Eötvös Loránd Tudományegyetem
Informatikai Kar
Programozási Nyelvek és Fordítóprogramok Tanszék

Egy ágens alapú Grid ütemező szimulációja

Nagyprogram dokumentáció

Készítette:
Kós Csaba
KOCHAAT.ELTE
programtervező matematikus szak
nappali tagozat

Témavezető:
Dr. Horváth Zoltán

Budapest, 2006. június 5.

Tartalomjegyzék

Bevezetés

Jelen dokumentum Kós Csaba programtervező matematikus hallgató által az Eötvös Loránd Tudományegyetem Informatika Karán, a 2005/2006-os tanév II. félévében írt nagyprogram leírását tartalmazza.

A nagyprogram kitűzött feladata komplex: egy meglévő Grid rendszer szimulációs eszköz, az **OptorSim** [Opt05] új elemekkel való kiterjesztése, és egy – ezekkel együttműködő – új ütemező (Resource Broker) algoritmus szimulálása és teljesítményének tesztelése.

A Grid rendszerek (vagyis a világméretű, számítógépfürtök összekapcsolásával keletkező nagy számítási kapacitású számítógépes rendszerek, vagy röviden: *Grid-ek*) technológiái a párhuzamos programozás egy igen jelentős, aktívan kutatott ágát képezik. Gyakran jelennek meg Grid-ekkel kapcsolatos új ötletek, algoritmusok, eszközök, amelyekkel meglévő erőforrásainkat (számítási kapacitás, lemezterület, stb.) bizonyos szempontokból egyre jobban és könnyebben ki tudjuk használni. Az új algoritmusok előnyös tulajdonságait, jogosságát valamilyen módon alá kell támasztani, aminek egy lehetséges módja a szimuláció. Az **OptorSim** egy szimulációs eszköz, amit fájl-replikációs és ütemező algoritmusok szimulálására hoztak létre. Feladatom a [LUKH06]-ban publikált ágens-alapú replikációs eljárás, és az ehhez kapcsolódó ütemező algoritmus szimulációja az **OptorSim** kibővítésével.

1. fejezet

Felhasználói dokumentáció

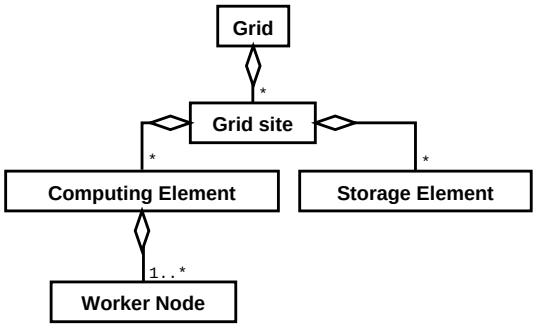
1.1. Bevezetés

Ez a termék egy szimulátor, amelynek segítségével Adat Grid rendszerek fájl-replikációs stratégiái és különböző elvű ütemezőinek (Resource Broker-einek) működése szimulálható. A felhasználó által megadott konfigurációt (Grid felépítést, feladatokat, stb.) betöltve a program szimulálja a Grid működését feladatok (*job-ok*) „végrehajtásával", és a futás végén statisztikákat szolgáltat, amelyeket opcionálisan a szimuláció alatt grafikus felületen vizualizálni is lehet. A szimuláció eredményei alapján a vizsgált algoritmusokat különböző szempontok szerint össze lehet hasonlítani, tulajdonságaikat lehet vizsgálni.

A termék alapjául egy meglévő Adat Grid szimulátor, az `OptorSim` szolgált, ami ki lett egészítve új ütemező eljárásokkal, valamint ágens alapú intelligens fájl-replikációs sémákkal. A program ezen új komponensek teljesítményének mérésére is lehetőséget ad.

A dokumentáció során feltételezzük, hogy a Felhasználó ismeri a Grid rendszerek legalapvetőbb fogalmait.

A Grid rendszerek. Egy *Adat Grid* rendszerben a felhasználó beküld egy *feladatot* (vagy *job-ot*) a Grid-be. A feladatok fájlokon dolgoznak, amik valószínűleg különböző *Grid site*-okon találhatók, és számítási erőforrásokat használva a fájokból beolvasott adatokkal számításokat végeznek. A Grid-nek ütemezési és replikációs döntéseket kell hoznia minden beküldött feladatra a Grid aktuális állapota alapján



1.1. ábra. *A Grid-rendszerek felépítése*

(az adatok elhelyezkedése, a hálózat terheltsége, a számítást végző erőforrások terheltsége, stb.) ahhoz, hogy minimalizája a feladat terhelését az erőforrásokra nézve, és biztosítsa, hogy a feladatok végrehajtása optimális legyen valamilyen értelemben (pl. a lehető legkorábban érjenek véget).

Egy Adat Grid általánosan a következő elemek segítségével építhető fel, amely felépítést az `OptorSim` is követ:

Grid site. A Grid ún. *site*-okra van osztva, ami a benne szereplő csomópontok logikai egysége (általában a csomópontok földrajzi és/vagy adminisztratív elosztása alapján alakulnak ki).

Computing Element. A Grid site-ok kétféle csomópontot tartalmazhatnak. Egyikük a *Computing Element* (röviden: *CE*), amely a feladatok futtatását, a feladatokban megszabott számításokat végzi. A Computing Element a valóságban egy számítógépfürtnek felel meg, és dolgozókból, **Worker Node**-okból épül fel, amelyek az individuális számítógépeknek felelnek meg. Egy Grid site-ban tetszőleges számú Computing Element lehet, bár az `OptorSim`-ben van erre vonatkozólag egy maximális korlátozás: site-onként legfeljebb egy Computing Element lehet. Egy Computing Element tetszőleges pozitív egész számú Worker Node-ból állhat.

Storage Element. A csomópontok másik fajtája a *Storage Element* (*SE*). Ezek az adatok, fájlok tárolásáért felelős eszközök. Számuk szintén tetszőleges lehet egy Grid site-on belül, `OptorSim`-ben legfeljebb egy.

Az olyan Grid site-okat, amelyek nem tartalmaznak sem Computing, sem Storage Element-eket, *forgalomirányítóknak* nevezzük.

Fájl-replikációs stratégiák. A fájl-replikáció egy fontos technika az erőforrások kihasználtságának javítására, és a teljes Grid hatékonyságának növelésére. Bármely Grid-fájlnak lehetnek másolatai (replikái) a Grid különböző pontjain, így elérhető például az, hogy az input fájlok közel legyenek magát a feladatot futtató géphez. A feladatok ugyanis általában a fájlok *logikai nevét* (LFN) használják azok azonosítására, és egy logikai fájlnévhez több *fizikai fájlnév* (PFN) tartozhat. A logikai fájlnév egy absztrakt azonosítója a fájlnak, független a fájl helyétől és a replikák számától. A fizikai név egy konkrét replikát azonosít egy-egyértelmű módon, és ezzel megadja a replika helyét a Grid-ben. Az LFN→PFN fordítás elvégzése a *Replika Menedzser* feladata a *Replika Katalógus* segítségével.

A replikák dinamikus létrehozása és törlése egy fontos része az optimalizálási folyamatnak. A szabályt, amely megadja, hogy mikor és hol jöjjenek létre új replikák, és mikor törlődjenek régiek, *replikációs startégiának* nevezzük.

Ütemezők (Resource Broker). Minden, a Grid-be beküldött feladathoz el kell dönteni, hogy melyik Computing Element hajtsa végre azt. Az *ütemező* vagy *Resource Broker* feladata ennek a bonyolult döntésnek a meghozása.

Ez a termék az EU DataGrid (`http://www.eu-datagrid.org/`) által fejlesztett szoftverrészeket tartalmaz.

1.1.1. Rendszerkövetelmények

A termék Java platformon fut, használatához legalább **1.4**-es verziójú **Java™ 2** futtatókörnyezet (JRE) szükséges. A mindenkori legfrissebb Java futtatókörnyezet ingyenesen letölthető a `http://java.sun.com/` oldalról, de az aktuális verzió telepítőjét a CD `Egyeb/Java/` könyvtárában is elhelyeztük. A program megelégszik egy`Pentium II` szintű processzorral, `64 MiB`[1] memóriával, és `10 MiB` lemezterülettel, mindazonáltal ajánlott a `Pentium III` szintű processzor és `256 MiB` memória.

1.1.2. Támogatott operációs rendszerek

A szimulátor minden olyan operációs rendszeren használható, amihez létezik **Java™ 2**-t támogató futtatókörnyezet (JRE)[2], valamint képes a hosszú fájlnevek kezelésére. A fejlesztőknek szánt forráskód megtekintéséhez valamilyen `.tar`-t és `.gz`-t (`gzip`-et) kicsomagolni képes segédprogram szükséges, fordításához **Java JDK**. Az alábbi operációs rendszerek alatt a program biztosan működik:

- Microsoft Windows XP, 2000, 2003, 98 (SE)
- Linux, BSD, (Open)Solaris alapú operációs rendszerek
- Mac OS X

1.2. A program telepítése

A szimulátor telepítése

A szimulátor CD-n érkezik. A CD-n a `Binaris/` könyvtár tartalmazza a futtatható `OptorSim.jar` állományt, illetve a példa konfigurációs fájlokat (`examples/`), amikből a program a beállításait beolvassa. Mivel a CD csak olvasható, érdemes ezt a könyvtárat a merevlemezre egy könnyen elérhető helyre másolni. Ez a művelet pl. a Windows Intézőben (vagy egyéb grafikus fájlkezelő programban, pl. Nautilusban) a mappa áthúzásával a kívánt helyre tehető meg.

A program semmilyen egyéb telepítést nem igényel, mindazonáltal a konfigurációs állományokat minden bizonnyal testre kívánja majd szabni a Felhasználó. A lehetőségekről az 1.3.1 részben lehet tájékozódni.

[1] 1 MiB = 2^{20} bájt = 1,048,576 bájt (azaz *egy mebibájt*, ld. az IEC 60027-2 szabványt)

[2] A Java futtatókörnyezet telepítéséhez a legtöbb operációs rendszer alatt adminisztrátori jogosultságok szükségesek.

A forráskód telepítése

A szimulátor program használatához nem szükséges a forráskód telepítése. Igény esetén azonban a forráskódot meg lehet tekinteni, vagy akár módosítani, és újrafordítani is szabad. Ezekhez a műveletekhez először ki kell csomagolni azt. A forráskód a CD-n a `Forraskod/optosim-ckos-source.tar.gz` tömörített fájlban található meg. A kitömörítés Windows alatt pl. a 7-Zip nevű, a `http://www.7-zip.org/` oldalról ingyenesen letölthető programmal tehető meg, UNIX-szerű operációs rendszerek esetén pedig a

```
$ tar xfz optorsim-ckos-source.tar.gz
```

paranccsal (a merevlemezre másolt könyvtárból indítva).

1.3. A program használata

A programnak két működési módja van: szöveges és grafikus. Azt, hogy melyik működési módban fusson, és minden egyéb paramétert (beleértve a szimulálandó Grid és feladatok jellemzőit is) állítani a konfigurációs állományokban lehet.

A programot a parancssorból futtatni a program könyvtárából a

```
java -jar OptorSim.jar [paraméter_fájl]
```

parancs kiadásával lehet, ahol a `paraméter_fájl` opcionális argumentum a szimulációs paramétereket leíró konfigurációs állomány elérési útja (ld. alább). Ha nincs megadva, alapértelmezésként az `examples/parameteres.conf` kerül beolvasásra.

A futás végén a szimuláció eredményeit, a statisztikákat az alkalmazás a *standard output*-ra nyomtatja, érdemes ezt fájlba irányítani:

```
java -jar OptorSim.jar [paraméter_fájl] > stats.txt
```

1.3.1. A konfigurációs állományok

Ahogyan az már említve lett, az `OptorSim` az összes futást befolyásoló paramétert konfigurációs állományokból olvassa be. A programkönyvtár `examples` alkönyvtárában elhelyeztünk minta konfigurációs állományokat.

Egy futtatás összes paraméterét az alábbi három állomány írja le:

Grid konfigurációs állomány. Ez a fájl a Grid-ben szereplő erőforrások állapotát írja le, és megadja a Grid felépítését (tehát a csomópontok jellemzőit és kapcsolódását).

A fájl minden sora egy *Grid site*-ot jellemez:

- Az **első oszlop** megadja, hány Worker Node szerepel a site-on lévő Computing Element-ben (feltesszük, hogy egy site-on egyetlen Computing Element van).
- A **második oszlop** a Storage Element-ek száma a site-on. (Egy vagy nulla.)

```
0 1 10000    0    0 1000    0    0    0    0    0    0    0
0 1 10000    0    0 1000    0    0    0    0    0    0    0
1 1 10000 1000 1000    0 1000 1000    0    0    0    0    0
0 1 10000    0    0 1000    0    0    0 1000    0    0    0
0 1 10000    0    0 1000    0    0    0    0    0    0    0
0 1 10000    0    0    0    0    0    0    0 1000    0    0
0 1 10000    0    0    0 1000    0    0    0 1000    0    0
1 1 10000    0    0    0    0    0 1000 1000    0 1000 1000
0 1 10000    0    0    0    0    0    0    0 1000    0    0
0 1 10000    0    0    0    0    0    0    0 1000    0    0
```

1.2. ábra. *Példa Grid konfigurációs állomány.* Az első sor például azt írja le, hogy az első Grid site-on nincs Computing Element, egyetlen Storage Element van 10000MiB kapacitással, és ez a Grid site csak a harmadik Grid site-tal van 1000Mib/sec-os összeköttetésben.

- A **harmadik oszlop** megadja a Storage Element méretét MiB-ban, ha van.
- A táblázat **további oszlopai** egy szimmetrikus site-site mátrixot alkotnak, ahol az i-edik sor és a táblázat ezen részének j-edik oszlopa által kijelölt elem (ami nemnegatív valós szám) megadja az i-edik és j-edik site közötti kapcsolat sebességét Mib/sec-ban (mebibit/másodperc). A 0 érték azt jelenti, hogy nincs összeköttetés a két site között. A fődiagonális elemeit a program figyelmen kívül hagyja, mivel feltesszük, hogy egy site-on belül a sávszélesség végtelen nagy.

Feladat konfigurációs állomány. Ez a fájl arról tartalmaz információt, milyen Grid fájlok vannak a Grid-ben, és milyen feladatokat küldenek be a „felhasználók” a szimuláció futása során.

- A `begin{filetable}` és az első `end` közötti rész a fájlok egy listája, ahol minden sor tartalmazza a fájl egyedi logikai fájlnevét (LFN), méretét MiB-ban, és szintén egyedi numerikus azonosítóját.
- A `begin{jobtable}` és a második `end` közötti minden sor egy feladat nevét tartalmazza, majd egy listát az ahhoz szükséges fájlokról (LFN).
- A `begin{cescheduletable}` és a harmadik `end` közötti rész sorai a következő adatokat tartalmazzák:
  - egy Grid site sorszámát,
  - egy vagy több feladat nevét.

  Ezek meghatározzák, hogy az adott sorszámú Grid site-on lévő Computing Element mely nevű feladatok futtatására képes.
- A `begin{jobselectionprobability}` és a negyedik `end` közötti rész tartalmazza, hogy az egyes feladatok milyen valószínűséggel kerülnek kiválasztásra

```
# Simple jobs for the 2 CEs in the simple network #
# File Table
#
\begin{filetable}
File1 1000 1 File2 1000 2 File3 1000 3 File4 1000 4 File5 1000 5
File6 1000 6 File7 1000 7 File8 1000 8 File9 1000 9
\end
#
# Job Table
# A job name and a list of files needed. #
\begin{jobtable}
job1 File1 File2 File3 File4 File5 File6 File7 File8 job2 File9
\end
#
# CE Schedule Table
# CE site id, jobs it will run
#
\begin{cescheduletable}
2 job1 7 job2
\end
#
# The probability each job runs
#
\begin{jobselectionprobability}
job1 0.5 job2 0.5
\end
```

1.3. ábra. *Példa feladat konfigurációs állomány.*

a Grid-be beküldésnél. Minden sor első tagja egy feladat neve, második tagja egy 0 és 1 közötti valós szám, a valószínűség.

Szimulációs paramétereket leíró konfigurációs állomány. Ez a fájl a szimuláció viselkedését szabályozó paramétereket tartalmazza `paraméter_neve = paraméter_értéke` formátumban. A lehetséges paraméterek felsorolása következik.

Általános beállítások

grid.configuration.file - a Grid felépítését leíró konfigurációs állomány elérési útja

job.configuration.file - a feladatokat leíró konfigurációs állomány elérési útja

bandwidth.configuration.file - a háttér hálózati forgalmat leíró fájl elérési útja (ld. [CCSF+04])

number.jobs - a szimuláció során beküldésre kerülő feladatok száma

users - meghatározza, hogyan küldjenek be a „felhasználók" feladatokat a Grid-be. Lehetséges értékei:

1. *Egyszerű.* A feladatok egyenlő, a **job.delay** paraméter által meghatározott időközönként kerülnek beküldésre.
2. *Véletlen.* A feladatok $U(0, 2 \cdot job.delay)$ egyenletes eloszlású időközönként kerülnek beküldésre.
3. *CMD DC04.* A feladatok a CMD Data Challenge 2004 beküldési mintáját követve kerülnek beküldésre. Ld. [CCSF+04].

scheduler - az ütemezőt (Resource Broker-t) kiválasztó kapcsoló. Lehetséges értékei:

1. *Véletlen.* A feladatok véletlenszerű CE-kre ütemeződnek (azok közül, amelyek tudják futtatni az adott feladatot).
2. *Sor hossza.* Az a CE kerül kiválasztásra a feladathoz, amely várakozási sorának hossza a legrövidebb.
3. *Fájl elérési költség.* Az a CE kerül kiválasztásra a feladathoz, ahonnan az input fájlok elérése a „legolcsóbb" (hálózati költség szempontjából).
4. *Fájl elérési költség + Sor futtatási költség.* Az ütemezés a fájlok elérési költsége és a CE sorában várakozó job-ok összköltsége alapján történik.
5. *Statikus adatszolgáltató.* Az [LKUH05] cikk algoritmusa.
6. *Ágens alapú ütemező.* A job-ok mellé olyan ágenseket küldő módszer, amelyek elkezdik a fájlok előtöltését (még az előző job-ok futása *alatt*). Ez az opció az így módosított költségekkel számoló Statikus adatszolgáltató algoritmust választja ki. ([LUKH06])

optimiser - a replikációs stratégiát meghatározó opció. Lehetséges értékei:

1. *Egyszerű.* Nincs replikálás, a fájlokat távoli módon éri el a job.

2. *Legrégebbi fájl törlése.* Mindig replikál, és a legrégebbi fájl(okat) törli, ha nincs elég hely.
3. *Legritkábban használt fájl törlése.* Mindig replikál, és a legritkábban használt fájl(okat) törli, ha nincs elég hely.
4. *EcoModelOptimiser binomiális előrejelző függvénnyel.* Ld. [CCSF+04].
5. *EcoModelOptimiser Zipf-szerű előrejelző függvénnyel.* Ld. [CCSF+04].

Megjegyzendő, hogy a `sceduler=6` opció mellett a replikációt az ágensek végzik, ennek az opciónak csak a nem használt fájlok törlése szempontjából van szerepe.

dt - a (3), (4), és (5) replikációs stratégiák esetén megadja, hogy azok hány századmásodperc ideig tekintsenek vissza a múltba a fájlok hozzáférési történetének nyilvántartásakor.

access.pattern.generator - meghatározza, hogy egy feladaton belül milyen sorrendben legyenek a fájlok felhasználva. Lehetőségek ([CCSF+04]):

1. *Szekvenciális.*
2. *Véletlen.*
3. *Véletlen végigjárás (egyenletes).* A kiinduló fájlt véletlen eloszlás szerint választja.
4. *Véletlen végigjárás (normális).* A kiinduló fájlt normális eloszlás szerint választja.
5. *Véletlen (Zipf-szerű).*

shape - a Zipf-szerű eloszlás α alak paramétere

job.set.fraction - megadja, hogy egy job input fájlainak mekkora hányadát használja fel. Pozitív valós.

initial.file.distribution - megadja, hogy a Grid fájlok mesterpéldányai hol helyezkedjenek el a szimuláció elején. Értéke lehet `random`, ekkor véletlenszerűen eloszlanak a fájlok az SE-k között, vagy egy vesszővel elválasztott számok listája, amelyben a számok azon Grid site-ok sorszámai, amelyeken a fájlok el lesznek osztva.

fill.all.sites - ha ez az érték `yes`, akkor a Grid összes SE-jét véletlenszerűen *teletölti* a program Grid fájlok replikáival.

job.delay - két feladat beküldése között eltelt időintervallum századmásodpercben. Bővebben lásd a `users` opciót.

random.seed - ha `yes`, a pszeudorandom generátor minden futtatásnál más értékről indul, ha `no`, mindig ugyanarról (így az eredmények reprodukálhatóak).

max.queue.size - a CE-k várakozási sorának maximális hossza.

file.process.time - az az idő századmásodpercben, amit elosztva a CE-ben található Worker Node-ok számával egy CE eltölt egy job futtatásával.

gui - legyen-e grafikus felület (`yes`/`no`)

A további, kevésbé releváns paraméterek tekintetében ld. [CCSF+04]-t.

Kiegészítő konfigurációs fájl. Ez a standard XML fájl az (5) és (6) ütemező algoritmusok használata esetén tárolja a job-ok jellegzetességének magasszintű leírását, valamint az erőforrások paramétereit. A nagyprogram írója által használt változat megtalálható a programkönyvtárban, `examples/edg_testbed_extensions.xml` néven. A fájl formátumának részletei az [LKUH05] cikkben olvashatóak. Az állományt egy segédprogram generálja, ennek elérhetőségével kapcsolatban a készítőkkel kell felvenni a kapcsolatot: Lőrincz László Csaba (`lesliel@inf.elte.hu`), illetve Ulbert Attila (`mormota@inf.elte.hu`).

1.3.2. A grafikus kezelőfelület

A grafikus kezelőfelület használata az előző pontok ismeretében intuitív. Bővebb leírása megtalálható [CCSF+04]-ban.

1.3.3. A kimeneti statisztikák

A program kilépéskor a szimuláció statisztikáit fába rendezve kinyomtatja a képernyőre.

A teljes Grid-re jellemző egyes mezők jelentése:

remoteReads - a távoli fájl-elérések száma

localReads - a helyi fájl-elérések száma

replications - a fájl replikációk száma

ENU - effektív hálózat használat:

$$r_{ENU} = \frac{remoteReads + replications}{remoteReads + localReads}$$

ceUsage - az idő ennyi százalékában voltak a CE-k aktívak (átlag)

totalJobTime - az összes job futási idejének összege

A Grid site-okra jellemző egyes mezők jelentése:

remoteReads - a távoli fájl-elérések száma a site-on

localReads - a helyi fájl-elérések száma a site-on

ceUsage - az idő ennyi százalékában voltak a site CE-jei aktívak (átlag)

fileAccesses - az a szám, ahányszor ezen a site-on más site elért fájlokat

routedFiles - azon fájlok száma, amelyek átvitele során ez a site mint forgalomirányító játszott szerepet

totalJobTime - az összes erre a site-ra beküldött job futási idejének összege

A Storage Element-ekre jellemző egyes mezők jelentése:

capacity - az SE kapacitása MiB-ban

usage - a használt lemezterület MiB-ban

A Computing Element-ekre jellemző egyes mezők jelentése:

remoteReads - a távoli fájl-elérések száma

localReads - a helyi fájl-elérések száma

usage - az idő ennyi százalékában volt a CE aktív

jobTimes - minden job futási ideje fel van sorolva századmásodpercben

jobTimesWithQueue - minden job futási ideje plusz a sorban állásuk ideje fel van sorolva századmásodpercben

numberOfJobs - hány job futott ezen a CE-n

workerNodes - hány Worker Node-ja van a CE-nek

jobFiles - minden futtatott job-ra az input fájlok fel vannak sorolva abban a sorrendben, ahogyan a job feldolgozta őket

totalJobTime - a CE-n lefutott összes job futási idejének összege másodperc-ben

A `totalJobTime`-ba mindenhol beleértjük a sorbanállás idejét is.

2. fejezet

Fejlesztői dokumentáció

Ez a fejezet a nagyprogram feladatát, a kiindulást, a megoldás tervét és implementációjának részleteit rögzíti, valamint mérési eredményeket mutat be. A leírás során alkalmazom az `UML 2.0` elfogadott jelöléseit.

2.1. Feladatleírás

Ebben a részben pontosan rögzítem a feladatot és a követelményeket. A feladatot az alapötletének megalkotóitól, *Lőrincz László Csaba* és *Ulbert Attila* (ELTE IK) PhD hallgatóktól kaptam (a továbbiakban: Megrendelők). Röviden a következőképpen lehetne összefoglalni:

> A meglévő OptorSim szimulátor kiegészítése ágensek szimulációjával, amelyek egy új, szintén implementálandó Grid ütemező algoritmussal szorosan együttműködve optimalizálják a Grid-be beküldött feladatok futási idejét azáltal, hogy a szükséges input fájlokat még a feladat futásának megkezdése *előtt* elkezdik lemásolni a feladatot futtató CE-hez közeli egy vagy több SE-re.

Pontosítva a feladatleíráson, az `OptorSim`-nek is már az ötletgazdák által egy módosított változatát kell alapul venni, amelyben implementálva van egy új ütemező algoritmus. Ennek részleteit az [LKUH05] cikk, valamint egy rövid összefoglalást a 2.3. rész tartalmaz.

Ahhoz, hogy az új elemeket megfelelően el tudjuk helyezni a rendszerben, az `OptorSim` eredeti változatával meg kell ismerkednünk. Ennek rövid leírását foglalja össze a 2.2. rész, a teljesség igénye nélkül, csak a releváns részletekre kitérve.

2.1.1. Az ágensek feladata

A szimulálandó rendszer lényege: minden CE és SE egyben egy ágenshoszt is, azaz képesek ágensek fogadására és futtatására. Amikor a felhasználó beküld egy feladatot a Grid-be, az új ütemező fogadja azt, kiválasztja azt a CE-t, amin becslése szerint a leghamarabb véget érne a feladat futtatása, majd elküldi a feladatot a kiválasztott CE feladatkezelőjének (Job Handler), egy ágenssel együtt. Az ágens azonnal elkezd futni, és

megpróbálja az SE-ken már elve futó (vagy szintén elküldött) ágensekkel kommunikálva a feladathoz szükséges input fájlokat a CE-hez közeli SE-kre másolni. A másolást akkor kezdi csak el, ha az adott CE-n nincs olyan futó vagy várakozó feladat, amelynek ágense fájlokat másol. A feladat (job) az indulásakor megvárja, hogy az ágens lemásolja az összes lehetséges fájlt, majd lefut. Ezek után az input fájlok foglaltsága megszűnik, azok szükség esetén törölhetővé válnak, valamint a CE-n a feladathoz tartozó ágens életciklusa is véget ér.

A módszer ezen leírása a fejlesztési ciklus alatt a Megrendelőkkel folytatott kommunikáció során, fokozatosan alakult ki. A módszer részleteinek kidolgozása a nagyprogram írójára maradt.

2.1.2. Az új ütemező

A Megrendelők által javasolt ágensekkel kiegészített rendszerhez új ütemező is szükségeltetik, amely egyrészt képes együttműködni az ágensekkel, másrészt döntésénél figyelembe veszi az ágensek munkáját.

Az új ütemezőnek tehát legfontosabb tulajdonsága, hogy amikor számítja, hogy egy feladat mikor kerülne futtatásra az egyes CE-ken, figyelembe kell vennie, hogy az ágensek az input fájlok egy részét vagy egészét már *előre* (az előző job-ok futása *alatt*) odamásolták, így azok másolási idejét nem kell beleszámítani a várakozási időbe.

A következőkben a funkcionális és nem funkcionális követelmények rögzítése következik.

2.1.3. Funkcionális követelmények

A funkcionális követelményeket a 2.1. táblázat rögzíti. A követelmények fontossága és a megvalósításuk becsült nehézsége ötfokozatú skálán van feltűntetve, ahol 1 a legkevésbé fontos vagy legkönnyebb, 5 a legfontosabb vagy legnehezebb szintet jelöli.

2.1.4. Nem funkcionális követelmények

A nem funkcionális követelményeket a 2.2. táblázat rögzíti.

2.2. Az OptorSim keretprogram

Az `OptorSim` egy **Java™** csomag, amelyet Adat Grid-ek szimulációjára fejlesztettek ki. Az igazi Adat Grid-ek felépítését utánozza, és segítségével replika optimalizációs és ütemező algoritmusok viselkedése szimulálható. E dokumentum írásakor elérhető aktuális változata a nyílt forráskódú $v2.0$, letölthető a

`http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html`

Követelmény azonosító	A követelmény szöveges leírása	Fontosság	Nehézség
K-OptorExt	A megoldásnak a megadott `OptorSim` keretprogramot kell kiegészítenie.	5	-
K-OptorCnf	Az `OptorSim` konfigurációs fájljait kell használni a paraméterek beállításához, megtartva a kompatibilitást a régi konfigurációs fájlokkal.	3	1
K-OptorGUI	Az `OptorSim` grafikus felületével együtt kell működni, a hozzáadott komponensek munkájának szintén meg kell jelennie a grafikus felületen.	3	1
K-OptorSta	Az `OptorSim` statisztikáinak a hozzáadott komponensek munkáit is tartalmaznia kell, továbbá a statisztikáknak pontosaknak kell lenniük.	5	3
K-CEAHost	Szimulálni kell az egyes CE-ken az ágenshoszt funkcionalitást.	5	2
K-CEAgent	Szimulálni kell az ágenseket, amelyek a CE-ken futnak, és a feladatok input fájlainak előtöltését vezénylik.	5	5
K-SEAHost	Szimulálni kell az egyes SE-ken az ágenshoszt funkcionalitást.	5	2
K-SEAgent	Szimulálni kell az ágenseket, amelyek az SE-ken futnak, és kiszolgálják a CE-n futó ágensek kéréseit.	5	5
K-AgentRB	Implementálni kell az ágensekkel együttműködő Resource Broker algoritmust.	5	5
K-AgentJH	A jelenlegi feladatkezelőt (Job Handler-t) olyanra kell cserélni, amely képes ágensek fogadására.	5	3

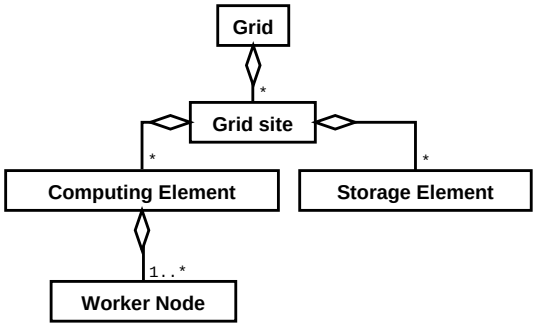
2.1. táblázat. Funkcionális követelmények

Követelmény azonosító	A követelmény szöveges leírása	Fontosság	Nehézség
K-Deadline	A rendszert egy megadott határidőre kellett elkészíteni.	5	5
K-Robust	A megoldásnak megbízhatónak kell lennie. (Például: lehetőleg ne legyenek benne holtpontok, olyanok sem, amelyek csak extrém esetben jönnek elő, stb.)	4	5
K-Java1.4	A programnak Java nyelven kell íródnia, és nem igényelhet Java 1.4-nél frissebb környezetet.	2	1
K-Generic	A megoldás lehetőleg legyen minél általánosabb, ne használja ki a rendszer speciális tulajdonságait (Motiváció: pl. a jövőbeli bővíthetőség kedvéért)	3	5

2.2. táblázat. Nem funkcionális követelmények

címről.

Az `OptorSim` a következő Grid felépítést feltételezi és követi:



2.1. ábra. *Az `OptorSim` által feltételezett Grid felépítés*

A **Grid** több logikai egységre, **Grid site**-ra van tagolva. Egy Grid site-on belül az adattárolásért felelős **Storage Element**ek (a továbbiakban: SE-k), illetve a számításért felelős **Computing Element**ek (CE-k) lehetnek, ez utóbbiak **Worker Node**-okból (WN-ek) állnak. A CE a számítógépfürt absztrakciója, így a WN egy konkrét számítógépet reprezentál. A CE-k feladatokat (**job**-okat) futtatnak, amikhez az SE-k szolgáltatják az adatokat. A feladatokat a CE-hez tartozó feladatkezelő (**Job Handler**) fogadja, és állítja várakozási sorba. Egy **Resource Broker** felelős a feladatok CE-khez rendeléséért. Az SE és CE nélküli site-ok hálózati csomópontként vagy forgalomirányítóként funkcionálnak.

Az Adat Grid-ek állapota időben dinamikusan változik, így szükséges, hogy megpróbáljuk az erőforrások kihasználtságát optimalizálni. A szimulátorban két lehetőség nyílik erre:

- a Resource Broker algoritmus megválasztásával, és
- a fájlok replikáinak megfelelő létrehozásával és törlésével, amit az `OptorSim`-ben az `Optimiser` interface-t implementáló oszályok valósítanak meg a **Replica Manager** komponens segítségével.

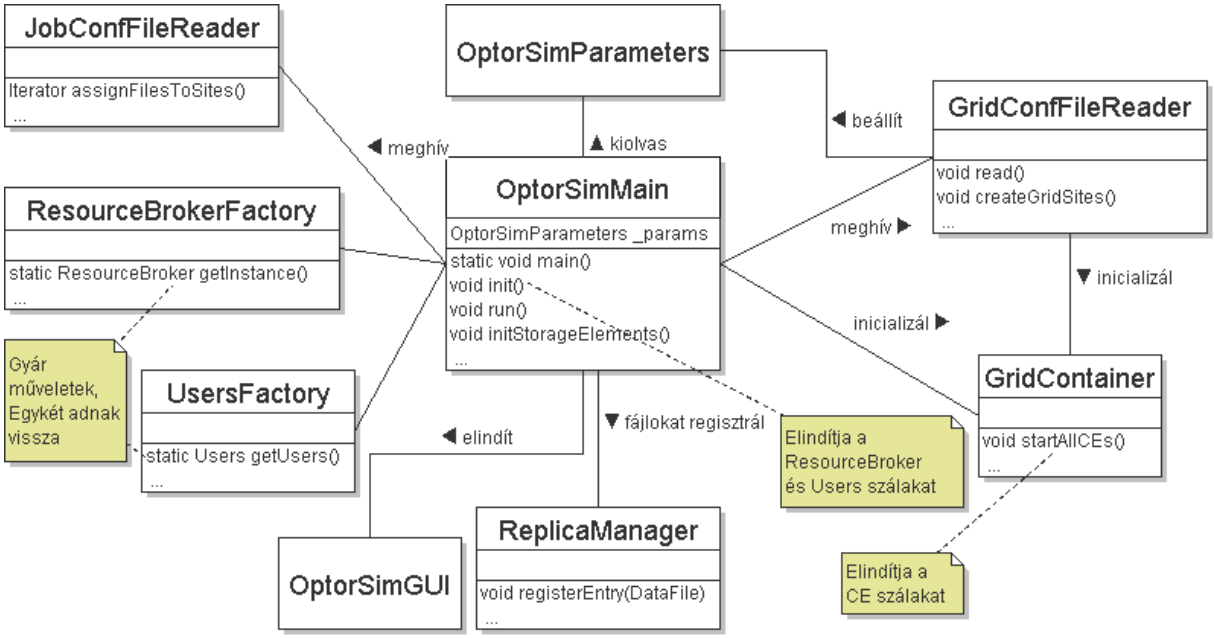
Az OptorSim indulása, a szimuláció menete

Az `OptorSim` indulása után beolvassa a konfigurációs fájlokat (ld. a felhasználói dokumentációt), megfelelően beállítja a `GridContainer` osztály egyetlen példányát (amely a Grid struktúrát foglalja magában), majd ez alapján elindítja a szimulációt. A szimulációban minden Computing Element-nek egy szál felel meg. A feladatok beküldését egy másik típusú (Users) szál végzi, ami a felhasználóknak felel meg. Ennek a szálnak a kéréseit egy harmadik típusú szál, a Resource Broker kezeli, elosztja a feladatokat a CE-k között. Az indulás menete és az ott szereplő osztályok közötti kapcsolat a 2.3. szekvencia és a 2.2. osztálydiagramokon látható. (*Megj.:* ezek és a további a diagramok a komplexitás csökkentése érdekében csak az adott szempontból érdekes osztályokat és metódusokat tartlamazzák, és néhol esetleg nem teljesek.)

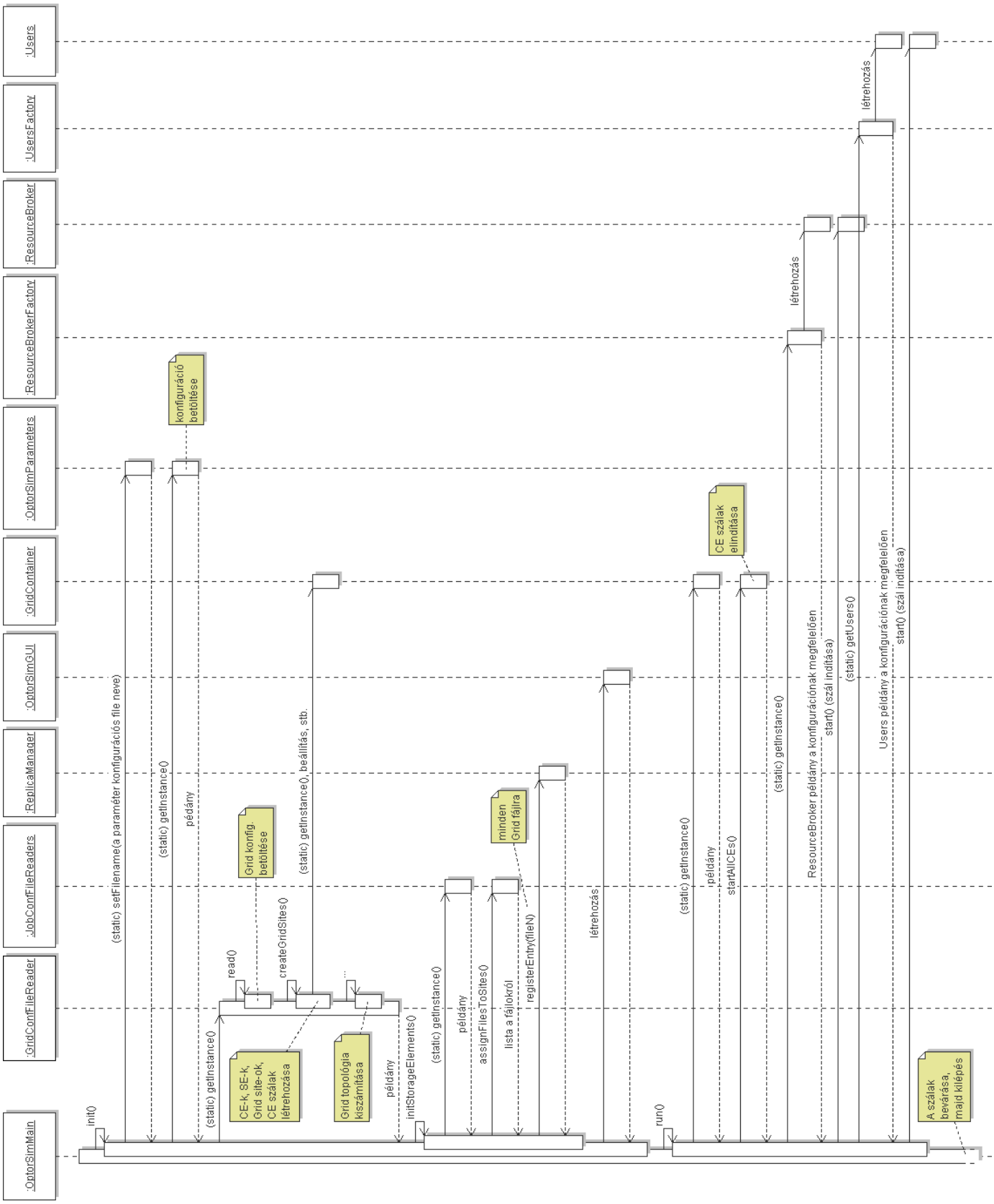
A `ResourceBrokerFactory`, `UsersFactory` osztályok megfelelő metódusai **Gyár művelet** tervmintával hozzák létre a konfigurációnak megfelelő `ResourceBroker` illetve `Users` interface-t implementáló osztályok példányait.

Az `OptorSimParameters`, `GridConfFileReader`, `JobConfFileReader`, `GridContainer` osztályok az **Egyke**[1] tervmintát alkalmazzák.

[1] Singleton



2.2. ábra. *Az `OptorSim` indulásának osztálydiagramja*



2.3. ábra. *Az* `OptorSim` *indulásának szekvencia diagramja*

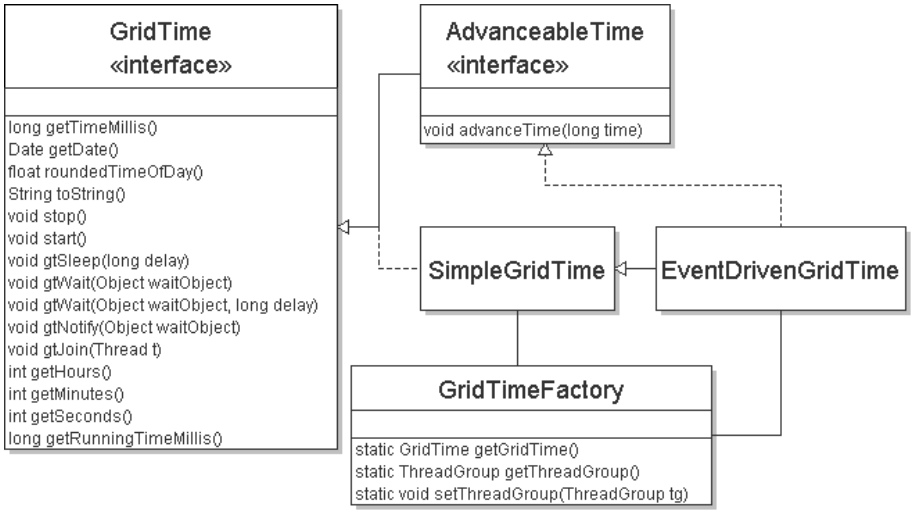
A Grid-idő szimulációja

Az `OptorSim`-ben kétfajta idő modell került implementálásra, és a program bármelyiket alkalmazva futtatható, egyező eredménnyekkel. A két idő modell:

Idő alapú. A szimuláció valós időben fut.

Eseményvezérelt. Az eseményvezérelt modellben, amikor minden CE, Resource Broker és Users szál inaktív, a program a szimuláció idejét a következő esemény bekövetkezésének idejére állítja előre.

Az eseményvezérelt modell jelentősen felgyorsítja a szimuláció futását, míg az idő alapú modell demonstrációs célokat szolgálhat (például a grafikus felület használatával).
A Grid idő szimulációjáért felelős osztályok diagramja a 2.4. ábrán látható.



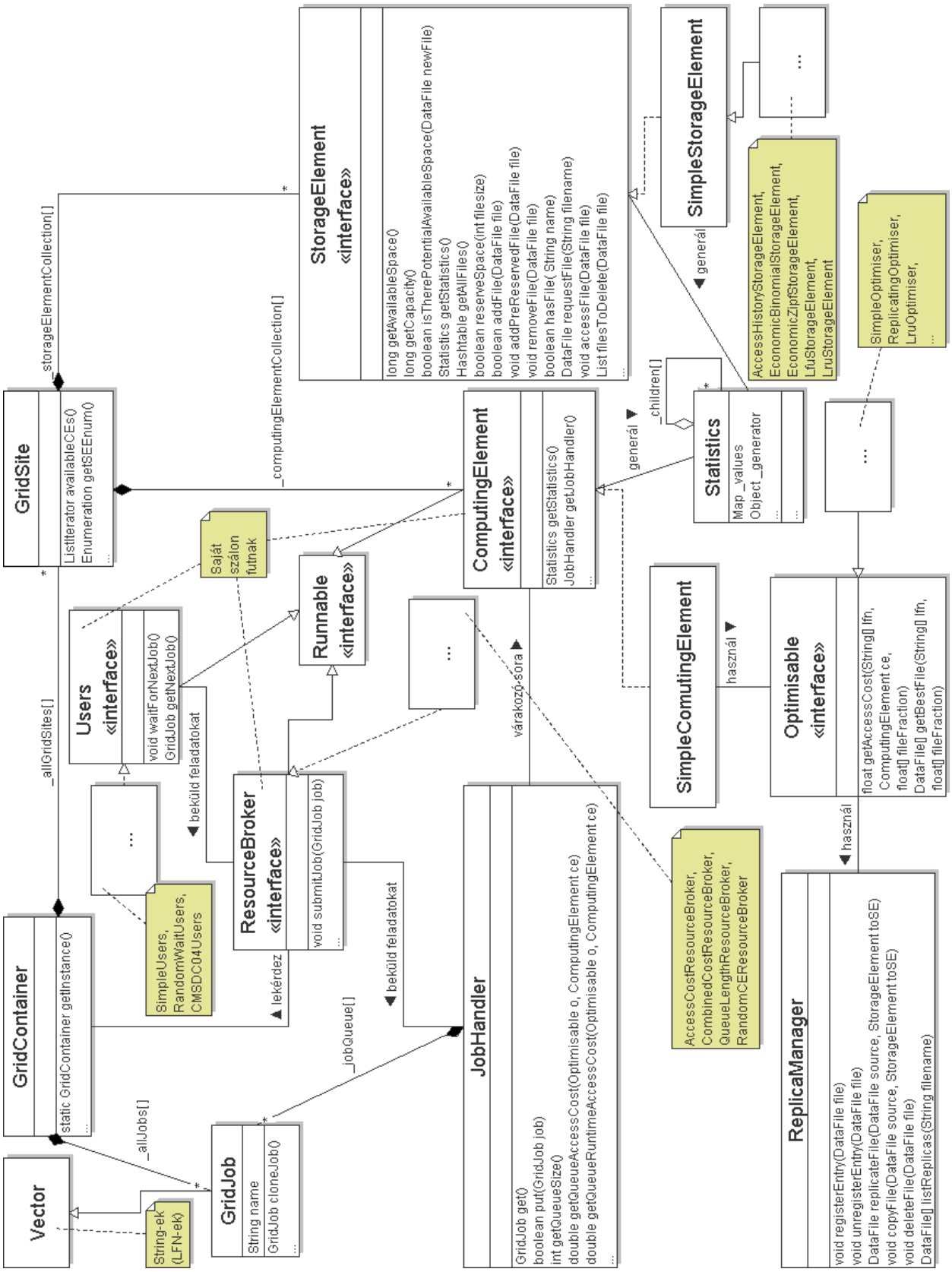
2.4. ábra. *A Grid idő szimulálásában részt vevő osztályok diagramja*

A programban a szálak szinkornizálásának technikája a Java-ban megszokott *wait/notify* technika. Ahhoz azonban, hogy az eseményvezérelt idő működni tudjon, a Grid időt megvalósító objektum hasonló szinkronizációs műveleteit kell meghívnunk (`gtWait()`, `gtNotify()`) a Java alapműveletei helyett, valamint a várakozásnál is ugyanez érvénes (`gtSleep()`, `gtJoin()`). Ezen műveletek a Java alapműveletek direkt helyettesítői, azonos szemantikával. A konfigurációs fájlban beállított Grid időt megvalósító objektumot a `GridTimeFactory` osztály `getGridTime()` metódusa adja vissza, Gyár művelet tervminta alkalmazásával. A későbbiekre az implementáció tekintetében megjegyzendő még, hogy szintén az eseményvezérelt idő működésének érdekében a szimuláció szereplőit megvalósító szálakat (CE, Resource Broker, stb.) egy közös szál-csoportba kell illesztenünk, a `GridTimeFactory` osztály `getTreadGroup()` metódusa által visszaadott csoportba.

A szimuláció felépítése

A szimuláció működése szempontjából érdekes osztályokat a 2.5. diagram szemlélteti. A diagramban a „...”-tal jelölt helyeken az interface-ek különböző implementációi szerepelnek, ezeken a helyeken van lehetőség a rendszer bővítésére újabb osztályok bevezetésével (amelyek megvalósítják az adott interface-t), és a megfelelő gyár művelet kiegészítésével.

A szimuláció menete a következő: a `Users` interface-t megvalósító egyke objektum egy külön szálon fut, `GridJob`-okat (amik lényegében LFN-eket tartalmazó vektorok) állít elő, és elküldi a szintén külön szálon futó `ResourceBroker` interface-t implementáló objektumnak. Az a saját algoritmusát alkalmazva, a `GridContainer`-től lekérdezve a lehetséges `ComputingElement`-ek listáját dönt a CE-k között. A kiválasztott CE `JobHandler`-ének beküldi a feladatot, azaz a feladat bekerül a CE várakozási sorába, ahonnan a saját szálon futó `ComputingElement`-et implementáló objektum fogyasztja a feladatokat. Egy feladat végrehajtásának elején a CE meghívja a beállított `Optimisable` interface-t implementáló objektumot, a `getBestFile(...)` metódussal választja ki a használandó input fájlokat az LFN-ek alapján (vagyis az `Optimisable` interface-t implementáló osztályok valósíthatják meg a *replika optimalizációs stratégiákat* ezen művelet megvalósításával). A fájlokat a job futtatásának az elején a replikációs stratégiát megvalósító metódus *leszögezi*, vagyis mások által törölhetetlenné teszi. Ezek után várakozik a fájlok eléréséhez szükséges ideig, törli a leszögezést a replikákról (így szükség esetén törölhetővé válnak), majd halad a következő feladatra. A folyamat során statisztikákat készít, amit kilépéskor a program megjelenít.



2.5. ábra. *A szimulációban részt vevő osztályok diagramja*

2.3. A módosított OptorSim

A fejlesztés alapjául a Megrendelők által kiegészített `OptorSim` csomagot kellett venni. Ebben az eredeti `OptorSim`-et egy új ütemező és végrehajtási időt becslő algoritmussal bővítették ki (ld. [LKUH05]), melynek lényege:

- Megfigyeli a feladatok végrehajtását, és az erőforrások elérésének jellegzetességeit.
- Analizálja az összegyűjtött információt, és egy magasszintű (XML) leírást készít el a feladat jellegzetességeiről.
- Az így generált leírásokat kihasználja, hogy a feladat további futtatásait a feladat jellegéhez optimizált módon végezhesse. Az ütemező az ezzel az optimizációval becsült futási idők alapján dönt a CE-k között.

Ez a szimuláció számára azt jelenti, hogy a feladatok CE-ken vett futási idejeit az új algoritmus által szolgáltatott adatokkal kell becsülni.

A bővítés az alábbi osztályokat adta az `OptorSim`-hez:

ExtensionsConfFileReader. A magas szintű job-jellegzetességek XML-ből való beolvasását végző komponens.

scheduler.* Az XML elemzését és a becslések számítását segítő osztályok.

StaticDataResourceBroker. A becsléseket használó ütemező.

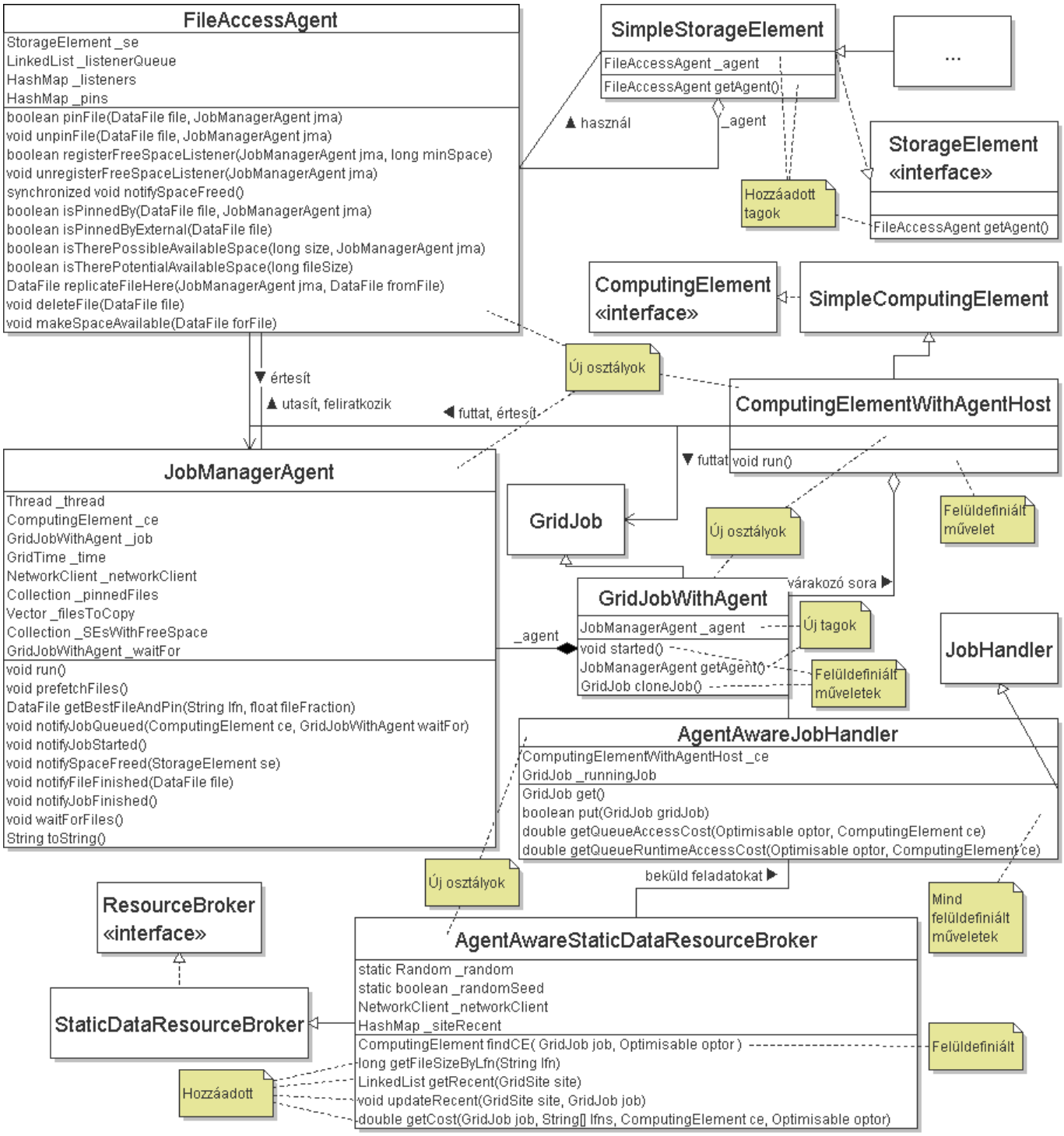
A továbbiakban ezt a hozzáadott komponenst fekete dobozként kezelem, megjegyezve, hogy a művelet, amely megadja egy `ce ComputingElement`-en egy `job GridJob` becsült futási idejét az

```
ExtensionsConfFileReader.JobSchedule.get(job,ce).getComputationCost()
```

metódus.

2.4. Tervezés és implementáció

A tervezést a jelenlegi `OptorSim` architektúra megismerésével és a lehetőségeinek feltérképezésével kezdtem. Megállapítottam, hogy egyik követelmény teljesíthetőségének sincs akadálya. Ezek után a kiegészítő komponensek statikus modelljét terveztem meg, folyamatosan ellenőrizve, hogy könnyen megvalósítható-e a meglévő `OptorSim`-ben. Ezután a dinamikus modell megtervezése következett. Alább ezeknek az implementáció során finomított változatait dokumentálom.



2.6. ábra. *A hozzáadott komponensek osztálydiagramja*

2.4.1. A hozzáadott komponensek és kapcsolódásuk

A hozzáadott komponensek osztálydiagramja a 2.6. diagramon látható. A hozzáadott vagy módosított osztályok listája, és az osztályok szerepe a következő:

FileAccessAgent. A Storage Element-ekhez tartozó ágenst szimuláló osztály. Feladata, hogy kiszolgálja a `JobManagerAgent`-ek kéréseit, illetve értesítse azokat változás (elsősorban szabad helyben történő változás) esetén.

SimpleStorageElement. Ez az osztály ki lett egészítve egy `FileAccessAgent` ágenssel.

JobManagerAgent. A Computing Element-eken futó, feladatokhoz tartozó ágenst szimuláló osztály. Feladata, hogy elvégezze a fájlok előtöltését a `FileAccessAgent`-ekkel kommunikálva, fogadja a CE értesítéseit a feladat állapotváltozásairól. (Megjegyzendő, hogy a valóságban a feladat maga kommunikálna a `JobManagerAgent`-tel a CE helyett, az `OptorSim`-ben azonban a feladat nagyon leegyszerűsített szerepe miatt ez nem így van.)

ComputingElementWithAgentHost. Olyan CE működését szimulálja, amely képes ágensek fogadására és futtatására.

AgentAwareJobHandler. Olyan feladatkezelőt szimulál az egyes CE-ken, amely képes a `JobManagerAgent`-ekkel együttműködni, és a várakozási sorhoz tartozó költséget az ágenseket munkáját figyelembe véve számolni.

GridJobWithAgent. Ágenssel rendelkező feladatot szimulál.

AgentAwareStaticDataResourceBroker. Az ágensek munkáját figyelembe vevő, a `StaticDataResourceBroker`-t kibővítő Grid ütemezőt valósít meg.

A fenti változtatásokon kívül a megfelelő `...Factory` osztályokat ki kell egészíteni az új komponensek létrehozásának lehetőségével, valamint a konfigurációs állományokat betöltő eljárásokban is be kell vezetni a megfelelő opciókat (ld. a felhasználói dokumentációt).

2.4.2. Metódus referencia

Az osztályok metódusai a következők szerint kerültek meghatározásra:

JobManagerAgent

JobManagerAgent(GridJobWithAgent job) Konstruktor, amely létrehoz egy `JobManagerAgent` objektumot, és hozzárendeli a `job` feladatot.

void run() Az ágens szálának belépési pontja, az ágens logikáját kell, hogy tartalmazza. A `prefetchFiles()`-t hívja.

void prefetchFiles() Az ágens fő működését valósítja meg, a fájlok előtöltését. Lépései:

1. Az előző ágens bevárása.
2. Minden nem lokális input fájl lokális (azonos Grid site-on lévő) SE-re másolása.
3. Ha még van nem lokális input fájl, de nem lehet másolni helyhiány miatt, akkor regisztrálás az SE-k hely felszabadulását jelző szolgáltatására.
4. Ha felszabadul hely, újra próbálkozás.
5. Ha minden input fájl lokális vagy nem lehet másolni kapacitás hiány miatt, akkor vége.

DataFile getBestFileAndPin(String lfn, float fileFraction) A legkisebb elérési költségű fájl megkeresése `lfn` alapján. `fileFraction`: a fájl mekkora hányada kell ($\in [0, 1]$).

void notifyJobQueued(ComputingElement ce, GridJobWithAgent waitFor) Értesítés, amit az ágens egy `AgentAwareJobHandler`-től kap, ha a feladat bekerül a várakozási sorba. `waitFor`: a sorban az előző job.

void notifyJobStarted() Értesítés, amit az ágens a job-ot éppen futtatni készülő `ComputingElementWithAgentHost`-tól kap.

void notifySpaceFreed(StorageElement se) Értesítés, amit az ágens az `se` SE `FileAccessAgent`-jétől kap, ha az SE-n felszabadul hely.

void notifyFileFinished(DataFile file) Értesítés, amit az ágens a job-ot futtató `ComputingElementWithAgentHost`-tól kap, amikor a job beolvasta a `file` input fájl.

void notifyJobFinished() Értesítés, amit az ágens a job-ot futtató `ComputingElementWithAgentHost`-tól kap, amikor a job lefutott.

void waitForFiles() Egy másik szálon futó ágens által hívott blokkoló eljárás, ami addig vár, amíg ez az ágens be nem fejezi az előtöltést.

String toString() Az ágens szöveges reprezentációját adja vissza.

FileAccessAgent

FileAccessAgent(StorageElement se) Konstruktor, amely egy `FileAccessAgent`-et hoz létre, és hozzárendeli az `se` SE-hez.

boolean pinFile(DataFile file, JobManagerAgent jma) Leszögezi a `file` fájlt, és nyilvántartásba veszi, hogy a `jma` ágens leszögezte a fájlt. Számolja a leszögezés multiplicitását. `true`-t ad vissza, ha sikeres a leszögezés, `false`-t egyébként.

void unpinFile(DataFile file, JobManagerAgent jma) Felenged egy leszögezést a `file` fájlról, amit előzőleg a `jma` tett rá (vagyis csökkenti a leszögezés multiplicitását eggyel).

boolean registerFreeSpaceListener(JobManagerAgent jma, long minSpace) Felíratja a `jma` ágenst a szabad hely hírdetésekre, hogy minden olyan szabad hely változásról kapjon majd értesítést az ágens, amely során legalább `minSpace` szabad hely válik elérhetővé az SE-n.

void unregisterFreeSpaceListener(JobManagerAgent jma) Leíratja a `jma` ágenst a szabad hely hírekről.

void notifySpaceFreed() Értesítés, amelyet az SE-től kap a `FileAccessAgent`, ha hely szabadul fel (ez a valóságban akár pl. *polling* technikával is történhet).

boolean isPinnedBy(DataFile file, JobManagerAgent jma) Visszaadja, hogy a `file` le van-e szögezve a `jma` ágens által.

boolean isPinnedByExternal(DataFile file) Visszaadja, hogy a `file` le van-e szögezve, de nem ágens által.

boolean isTherePossibleAvailableSpace(long size, JobManagerAgent jma) Visszaadja, hogy az SE-n elérhető-e, vagy elérhető lehet-e a jövőben legalább `size` hely a `jma` ágens számára. (Tehát a lehetőséget vizsgálja, hogy nem *eleve lehetetlen* kérés-e.)

boolean isTherePotentialAvailableSpace(long fileSize) Visszaadja, hogy az SE-n elérhető-e legalább `size` hely (esetlegesen nem használt, azaz nem leszögezett és nem *master* fájlok törlésével).

DataFile replicateFileHere(JobManagerAgent jma, DataFile fromFile) Függvény, amely a `fromFile` fájlt az SE-re másolja, és be is jegyzi a replika katalógusba. A hívót a fájl másolás idejéig blokkolja. Visszaadja a másolatot.

void deleteFile(DataFile file) Logikailag törli a `file` fájlt az SE-n. (Azaz megjelöli, hogy a jövőben, ha kell, törölhető. – „lusta törlés”)

void makeSpaceAvailable(DataFile forFile) Megpróbál letörölni annyi fölösleges fájlt, hogy legyen hely a `forFile`-nak az SE-n.

ComputingElementWithAgentHost

void run() A CE szálának belépési pontja. Implementálnia kell a CE logikáját, vagyis a feladatok várását, majd futtatását, és a megfelelő pontokon a feladatokhoz tartozó ágensek értesítését.

GridJobWithAgent

void started() A job elindulásakor (egy CE-n) meghívott eljárás. A `GridJob started()` eljárásától annyiban több, hogy értesíti az ágenst a feladat indulásáról (`notifyJobStarted()`), majd megvárja, amíg megjönnek a fájlok (`waitForFiles()`), ezzel felfüggesztve a CE szálát, és ez a várakozási idő beleszámít majd a futtatás idejébe.

GridJob cloneJob() Duplikálja a job-ot és az ágensét.

AgentAwareJobHandler

GridJob get() Kivesz egy job-ot a várakozási sorból. Blokkol, ha az üres.

boolean put(GridJob gridJob) Betesz egy job-ot a várakozási sorba. Ha az ágenssel rendelkező job volt, akkor értesíti az ágenst (`notifyJobQueued(...)`). Visszaadja, hogy sikeres volt-e a job sorba illesztése.

double getQueueRuntimeAccessCost(Optimisable optor, ComputingElement ce) Kiszámolja azt az időt, ami az összes sorban álló job lefuttatásához szükséges. Itt figyelembe kell venni az ágensek munkáját, vagyis hogy párhuzamosság van a job-ok futása és az input fájlok másolása között.

double getQueueAccessCost(Optimisable optor, ComputingElement ce) Kiszámolja a sorban álló job-ok fájl elérési idejeinek összegét.

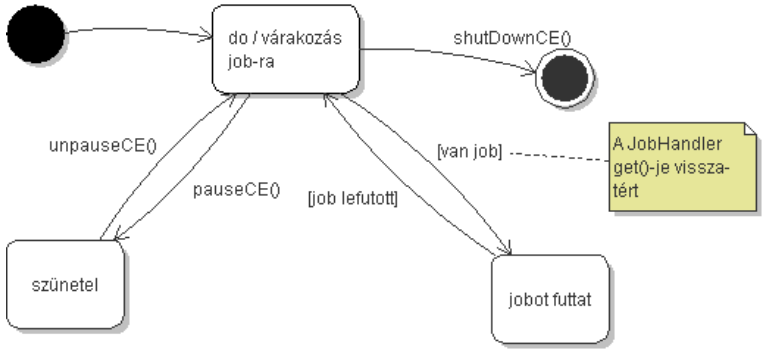
AgentAwareStaticDataResourceBroker

ComputingElement findCE(GridJob job, Optimisable optor) Kiszámolja az összes CE költségét, vagyis hogy melyiken mikor érne véget a `job`. A minimális költségű CE-t adja vissza. Az egyenlő költségű CE-k közül úgy választ, hogy egy listát vezet a legutóbb az egyes site-okra küldött job-ok fájljairól, és arra a CE-re küldi a job-ot, ahol a `job` és a legutóbbi fájlok között a legtöbb egyezés van („recent rendszer”). Ha még mindig nem egyértelmű a CE, akkor véletlenszerűen dönt.

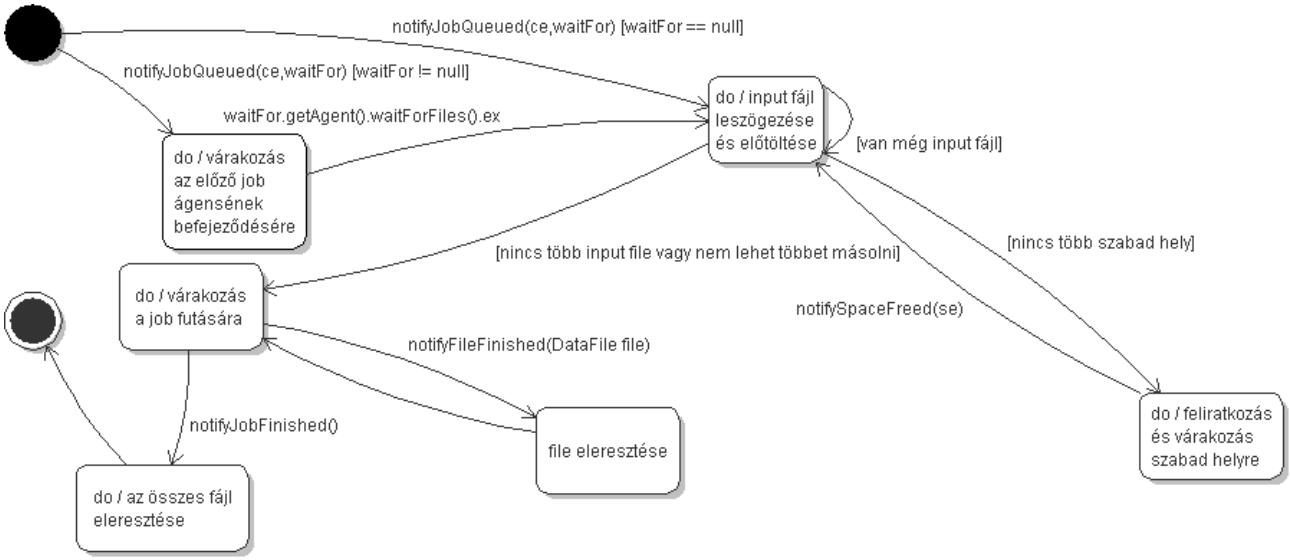
long getFileSizeByLfn(String lfn) Visszaadja az `lfn` logikai fájlnevű fájl méretét, vagy 0-t ha nincs ilyen.

LinkedList getRecent(GridSite site) Visszaadja a legutóbbi fájlok listáját, amik a `site` site-ra lettek küldve.

void updateRecent(GridSite site, GridJob job) Frissíti a legutóbbi fájlok listáját, amik a `site` site-ra lettek küldve. Teszi ezt úgy, hogy az azon a site-on lévő SE-k összkapacitásán belül elférő legutóbbi fájlokról listát vezet, a kapacitáson túllógó legrégebbi fájlokat eldobja.



2.7. ábra. A `ComputingElementWithAgentHost` állapotdiagramja

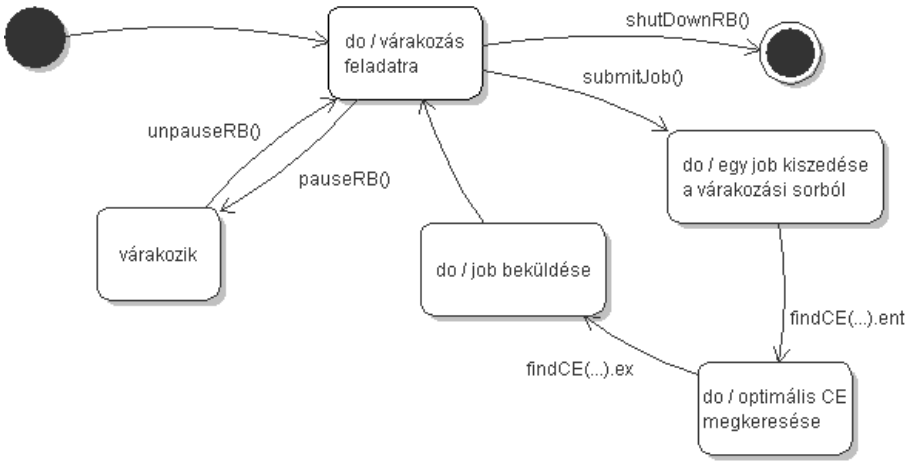


2.8. ábra. A `JobManagerAgent` állapotdiagramja

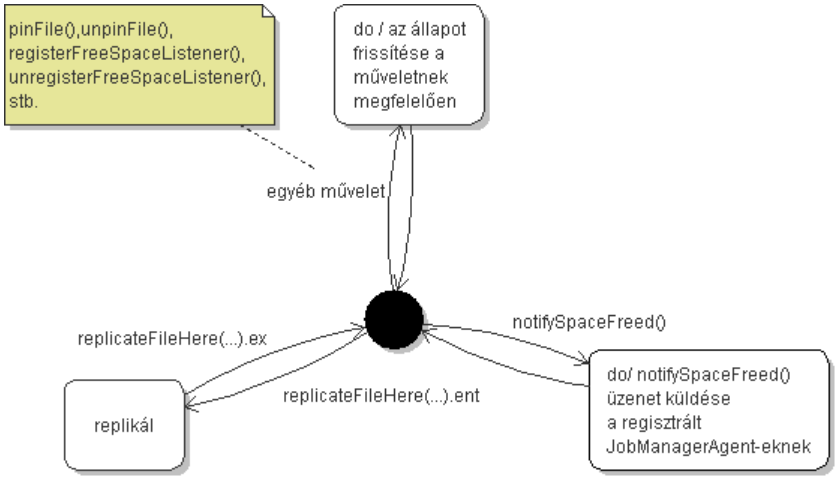
double getCost(GridJob job, String[] lfns, ComputingElement ce, Optimisable optor)
Megbecsüli az `lfns` input fájlokkal rendelkező `job` futtatásának költségét a `ce` CE-n. A költség a CE várakozási sorának összes job-jának futtatási költségének összege, plusz a `job` futtatási költsége (fájl elérési + számítási ideje).

2.4.3. A hozzáadott komponensek dinamikus viselkedése

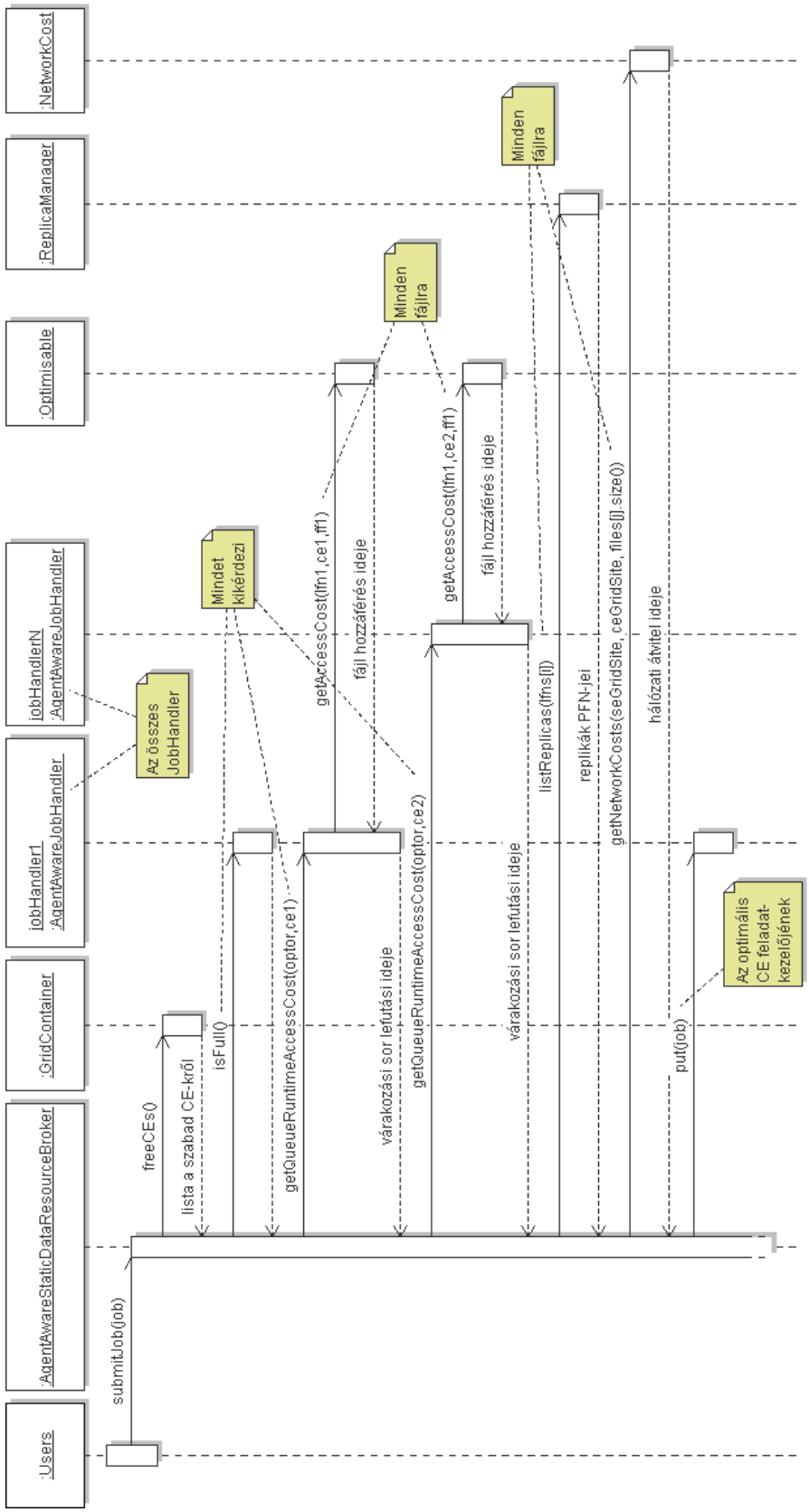
A dinamikus modellt a 2.7–2.12. ábrák írják le. Megjegyzendő, hogy az implementáció során különös tekintettel kell lenni a szereplők párhuzamos viselkedésének helyességére.



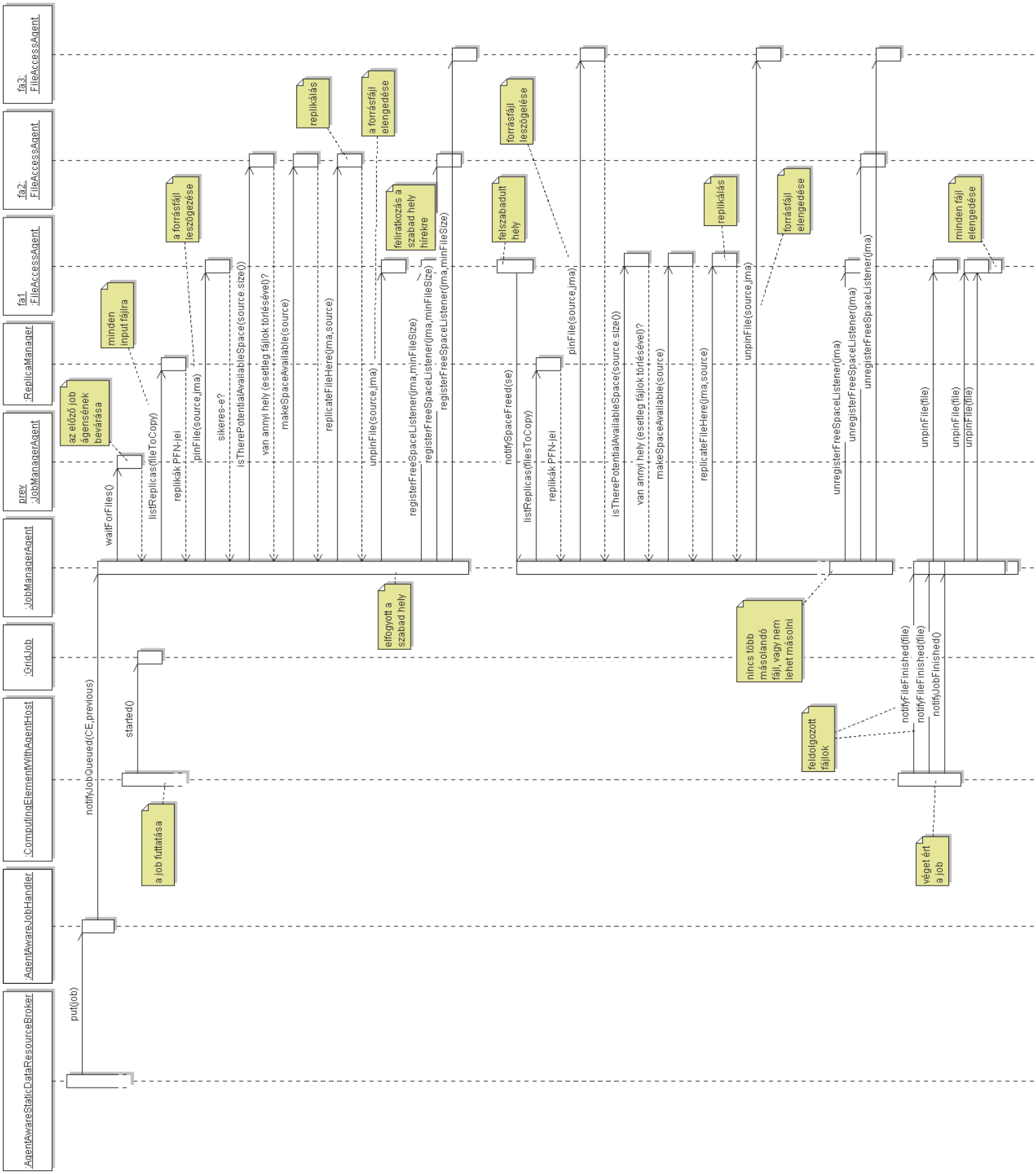
2.9. ábra. A `AgentAwareStaticDataResourceBroker` állapotdiagramja



2.10. ábra. A `FileAccessAgent` állapotdiagramja



2.11. ábra. Egy job beküldésének szekvencia diagramja



2.12. ábra. Egy job feldolgozásának szekvencia diagramja

2.4.4. Fejlesztői környezet

Az alkalmazás kifejlesztéséhez a `JetBrains IntelliJ Idea 5.1` ([Ide]) fejlesztői környezet próbaváltozatát használtam (megtalálható a CD `Egyeb/IntelliJ IDEA/` könyvtárában). Az `UML` diagramok megrajzolásában a `Violet` ([Hor], a CD `Egyeb/Violet/` könyvtárában) és a `Dia` ([Dia], a CD `Egyeb/Dia/` könyvtárában) programok voltak a segítségemre. A programozási nyelv az `OptorSim` miatt kötött volt, így a program `Java` nyelven készült.

2.4.5. Implementáció

Az implementáció a fenti statikus illetve dinamikus modellnek megfelelően történt, ügyelve a szálak megfelelő kölcsönös kizárásának és szinkronizációjának biztosítására.

Az alábbi módosításokat kellet végrehajtani az `OptorSim`-en az új komponensek hozzáadásán túl, hogy az együtt tudjon működni az új elemekkel:

- A `GridConfFileReader`-ben a `void createGridSites()` metódust ki kellett egészíteni egy elágazással, hogy szükség esetén `ComputingElementWithAgentHost`-ot hozzon létre `SimpleComputingElement` helyett.
- A kódismétlés elkerülése érdekében a `JobHandler` néhány adattagjának láthatóságát `private`-ről `protected`-re kellett változtatni.
- A `ResourceBrokerFactory`-t ki kellett egészíteni egy újabb `case` ággal, hogy szükség esetén `AgentAwareStaticDataResourceBroker`-t hozzon létre.
- A `SimpleComputingElement` néhány tagjának láthatóságát `private`-ről `protected`-re kellett változtatni.
- A `GridContainer`-ben be kellett vezetni egy egyszerű `increaseReplicationCount()` segédfüggvényt a pontos statisztikák vezetése érdekében.
- A `JobConfFileReader` `readFile()` metódusában egy elágazással biztosítani kellett, hogy szükség esetén `GridJobWithAgent`-ek jöjjenek létre `GridJob`-ok helyett.
- A `OptimiserFactory`-t ki kellett egészíteni egy újabb `case` ággal, hogy szükség esetén `SimpleOptimiser`-t hozzon létre.
- A `SimpleStorageElement` `removeFile()` metódusában meg kellett hívni az ágens `notifySpaceFreed()` metódusát.

2.4.6. Tesztelés és a követelmények ellenőrzése

A probléma természeténél fogva a tesztelés nehézkes, nehezen jellemezhető a rendszer várt reakciója a különböző bemenetekre. Ezért azt a módszert választottam, hogy a paraméterek sokféle beállítása mellett végigkövettem a rendszer működését (nyomkövetéssel

és a rendszer állapotát jellemző képernyőüzenetekkel), és megbizonyosodtam róla, hogy valóban a vártaknak megfelelően működik, ellenőriztem a statisztikák konzisztenciáját. A tesztesetek többek között a 2.3. táblázatban feltűntetettek voltak. A követelmények teljesülésének mértékét a 2.4. táblázat foglalja össze.

A tesztelés és már a fejlesztés során is rábukkantam néhány nem várt hibára, amik már az eredeti `OptorSim`-ben voltak benne. Ezek közül a legfontosabbak a következők:

2.4.7. Felfedezett hibák

Adatvesztés – A fájlok kezdeti elosztása

A `DataFile` osztálynak van egy `_master` adattagja, ami azt jelzi, hogy a fájl egy mesterpéldány, nem replika. A fájlok eredeti példányának elhelyezésekor be kell tehát ezt állítani, különben az egyetlen példány később letörlődhet. A `GridSite.java` `DataFile addFileToRandomSE(DataFile file)` függvényében ezt nem tették meg az `OptorSim` írói, mert a fájlra a `cloneFile()` metódust hívták, ami törli a `master` jelzést a másolaton. A hibát javítottam egy `bool isMaster()` és egy másik `cloneFile(bool master)` függvény hozzáadásával.

Holtpont – Eseményvezérelt Grid idő

A `WaitObject` implementáció súlyosan hibás volt, mert az objektumokat nem referencia, hanem érték szerint azonosította, így holtpontot idézett elő. Javítottam egy `WaitObjectMap` osztály bevezetésével. Az eredeti `OptorSim`-ben valószínűleg azért nem okozott ez holtpontot, mert kellően bonyolult objektumokon szinkronizáltak (`gtWait()`-tel, stb.), így amikor az objektumok referencia szerint különböztek akkor érték szerint is.

Holtpont – Kilépés

A `SkelResourceBroker` osztály `void standBy()` metódusa kilépésnél gyakran holtpontot okozott. A szükséges elágazás hozzáadásával a hibát javítottam.

2.5. Mérési eredmények

Hogy össze tudjuk hasonlítani a különféle Grid ütemezők teljesítményét és viszonyát az új ütemezőhöz, az `examples/edg_testbed_*` Grid felépítésre lefuttattam a szimulációt az összes ütemezővel. A méréseket minden esetben 10-szer ismételtem, a legjobb és legrosszabb eredményt eldobtam, a maradékokat átlagoltam. A mért tulajdonságok a *feladatok átlagos futási ideje* és az *effektív hálózat használat* (ENU, ld. a felhasználói dokumentációban). Az átlagos futási időbe beleértjük a sorbanállási időt is.

A mérések eredményeit a 2.5. táblázat, grafikusan pedig a 2.13. és a 2.14. ábrák foglalják össze. Az ott szereplő jelölések magyarázata:

Teszteset azonosító	Paraméterek	Teljesülés?

T-01	Grid felépítés: `edg_testbed_*`
	`number.jobs = 500`
	`users = 1`
	`access.pattern.generator = 1`
	`fill.all.sites = no`
	`job.delay = 25000`
	`random.seed = no`
	`background.bandwidth = no`
	`initial.file.distribution = 8`
		igen

T-02	Grid felépítés: `edg_testbed_*`
	`number.jobs = 500`
	`users = 2`
	`access.pattern.generator = 2`
	`fill.all.sites = no`
	`job.delay = 25000`
	`random.seed = no`
	`background.bandwidth = no`
	`initial.file.distribution = 8`
		igen

T-03	Grid felépítés: `edg_testbed_*`
	`number.jobs = 1500`
	`users = 3`
	`access.pattern.generator = 3`
	`fill.all.sites = yes`
	`job.delay = 5000`
	`random.seed = yes`
	`background.bandwidth = yes`
	`initial.file.distribution = 8`
		igen

T-04	Grid felépítés: `edg_testbed_*`
	`number.jobs = 500`
	`users = 1`
	`access.pattern.generator = 1`
	`fill.all.sites = no`
	`job.delay = 25000`
	`random.seed = no`
	`background.bandwidth = no`
	`gui = yes`
	`initial.file.distribution = random`
		igen

2.3. táblázat. Néhány lefuttatott teszteset

Követelmény azonosító	Teljesítés mértéke	Megjegyzés
K-OptorExt	100%	
K-OptorCnf	100%	`scheduler=6` új lehetőség.
K-OptorGUI	100%	A hozzáadott komponensek munkája szintén bele van számítva a grafikus felületen.
K-OptorSta	100%	Az új komponensek pontosan frissítik a statisztikákat.
K-CEAHost	100%	A CE-k ágenst fogadó és futtató képessége szimulálva van. `ComputingElementWithAgentHost`
K-CEAgent	100%	`JobManagerAgent`
K-SEAHost	100%	Az SE-k ágenst fogadó és futtató képessége szimulálva van.
K-SEAgent	100%	`FileAccessAgent`
K-AgentRB	100%	`AgentAwareStaticDataResourceBroker`
K-AgentJH	100%	`AgentAwareJobHandler`
K-Deadline	100%	A program és a mérések időre elkészültek.
K-Robust	100%	A holtpontokat kiköszöböltem, a megoldást a legkülönbözőbb paraméterválasztásokra teszteltem.
K-Java1.4	100%	A program fut Java 1.4 alatt, nem használtam ki a Java 5 új szolgáltatásait.
K-Generic	80%	A hozzáadott komponensek nem feltételeznek semmit a Grid felépítéséről, pl. nem feltételezik, hogy minden site-on csak egy SE és/vagy CE van. Mindazonáltal általánosabb ágenshoszt-ágens szimulációt is el lehetne képzelni.

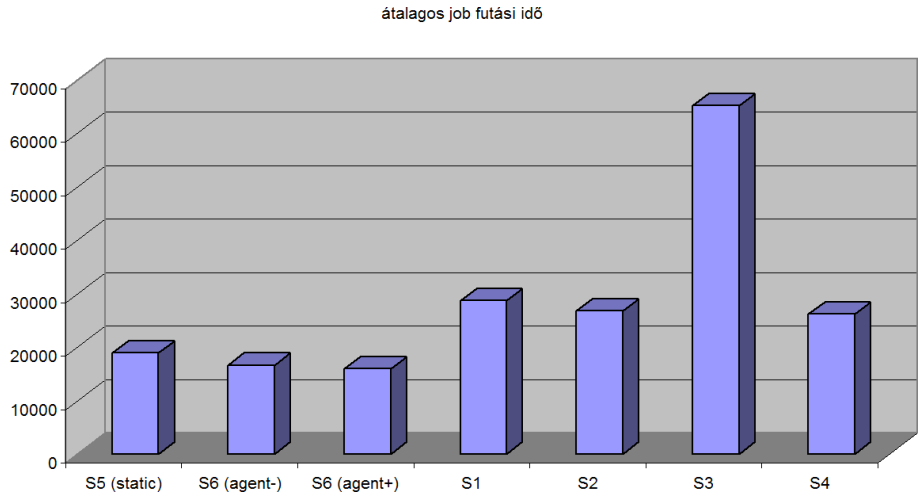
2.4. táblázat. A követelmények teljesülése

Ütemező	átlagos job futási idő	ENU
S5 (static)	18909	0.3168508625
S6 (agent-)	16612	0.1493267350
S6 (agent+)	15978	0.1357024950
S1	28721	0.6417047725
S2	26762	0.5918286400
S3	65186	0.3641069650
S4	26232	0.2156377050

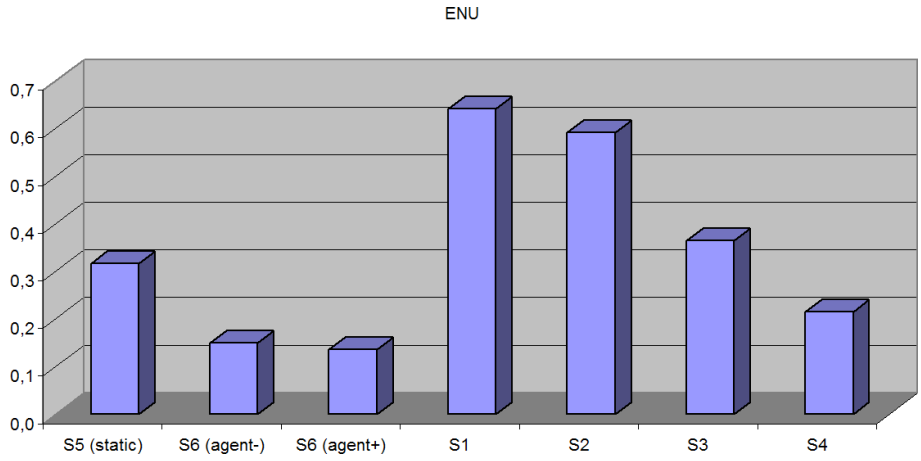
2.5. táblázat. Mérési eredmények

Jelölés	Ütemező	Megjegyzés
S1	RandomCEResourceBroker	`OptorSim` beépített.
S2	QueueLengthResourceBroker	`OptorSim` beépített.
S3	AccessCostResourceBroker	`OptorSim` beépített.
S4	CombinedCostResourceBroker	`OptorSim` beépített.
S5	StaticDataResourceBroker	A [LKUH05] cikk ütemezője.
S6 (agent-)	AgentAwareStaticDataResourceBroker	Az új ütemező a „recent” rendszer nélkül.
S6 (agent+)	AgentAwareStaticDataResourceBroker	Az új ütemező a „recent” rendszerrel.

Az új ütemező a fenti Grid felépítésre és feladat-összeállításra több, mint 15%-os javulást hozott az eddigi legjobb ütemezőhöz képest az átlagos futási időt tekintve.



2.13. ábra. *Mérési eredmények: átlagos futási idő.* Az átlagos futási időbe beleértjük a sorbanállási időt is.



2.14. ábra. *Mérési eredmények: ENU.* Az ENU definícióját ld. a felhasználói dokumentációban.

2.6. Értékelés

Ebben a nagyprogramban egy újszerű Grid optimalizációs eljárást szimuláltam. Ennek érdekében kb. 1300 kódsorral bővítettem ki egy meglévő Grid szimulátort, az `OptorSim`-et ágensek szimulációjával, és az ezekhez szorosan kapcsolódó új ütemező algoritmussal. A feladat meglehetősen komplex volt, hiszen egy már amúgy is bonyolult, nehezen áttekinthető szimulátort kellett kiegészíteni önnálló „öntudattal rendelkező" elemekkel (*ágensekkel*), és egy ütemezővel, amik egységet alkotva és együttműködve a rendszer többi részével optimalizálják a feladatok befejeződési idejét. A szimuláció erősen párhuzamos jellege tovább növelte a feladat komplexitását.

Az erőfeszítés nem volt hiábavaló: a mérési eredmények igazolják az eljárás létjogosultságát, és egy továbbfejleszthető, a céljainkat jól szolgáló rendszert kaptunk a módosításokkal. Mi több, még az `OptorSim` néhány súlyos és igen nehezen észrevehető hibájának kijavítására is sor kerülhetett. A feladatot így eredményesen befejezettnek tekinthetjük.

A. függelék

A CD tartalma

- `Forraskod/` könyvtár: az ágensekkel és új ütemezővel kibővített `OptorSim` forráskódját tartalmazó könyvtár
  - `optrosim-ckos-source.tar.gz` - a forráskódot tartalmazó tömörített fájl
- `Binaris/` könyvtár: az ágensekkel és új ütemezővel kibővített `OptorSim` futtatható állományait tartalmazó könyvtár
  - `OptorSim.jar` - a Java bináris állomány
  - `examples/` - a minta konfigurációs állományokat tartalmazó alkönyvtár
  - `run.sh` - a Linux alatti futtatást elősegítő parancsfájl
  - `run.bat` - a Windows alatti futtatást elősegítő parancsfájl
- `Dokumentacio/` könyvtár: jelen dokumentáció és az algoritmusokat leíró cikkek
  - `Cikkek/` - a Java bináris állomány
- `Egyeb/` könyvtár: kiegészítő anyagok
  - `optorsim-orig/` könyvtár: az eredeti `OptorSim` verzió
  - `optorsim-elte/` könyvtár: a Megrendelők `OptorSim` verziója
  - `Java/` könyvtár: Java JRE 5.0 Update 7 (Java futtatókörnyezet) és JDK 5.0 Update 7 (fejlesztői környezet)
  - `IntelliJ IDEA/` könyvtár: JetBrains IntelliJ Idea fejlesztői környezet próbaváltozat
  - `Violet/` könyvtár: a Violet UML rajzprogram
  - `Dia/` könyvtár: a Dia rajzprogram
  - `7-Zip/` könyvtár: ingyenes tömörítőprogram Windows-ra

Irodalomjegyzék

[CCSF+04] Davig G. Cameron, Ruben Carvajal-Schiaffino, Jamie Ferguson, A. Paul Millar, Caitriana Nicholson, Kurt Stockinger, and Floriano Zini. Optorsim v2.0 installation and user guide. http://edg-wp2.web.cern.ch/edg-wp2/optimization/downloads/v2_0/edg-optorsim/doc/userguide-optorsim.ps. 2004. Nov.

[Dia] Dia rajzprogram. Weboldal, `http://www.gnome.org/projects/dia/`.

[Hor] Cay Horstmann. Violet uml szerkesztő. Weboldal, `http://www.horstmann.com/violet/`.

[Ide] Intellij idea 5.1 fejlesztői környezet. Weboldal, `http://www.jetbrains.com/idea/`.

[LKUH05] László Csaba Lőrincz, Tamás Kozsik, Attila Ulbert, and Zoltán Horváth. A method for job scheduling in grid based on job execution status. *Multiagent and Grid Systems - An International Journal 4 (MAGS)*, 1(3):197–208, 2005.

[LUKH06] László Csaba Lőrincz, Attila Ulbert, Tamás Kozsik, and Zoltán Horváth. Towards job scheduling in the next generation of grid. Publikálás alatt. 2006.

[Opt05] Optorsim v2.0.0. Web oldal. 2005. márc. 31., `http://edg-wp2.web.cern.ch/edg-wp2/optimization/optorsim.html`.

Ábrák jegyzéke

Táblázatok jegyzéke