/10.1109/TCBB.2008.98, Accessed at: 28 October 2008
(Mauve, 2000) Martin Mauve: Consistency in Replicated Continuous Interactive Media, Proc. of the ACM Conference on Computer Supported Cooperative Work (CSCW) 2000, Philadelphia, PA, USA, 2000, pp. 181-190.
(Mecella et al, 2001) M. Mecella, B. Pernici, M. Rossi, and A. Testi, A Repository of Workflow Components for Cooperative e-Applications, Proc. of the 1st IFIP TC8 Working Conference on e-Commerce/e-Business, Salzburg, Austria, 2001.
(Medina-Mora et al, 1993) R. Medina-Mora, T. Winograd, and R. Flores, ActionWorkflow as the Enterprise Integration Technology, Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, Vol. 16, No.2, June, 1993.
(Meilin et al, 1998) Shi Meilin, Yang Guangxin, Xiang Yong, Wu Shangguang, Workflow Management Systems: A Survey, Proceedings of IEEE Intl Conf on Communication Technology, Beijing, 1998.
(Müller et al, 2006) Müller, J. M., Zhang, G., Lapayre, J.C., Müller, P., 2006, Service-oriented Support of Cooperative Workflows. Considerations for Urban Planning Processes, Proc. of 11th Conference of the Association for Information And Management, Luxemburg
(Monk et al, 1996) Monk, A., McCarthy, J., Watts, L., Daly-Jones, O., 1996, Measures of Process. In: Thomas, P.J. (ed.), CSCW Requirements and Evaluation, Berlin, Springer.
(Munson, Dewan, 1996) Jonathan Munson, Prasun Dewan, A concurrency control framework for collaborative systems, Proceedings of the 1996 ACM conference on Computer supported cooperative work, p.278-287, November 16-20, 1996, Boston, Massachusetts, United States
(Nam, Wright, 2001) Nam T. K., Wright D, (2001) The development and evaluation of Syco3D: a real-time collaborative 3D CAD system, Design Studies Vol 22, Issue6, pp 557-582.

(Neale et al, 2004) Neale, D.C., Carroll, J.M., Rosson, M.B., 2004. Evaluating Computer-Supported Cooperative Work: Models and Frameworks. Proc. of CSCW 2004: Conference on Computer-Supported Cooperative Work. (Chicago, November 8-10). ACM Press, New York, pp. 368-377.

(Newman-Wolfe et al, 1992) R. E. Newman-Wolfe , M. L. Webb , M. Montes, Implicit locking in the ensemble concurrent object-oriented graphics editor, Proceedings of the 1992 ACM conference on Computer-supported cooperative work, p.265-272, November 01-04, 1992, Toronto, Ontario, Canada

(Nichols et al, 1995) Nichols, D.A., P. Curtis, M. Dixon and J. Lamping (1995) High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System. Proc. of the ACM Symposium on User Interface Software and Technology, pp. 111-120.

(Noel, Robert, 2003) Noel, S., and Robert, J. M. How the Web is used to support collaborative writing. Behaviour & Information Technology, 22 (2003) 245-262.

(Oinn et al, 2004) T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. Bioinformatics Journal, 20(17):3045-3054, 2004.

(Okada, Tanaka, 1995) Okada, Y. and Tanaka, Y. : IntelligentBox: A Constructive Visual Software Development System for Interactive 3D Graphic Applications, Proc. of Computer Animation '95, IEEE Computer Society Press, pp.114-125,1995.

(Oliveira, 2010) Daniel de Oliveira: Merge Sort Merge Join workflow, http://www.myexperiment.org/workflows/743, Elérés: 2010.03.09.

(Oster et al, 2006) Oster, G., Urso, P., Molli, P., Imine, A.: Data consistency for p2p collaborative editing. In: Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work, CSCW 2006, Banff, Alberta, Canada, November 4-8, 2006, ACM (2006)

(Pedersen et al, 1993) Pedersen, E. R., McCall, K., Moran T. P., Halasz, F. G. (1993). Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings. Proc. of Human Factors in Computing Systems (InterCHI 93) ACM Press, pp. 391-398.

(Phung et al, 2009) Hoang M. Phung, Doan B. Hoang, Elaine Lawrence, A Novel Collaborative Grid Framework for Distributed Healthcare, Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), Shanghai, China, pp. 514-519, 2009.

(Pichiliani et al, 2009) Mauro C. Pichiliani, Celso M. Hirata, Fabricio S. Soares, Carlos H. Q. Forster, TeleEye: An Awareness Widget for Providing the Focus of Attention in Collaborative Editing Systems, in Collaborative Computing: Networking, Applications and Worksharing, Springer Berlin Heidelberg, pp. 258-270, 2009.

(Pinelle, 2000) Pinelle, D. A Survey of Groupware Evaluations in CSCW Proceedings. Technical Report HCI-TR-2000-01, Computer Science Department, University of Saskatchewan. 2000.

(Pinelle, Gutwin, 2000) Pinelle, D., and Gutwin, C., 2000, A Review of Groupware Evaluations, Proceedings of Ninth IEEE WETICE 2000 Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Gaithersburg, Maryland, pp. 86-91, 2000.

(Polson et al, 1992) Polson, P.G., Lewis, C., Riemann, J., Wharton, C., Cognitive walkthroughs: a method for theory-based evaluation of user interfaces, International Journal of Man-Machine Studies, 36, Academic Press, 1992, pp. 741-773.

(Prakash, Knister, 1992) Prakash, A. and Knister, M.J. (1992) Undoing Actions in Collaborative Work. Proc. of the ACM CSCW Conference on Computer-Supported Cooperative Work, Toronto, Nov 1-4, pp. 273-280.

(Preston, 2007) Preston, J. A., Rethinking consistency management in real-time collaborative editing systems. PhD thesis, Georgia State University, 2007.

(Ressel et al, 1996) Ressel, M., D. Nitsche-Ruhland and R. Gunzenhäuser (1996): An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors. Proc. of the ACM Conference on Computer-Supported Cooperative Work (CSCW'96), pp. 288-297. Boston, Massachusetts, USA: ACM Press, November.

(Ryall et al, 1997) Kathy Ryall, Joe Marks, Stuart Shieber, An interactive constraint-based system for drawing graphs, Proceedings of the 10th annual ACM symposium on User interface software and technology, p.97-104, Banff, Alberta, Canada, October 14-17, 1997.

(Sarin, Greif, 1985) Sarin, S. and Greif, 1. Computer-based real-time conferencing systems. Computer 18, 10 (October 1985), 33-45.

(Schmidt, 2006) D.C. Schmidt: Model-Driven Engineering. IEEE Computer vol. 39 issue 2, pp. 25-31. February, 2006.

(SHIWA, 2010) Sharing onteroperable Workflows for Large-Scale Scientific Simmulations on Available DCIs (SHIWA), Project in European Commission, 7th Research Framework Programme, 2010.

(Sipos et al, 2005) Gergely Sipos, Csaba Nemeth, Gareth J. Lewis, Vassil N. Alexandrov, Peter Kacsuk: Executing Workflow Based Grid Applications with the Collaborative P-GRADE Portal. Proc. of UK e-Science All Hands Meeting, Nottingham, UK, 2005.

(Skaf-Molli et al, 2003) Skaf-Molli, H., Molli, P. and Oster, G.: Semantic Consistency for collaborative systems. In: Fifth International Workshop on Collaborative Editing – ECSCW'2003, Helsinki (2003)

(Skouteris et al, 2008) Skouteris, Dimitrios; Costantini, Alessandro; Lagan , Antonio; Sipos, Gergely; Balaskó, Ákos; Kacsuk, P,ter: Implementation of the ABC quantum mechanical reactive scattering program on the EGEE Grid platform, Proc. of International Conference on Computer Science and its Applications (ICCSA'08), LNCS 5072, Peruggia, Italy, pp. 1108-1120., 2008.

(Smashingmagazine, 2008) http://www.smashingmagazine.com/2008/09/18/the-top-7-open-source-version-control-systems/, Elérés: 2009.11.12

(Stetik et al, 1987) Stetik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tatar, D. WYSYIWIS revised: early experiences with multiuser interfaces. ACM Transactions on Office Information Systems 5,2 (April 1987), 147-167.

(Sudholt et al, 2006) W. Sudholt, I. Altintas, K.K. Baldridge, A Scientific Workflow Infrastructure for Computational Chemistry on the Grid, 1st Int. Workshop on Computational Chemistry and Its Application in e-Science in conjunction with ICCS (2006)

(Suleiman et al, 1997) Suleiman, M., M. Cart and J. Ferri, (1997): Serialization of Concurrent Operations in a Distributed Collaborative Environment. Proc. of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP '97), pp. 435-445. Phoenix, Arizona, USA: ACM Press.

(Sun et al, 1998) Chengzheng Sun , Xiaohua Jia , Yanchun Zhang , Yun Yang , David Chen, Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems, ACM Transactions on Computer-Human Interaction (TOCHI), vol.5 no.1, pp.63-108, 1998.

(Sun et al, 2006) Chengzheng Sun, Steven Xia, David Sun, David Chen. Haifeng Shen, Wentong Cai: "Transparent adaptation of single-user applications for multi-user real-time collaboration," ACM Transactions on Computer-Human Interaction, Vol. 13, No.4, December 2006, pp.531-582.

(Sun, 2002) Chengzheng Sun: Optional and Responsive Fine-Grain Locking in Internet-Based Collaborative Systems, IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 9, pp. 994-1008., 2002.

(Sun, Chen, 2002) C. Sun and D. Chen. Consistency maintenance in real-time collaborative graphics editing systems. ACM Trans. On Computer-Human Interaction, vol. 9 no. 1, pp 1-41, March 2002.
(Sun, Ellis, 1998.) Chengzheng Sun and Clarence Ellis, Operational transformation in real-time group editors: issues, algorithms, and achievements, Proceedings of the 1998 ACM conference on Computer supported cooperative work, Seattle, Washington, United States, pp. 59-68., 1998.
(Sun, Sun, 2006) Sun, D. and C. Sun (2006): Operation Context and Context-Based Operational Transformation. Proc. of the 2006 20th anniversary conference on Computer supported cooperative work (CSCW'06), pp. 279-288. New York, NY, USA: ACM Press.
(Taylor et al, 2006) Ian J. Taylor, Ewa Deelman, Dennis B. Gannon and Matthew Shields, Workflows for E-Science: Scientific Workflows for Grids, Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006).
(Terveen, Wroblewski, 1990) Terveen, L. G. & Wroblewski, D. A. (1990). A collaborative interface for editing large knowledge bases. Proc. of the Eighth National Conference on Articial Intelligence, AAAI-90, pp. 491-496.
(Vasko, Dustdar, 2009) M. Vasko and S. Dustdar, Introducing Collaborative Service Mashup Design, in Lightweight Integration on the Web (ComposableWeb'09. CEUR - Workshop Proceedings, June 2009, pp. 51-62.
(Vidot et al, 2000) Vidot, N., M. Cart, J. Ferri, and M. Suleiman (2000): Copies Convergence in a Distributed Real-Time Collaborative Environment. Proc. of ACM Conference on Computer Supported Cooperative Work (CSCW'00), pp. 171-180.
(WfMC, 1993) Workflow Management Coalition. http://www.wfmc.org/, Elérés: 2010.02.24.
(WfRM, 1995) Workflow Management Coalition: Workflow Reference Model. http://www.wfmc.org/standards/docs/tc003v11.pdf, Elérés: 2010.02.24.
(WGRASS, 2010) WGRASS - Westminster Grid Application Support Service: Molecular Dynamics Simulation by CHARMM, http://wgrass.wmin.ac.uk/index.php/W-Grass:charmm, Elérés: 2010.03.09.
(Wilson, 1991) Wilson, P. Computer Supported Cooperative Work: An Introduction. Kluwer Academic Pub., 1991.
(Winograd, 2004) Terry Winograd, Categories, disciplines, and social coordination, Journal of Computer Supported Cooperative Work (CSCW), vol. 2, no. 3, pp. 191-197, Springer, 2004.
(Workflow, 2010) Workflow sz¢cikk a Wikipedia-ban: http://en.wikipedia.org/wiki/Workflow Elérés: 2010.04.30.
(Xue, Orgun, 2005) Xue, L., Orgun, M.: Locking without requesting a lock: A consistency maintenance mechanism in Internet-based real-time group editors, Journal of Parallel and Distributed Computing, 65, pp. 801-814, 2005.
(Xue, Orgun, Zhang, 2002) Xue, L., Orgun, M., Zhang, K.: A User-Centred Consistency Model in Real-Time Collaborative Systems, In: J. Plaice et al (eds.): DCW 2002, LNCS 2468, pp. 138-150, 2002.
(Yang et al, 2000) Y. Yang, C. Sun, Y. Zhang, and X. Jia, Real-Time Cooperative Editing on the Internet, IEEE Internet Computing, pp. 18-25, May/June 2000.
(Yang, Li, 2004) Y. Yang and D. Li. Separating Data and Control: Support for Adaptable Consistency Protocols in Collaborative Systems. Proceedings of CSCW'04. Chicago, IL. November 2004. CHI Letters. Vol 6, Issue 3. pp. 11-20.
(Yang, Li, 2005) Yang, Y.,Li, D., 2005. Supporting Adaptable Consistency Control in Structured Collaborative Workspaces. Computer Supported Cooperative Work, 14, 469-503.
(Yilmaz et al, 2001) Yilmaz, O., I. Tanir, and C. Gregory, 2001, A unified 3-D seismic workflow, Geophysics, 66, pp. 1699-1713.

(Yu, Buyya, 2005) J. Yu, R. Buyya, A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing, Vol. 3, No. 3-4, pp. 171-200. 2005.
(Zaffer et al. 2001) Zaffer, A.A., C.A. Shaffer, R.W. Ehrich and M. Perez (2001): Netedit: A Collaborative Editor. Technical report, January.
(Zhao et al, 2006) Zhao, Z., Booms, S., Belloum, A., de Laat, C., Hertzberger, B.: Vle-wfbus: a scientific workflow bus for multi e-science domains. In: E-science 2006, 2nd IEEE International Conference on e-Science and Grid Computing, Amsterdam, Netherlands (2006)