Elosztott adatelérés GLOBUS alapú technológiával

Várdai Márk

Témavezető: Horváth Zoltán

Budapest, 2004

Project: IKTA-00064/2003 A magyar SuperGrid és KlaszterGrid rendszerek felhasználó orientált egységesítése.

Tartalomjegyzék

1. fejezet

Bevezetés

A *Grid* napjaink egy fontos és nagyon szerteágazó kutatási területe. A *Grid* célja inhomogén, földrajzilag különböző helyen lévő számítógépek erőforrásainak, processzorainak, háttértárjainak, szolgáltatásainak optimális, egységes és univerzális felhasználása, oly módon, hogy a felhasználó egy nagy virtuális egésznek látja az előbb felsorolt eszközöket, nem foglalkozván a replikákkal, adatok és processzorok közelségével és még sok más későbbiekben tárgyalt kérdéssel.

Napjainkban a *Grid* nem csak tudományos szempontból érdekes, hanem fontos gyakorlati jelentősége is van, az élet számtalan területén alkalmazható. Régebben főleg a számításigényre fektették a hangsúlyt, mint például prímszám keresés, kulcsok törése. Ezek a feladatok főleg a párhuzamos számolást, számítást igényelték, de nem voltak *adat-intenzívek.* Az utóbbi években az alkalmazások körében áthelyeződött a hangsúly az adat-intenzív számításokra, azaz ahol a számítás nagy méretű adatokon folyik. Napjaink *adatbázis-grid* igénye is két nagy csoportba sorolható:

Együttműködés Az adat nagyon sok helyen keletkezik és az a fontos, hogy egységesen lehessen hozzáférni és feldolgozni.

Szétoszthatóság Nagyi méretű adat, ami kevés helyen keletkezik. A feladatot az adat szempontjából erősen párhuzamosítani lehet és így szét lehet osztani a Grid pontjai között.

A diploma munka célja, az előbbiekben említett adatbázis-grid elméleti vizsgálata. Ezen túl az adatbázis-grid egy konkrét implementációjának, a *GLOBUS*-ra épülő *OGSA-DAI*-nak, vizsgálata példaprogramon keresztül.

2. fejezet

Áttekintés

Ebben a fejezetben néhány olyan technológiát mutatunk be a teljesség igénye nélkül, melynek ismerete elengedhetetlenül szükséges a későbbi elméleti és gyakorlati részek megértéshez.

2.1. XML

Az XML [1](Extensible Markup Language – Kiterjesztett jelölő nyelv) a World Wide Web Consortium (W^3C) által kifejlesztett dokumentum tárolási formátum. Napjainkban a különböző elektronikus dokumentumok mellett a szöveges dokumentumok száma jelentős. Az XML ideális adatformátum a különböző médiákon terjesztett, megjelenített strukturált vagy féligstruktúrált szövegek számára. A megjelenítésen túl, a különböző programok és rendszerek közötti kommunikációban, információcserében is nagy szerepe van. Fontos elvárás, hogy önleíró legyen, azaz a kliens programok is értelmezni tudják a kapott struktúrát, illetve végre tudjanak hajtani utasításokat önállóan, újabb kapcsolat kialakítás nélkül. Az XML dokumentum speciális jelölő elemeket tartalmaz, melyek a dokumentum egyes részeit határolják el. Van külön logikai és fizikai szerkezete. A logikai szerkezet során a dokumentumot részekre, alrészekre lehet bontani, és így *elemeket* hozunk létre. A fizikai felépítés során *egyedeket* hozunk létre, melyeket külön tárolunk. Például egy könyv minden fejezete leírható elemekkel, melyek tartalmazhatnak további táblázat, képleíró elemeket, de az újra felhasználandó kép adatai, bekezdései külön fájl egyedben tárolandók. Minden XML dokumentumnak meg kell felelnie bizonyos megkötéseknek. Igazából az XML[XML] egy köztes nyelv, mivel nincsen előre definiált elemkészlete. A felhasználó saját maga definiálhat elemeket. Egy adott dokumentumhoz megadhatjuk a felhasználható

[1] Az XML-ről bővebben a http://www.w3c.org/XML címen olvasható.

elemeket, a hozzájuk kapcsolódó szabályokkal. Ez a DTD (Document Type Definition – Dokumentumtípus-meghatározás).

Az XML elemek általában közrefogják az azonosítandó objektumot *nyitó* és *záró* elemekkel. A nyitó-, záró elemek és a közre zárt objektum együtt jelent egy elemi egységet. Az XML-elemek tartalmazhatnak beágyazva további XML-elemeket, és az egész dokumentumot egyetlen XML-elembe kell beágyazni. Az előbbiek figyelembe vételével a dokumentum fa szerkezettel ábrázolható. Egy elemnek tetszőleges számú leszármazottja (beágyazott eleme) lehet.

Lehetőség van arra, hogy egy elem többlet információt is hordozzon a tartalmáról. Ezeket az információkat az elemhez tartozó attribútumokban tároljuk.

2.1.1. DTD

Ez a fogalom az SGML-ből származik. Az XML-ben nem szükséges használni, de hasznos. A dokumentum érvényességét ellenőrizhetjük, különböző kiadványokat, újságokat, műszaki leírásokat egységes formára hozhatunk. Hasonlít a szövegszerkesztőkben megtalálható sablonokra, de azoknál jóval többet tud. Meghatározza a felhasználható elemeket és az egymáshoz fűződő viszonyukat. Ezzel a dokumentum hierarchiáját és szemcsézettségét, tagoltságát alakíthatjuk ki. A DTD olyan meghatározásokat tartalmaz, amelyekből a dokumentum szerkezeti fa építhető fel. Meghatározza a jellemzők nevét és azt is, hogy milyen elemekben fordulhatnak elő. Minden jellemzőhöz megadható, hogy milyen típusú értéket vehet fel, illetve meghatározható, hogy kötelező vagy opcionális. Meghatározhat sorrendiséget az egyes elemek között, valamint előfordulásuk számát. A jelölések ismertek lehetnek a HTML-ből, illetve a formális nyelvekből.

Elem deklaráció

```
<!-- ELEMENT név ... -->
```

A ... helyén írhatjuk le, hogy a név elem mit határoz meg. Lehet üres (EMPTY), csak szöveget tartalmazó (#PCDATA), tartalmazhat gyermek elemeket vagy keverve gyermek elemeket és szöveget.

```
<!-- ELEMENT konyv (bekezdes?, fejezet*) -->
```

Az elemek előfordulását és mennyiségét az alábbi jelek határozzák meg.

Mennyiség

- ? határozza meg egy elemről, hogy opcionális-e vagy sem.
- * határozza meg, hogy egy elem tetszőlegesi számú előfordulású.
- + határozza meg, hogy egy elemnek tetszőleges számú, de legalább egy előfordulása van.

Sorrendiség

A sorrendiség meghatározásához a következő jelek használatosak.

- , határozza meg a sorozat összekapcsolást (szekvenciát). Ilyenkor egymást követő sorrendben kell lennie az elemeknek.
- | határozza meg az elágazás összekapcsolót. Ebben az esetben valamelyik elemet kell választani, de csak az egyiket.

Elemcsoport

Gyakran előfordul, hogy összetett elemet szeretnénk definiálni. Ilyenkor zárójelek közé kell írni a kifejezést. A zárójelre ugyanúgy alkalmazhatók a *Mennyiségi* és *Sorrendiségi* jelzők, mint bármelyik elemre.

Attribútum

Egy elemhez szükség lehet kiegészítő információkra is, melyeket nem akarunk beágyazni. Erre szolgálnak az attribútumok.

```
<!-- ATTLIST element_nev attr_nev1 attr_tipus1
                        attr_nev2 attr_tipus2-->
```

Egy elemhez több attribútum deklaráció is tartozhat, ilyenkor összegződnek a definíciók. Az előre definiált típusok mellett megadható vagylagos típus is, a zárójel és a | megfelelő használatával. Néhány típusnak létezik felsorolás jellegű típusa is, ilyenkor szóközökkel adhatjuk meg az értéküket az XML adatban. Az attribútumok előfordulása is szabályozható.

- Kötelező : #REQUIRED
- Elhagyható : #IMPLIED

Alapérték is rendelhető az egyes attribútumokhoz, kivéve az előbb említett #REQUIRED és #IMPLIED kulcsszóval megjelöltekhez.

Névterek

Előfordulhat, hogy különböző tartományokba, dtd-kbe tartozó részekből szeretnénk XML dokumentumot összeállítani. Erre ad lehetőséget a névtér (namespace). A névterek használata jól definiált. Egyértelműen eldönti, hogy egy adott elem melyik névtérbe tartozik, és így a duplikációt feloldja. Névtér azonosítás URL-eken keresztül történik. Nem kell az internethez kapcsolódni, csak megkülönböztetésre szolgál, mindemellett magában hordozhatja azt az információt is, hogy hol érdemes a névtérhez kapcsolódó specifikációnak részletesebben utánajárni. Igaz, hogy az URL-ek azonosításra hasznosak, egyedi megkülönböztetésre nem alkalmasak, ezért az alábbi módon kell használni.

```
<X:valami xmlns:X="http://valami.web.cim/egyeb/">
    <X:proba/>
</X:valami>
```

Az xmlns kulcsszó vezeti be a névtér használatát. Állhat önmagában vagy : -tal elválasztott, a későbbiekben prefixnek hívott taggal. Ezt követi az egyenlőségjellel megadott egyedi azonosító, amit a prefix helyettesít az XML dokumentumban az egyes elemek előtt. Annak érdekében, hogy a dokumentumok érvényességét ellenőrizni lehessen, a DTD-kben is használhatóak.

2.1.2. XML-SCHEMA

Mint már láttuk az XML alkalmas arra, hogy a fejlesztők tetszőleges struktúrákat hozhassanak létre adataik tárolására és megosztására. Az XML-Schema[XML-Schema] nem más, mint megszorítások gyűjteménye, amik szabályok vagy struktúra modellekkel vannak kifejezve. Ezekkel a megszorításokkal lehet XML dokumentum példányokat elfogadtatni. A schemák felhasználása szinte korlátlan lehetőséget rejt magában, de napjaink alkalmazásainak többségét az alábbi kategóriákba lehet besorolni:

Validáció Érvényesség ellenőrzés. Rengeteg oka lehet annak, hogy ellenőrizünk egy XML dokumentumot. Például amikor kapunk egy dokumentumot és a rendszerbe akarjuk importálni, kézzel szerkesztettük és meg akarunk győződni arról, hogy helyes dokumentumot készítettünk, de még sorolhatnánk. A validációnak több szintje van.

Struktúra ellenőrzés Amikor arra vagyunk kíváncsiak, hogy a rendelkezésünkre álló XML dokumentum elemei és attribútumai megfelelnek-e bizonyos megkötéseknek, de nem vagyunk kíváncsiak az általuk hordozott adatok típusára, érvényességére.

Adat validáció Amikor arra is kíváncsiak vagyunk, hogy a hordozott adat megfelel-e a típusban leírtaknak, az előzőeken felül.

Dokumentáció A schemákat a validáció mellett gyakran használják XML szótárként is, így magyarázván meg az egyes elemek és attribútumok jelentését, hordozott tartalmát. Napjainkban szinte minden XML dokumentum mellé csatolnak schemát is.

Vezetett szerkesztés Sok XML editor használ schemákat a felhasználók megsegítésére, azaz az adott környezetben mit tehet meg a felhasználó, mit szúrhat éppen be és mit nem tehet meg. A W3C egyik új API-ja az *Abstract Schemas* arra szolgál majd, hogy lefedje a DTD-ket és schemákat, segítvén a felhasználókat a guided editing alkalmazásokon keresztül, válaszolván az előbb feltett kérdésekre.

A XML 1.0-ás szabvány, sok más mellett magában foglalja a DTD-t, azaz XML dokumentumok struktúrájának definícióját, de ez a leírás napjainkra kevésnek bizonyult. Az XML schema típusokon keresztül, objektum orientált szemmel próbálja a kérdést megközelíteni.

2.2. ODMG

Az Object Database Management Group (*ODMG*) létrehozott egy szabványt – ODMG-93 néven ismert – az Objektum Orientált Adatbázisokhoz. A szabvány a platformfüggetlenséget helyezte előtérbe. Az alábbi fő kategóriákra bontható:

Object Model Az objektum modell.

Object Definition Language Az objektumok definiálására szolgáló nyelv.

Object Query Language Az objektumok lekérdezésére szolgáló nyelv.

C++ Language Binding Kapcsolódás a C++ programozási nyelvhez.

Java Language Binding Kapcsolódás a Java programozási nyelvhez.

Az ODMG objektum modellje az OMG objektum modelljének kiterjesztése. Az OMG objektum modellje a megszokott osztályfogalom, az attribútumokkal és metódusaival, az örökléssel és specializációval. Az ODMG objektum modellje megenged kapcsolatot az egyes osztályok objektum példányai között, támogatja az osztályok gyűjteményét *(collection classes)*, valamint definiál alaptípus osztályokat, mint dátum, idő és sztring. Egy létező implementáció, az O_2, teljesen kompatibilis a szabvány alábbi részeivel:

- Object Model
- Object Query Language
- C++ Language Binding

Az OGSA-DAI DQP része (3.2.8. fejezet)[DQP] csak az Object Query Language-et használja, ezért csak ezt mutatjuk be.

2.2.1. OQL

A OQL[2] (Object Query Language) egy SQL jellegű, objektum orientált deklaratív lekérdező nyelv, mely egyszerű hozzáférést biztosít az objektumokhoz. A szintaxisa nagyon hasonlít az SQL-hez.

$$\text{SELECT} \ldots \text{FROM} \ldots \text{WHERE} \ldots \text{ORDER BY}.$$

A lekérdezések objektumokat használnak és az attribútumaikat (adattagjaikat). Az egyes részek jelentése a következő.

[2]Az ODMG-ről és OQL-ről bővebben a http://www.odmg.org/-on lehet olvasni.

SELECT Megmondja, hogy mely objektumokat, objektum-tagokat adja meg.

FROM Specifikálja, hogy milyen halmazból keressen. A DQP-beli (3.2.8. fejezet) lekérdezésekben az objektum-osztály azonosítása *variable* ***in*** *table_ name* formában történik.

WHERE Specifikálja, hogy milyen tulajdonsággal rendelkező egyedeket adjon meg.

ORDER BY Specifikálja, hogy melyik objektum-tag szerint legyen rendezve. Nem kötelező megadni.

Többféle az SQL-ből már megszokott lekérdezésre van lehetőség.

- Egyszerű

```
select x.(fieldname) from x in (table)
```

- Join

```
select {[x,y].(fieldname)} from x,y in (table)
where x.(fieldname)=y.(fieldname)
```

- Beágyazott

```
select x.(fieldname) from
(select y from y in (table)
    where and{y.(fieldname) op (value)}
as x
```

- Művelet hívás

```
select (method)({x.(fieldname)},{value})
from x in (table)
```

- Aggregátumok

```
count(query), sum(query), max(query),
min(query), avg(query)
```

- Beágyazatlan

```
select x.collectionattribute from x in (table)

select y from (x in table), y in x.collectionattribute

select y.structattribute
from (x in table), y in x.collection_of_structs_attribute
```

Ezek a parancsok nem fedik le a teljes OQL szabványt, csak azokat ismertettük, melyeket a DQP használ.

3. fejezet

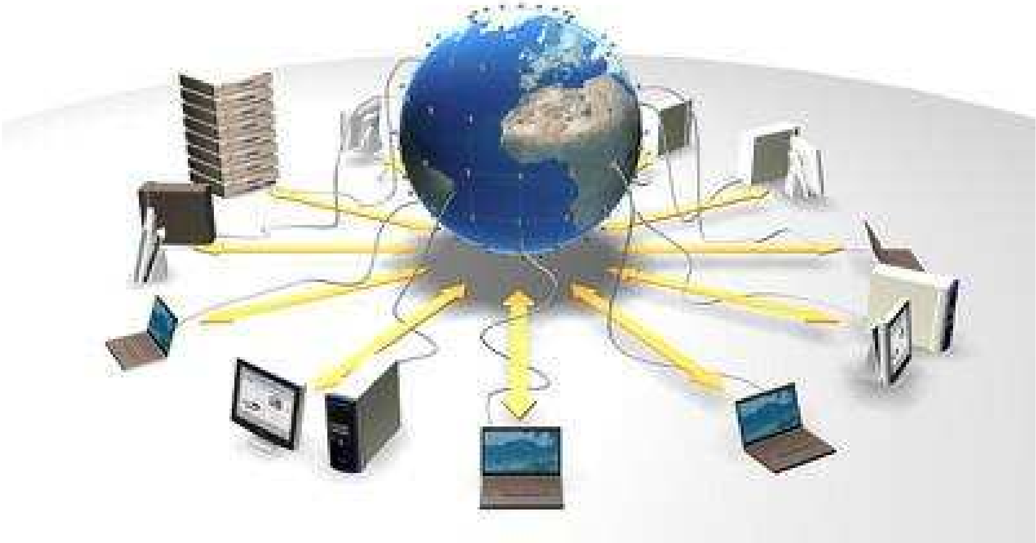
GRID

A Grid, néhányak fordítása szerint rácshálózat, hasonló utat járt be, mint anno a www. Ez utóbbi az információ elérést forradalmasította, míg az előző korlátlanul tágítja az információ feldolgozás lehetőségeit. A Grid lényege a földrajzilag elosztott számítógépek összekapcsolása és az ehhez szükséges futtató/kezelő middleware kifejlesztése. A Web az elosztott információhoz, a Grid az elosztott feldolgozási és tárkapacitáshoz biztosít hozzáférést. A Grid a számítástechnikai szabad kapacitások hatékonyabb hozzáférését célozta meg. A Grid leginkább egy hatalmas virtuális számítógépes rendszer, szerverek együttese, tárrendszerek és hálózatok összessége. Az adatfeldolgozás és tárolás, a sávszélesség a legfontosabb megoldandó problémák. A rácshálózatba kapcsolt rendszerek akár ugyanabban a szobában, akár több kontinensen is lehetnek, különböző hardware-platformokon és operációs rendszerek alatt futnak, más és más szervezetek a tulajdonosok.

A Globális Grid Fórum (GGF) által kidolgozott architektúra, az OGSA (Open Grid Services Architecture) meghatározza a Grid-szolgáltatásokat, a mindenre kiterjedő struktúrát, illetve, hogy milyen szolgáltatások biztosítottak a Grid-környezetekben. Az OGSA által leírt elképzelések formális specifikációja, az OGSI (Open Grid Services Infrastructure) megadja a szolgáltatások működését definiáló alapelemsort (set of service primitives).

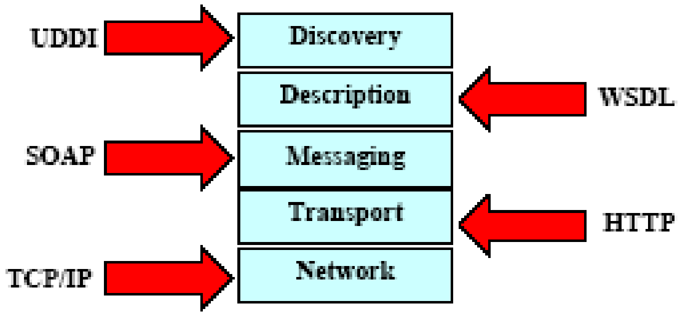
3.1. GLOBUS

A Globus[IBMRED] az Open Grid Service Infrastructure egy referencia implementációja. Amellett, hogy az aktuális specifikációt implementálja, a régebbi Globus verziókkal is megtartja a kompatibilitást. Java és C alapokon nyugszik. A Java alapú eszköztár egy Grid Service hosting container felületet nyújt, ami egy hagyományos Web szerver hosting container környezet-



3.1. ábra. Grid

ben is futhat. Az eszköztár GWSDL és WSDL dokumentumokkal írja le a Grid Service portType-okat. A GWSDL a WSDL (Web Service Description Language) dokumentum kiterjesztése, ami egy XML Schema, mely olyan dokumentumokat validál, amik leírják a Web Service-t és így vele együtt a szolgáltatásait is. A WSDL-ben interfész kifejezésekkel vagy másképpen portType-okkal írják le egy Web Service szolgáltatásait. A portType egy interfész gyüjtő, ami bizonyos szempontok alapján csoportosítja az egyes műveleteket. Például a regisztrációs és a lekérdező műveletek a Registration portType illetve a Query portType alá kerülhetnek. A WSDL dokumentumban határozzák meg a szolgáltatás konkrét fizikai portszámát, használandó protokol és adattovábbítás módját, a szolgáltatás által használt üzeneteket. A Grid egyes szolgáltatásai SOAP-on (Simple Object Access Protocol) keresztül kommunikálnak egymással. A 3.2 ábrán látható a Web Service logikai felépítése.



3.2. ábra. Web Szolgáltatások szintjei

3.2. OGSA-DAI

3.2.1. Általában

Mint ahogy már említettük, a Grid alapvető értéke a virtualizáció, vagy másképpen az átlátszó hozzáférés a szétosztott számítási erőforrásokhoz. Az adat intenzív alkalmazások számára a virtualizációnak a szétosztott adatforrásokat is magában kell foglalnia. Kétfajta átlátszóságot kell az adatbázisokkal kapcsolatban megemlítenünk, melyek tovább bonthatóak:

- Átlátszóság az adathozzáféréshez.
- Átlátszóság az adatfeldolgozáshoz.

Adathozzáférés átlátszósága

Heterogén átlátszóság Egy alkalmazásnak az adat elérés folyamán függetlennek kell lennie az adatforrás implementációjától. Ennek érdekében a programoknak logikai schémákkal kell dolgozniuk.

Név átlátszóság Az adatok elérésének nem direkt címzésen keresztül, hanem *logikai tartományokon* keresztül kell történnie, minősítve a keresett objektum attribútumaival. Ez az elvárás értelemszerűen magával hozza az elhelyezkedés és másolat átlátszóságát. Mivel az alkalmazás nem direkt címezi az adatforrást, ezért nyugodtan lehet az adatbázist másolni, átmenetileg tárolni, anélkül, hogy az alkalmazást megváltoztatnánk.

Adatfeldolgozás átlátszósága

Párhuzamosság Egy adatokat feldolgozó alkalmazásnak automatikusan hasznosítania kell a Grid csúcsai feletti párhuzamos feldolgozást. Az alkalmazásnak az egyes részfolyamatok közötti függőséget kell definiálnia egy deklaratív folyamatleíró nyelv segítségével. A *folyamat koordinációs szolgáltatás* automatikusan párhuzamosítja ezek alapján a folyamatokat.

Szétosztottság Az alkalmazásnak karban kell tartania szétosztott adatokat is egységes formában.

3.2.2. Napjaink adatbázis technológiái

Az alábbiakban megvizsgáljuk napjaink adatbázis kezelőit a fenti átlátszóságokat figyelembe véve.

File rendszerek A legegyszerűbb adat hozzáférési interfész amivel a legtöbb alkalmazás kapcsolatba kerül az egyszerű file műveletek. A szétosztott fájlrendszerek kiterjesztik az egyszerű nyitás, írás, olvasás, bezárás műveleteket *helyátlátszósággal*, azaz a logikai fileneveket összekapcsolják a fizikai elhelyezkedéssel, de minden egyéb adatfeldolgozást le kell programozni.

DBMS Az adatbáziskezelők egy magasabb szintű interfészt nyújtanak az adatok karbantartásához. Az adatok fizikai szervezését elrejtik a felhasználó elől. Az alkalmazások számára egy magasszintű deklaratív nyelvet, az SQL-t, nyújtják. Az alkalmazásnak csak azt kell megmondani, hogy milyen feladatot akarnak végrehajtani, azt már nem, hogy hogyan, ez már az adatbázismotor feladata, hogy kikeresse a lehetséges megoldások közül a legoptimálisabbat. Az alkalmazások számára többféle lehetőség van az adatbázisok elérésére.

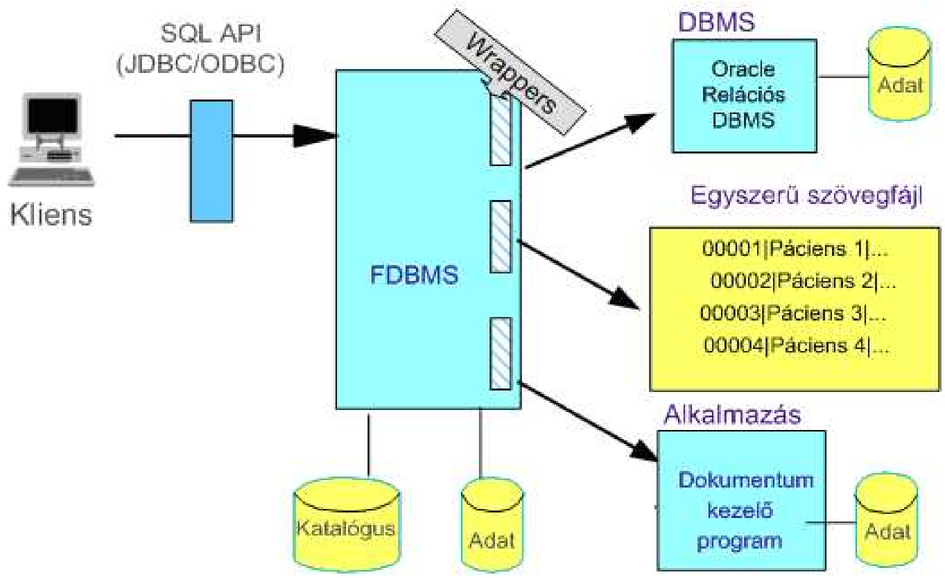
- JDBC
- ODBC
- ADO.NET

Ezek az interfészek megvalósítják valamilyen módon a heterogenitás átlátszóságot. Kezelnek néhány típust, de nem valósítják meg a Griddel szemben elvárt más átlátszósági kritériumokat, és kevés adatforrást kezelnek.

Federated DBMS A Federated DBMS egy middleware, ami átlátszó hozzáférést biztosít számos heterogén és szétosztott adatbázis között. Az alkalmazások a már említett módon kommunikálhatnak a FDBMS-sel. Az FDBMS az egyes adatforrásokkal úgynevezett *wrapper* modulokon keresztül kommunikál. Amikor jön egy kérés, az FDBMS meghatározza az adatforrást és elkészít egy végrehajtási tervet. A terv a következőkből áll:

- A kérés feldarabolása a megfelelő adatforrásokra.
- Későbbi adatfeldolgozás (szűrés, aggregálás, összefésülés) időzítése.

A FDBMS lokálisan tárolja az adatokat a későbbi feldolgozás érdekében. Az adatok megszerzéséhez a wrapper töltődik be. Ez egy dinamikusan betölthető megosztott könyvtár. A process, ami adatot szeretne elérni a wrapperen keresztül, regisztrálja magát a rendszer katalógusban. A 3.3. ábrán láthatunk egy példát elosztott adatbázis rendszerre.

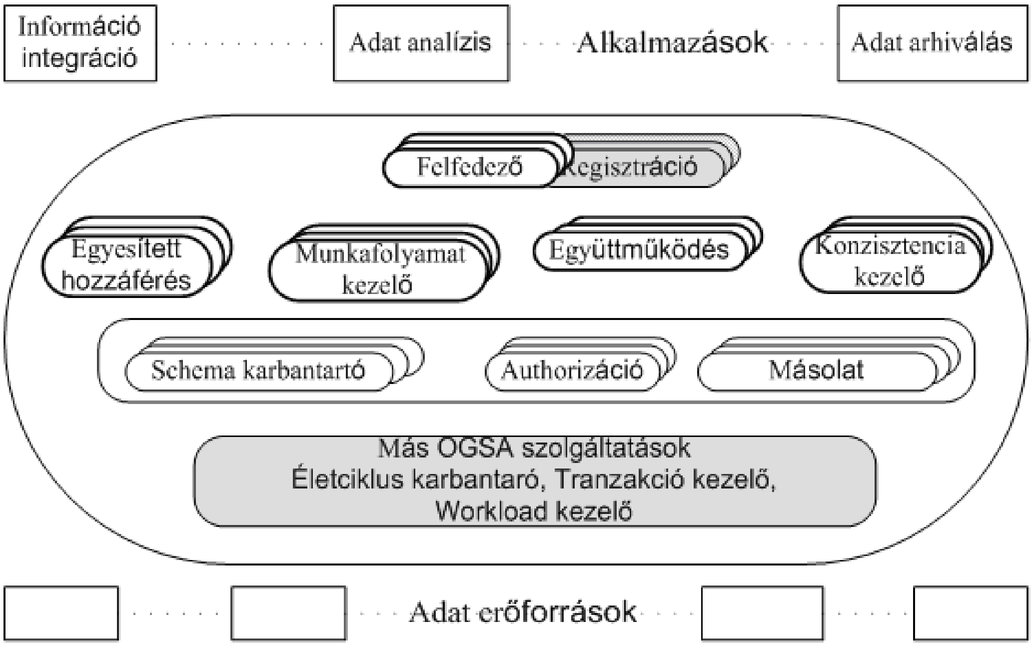


3.3. ábra. Elosztott adatbázis rendszerre példa

Összegezve az elosztott adatbázis rendszerek elég jó heterogenitás és szétosztottság átlátszóságot nyújtanak. A fő korlátja a rendszernek, hogy az alkalmazásnak az elosztott lekérdezésben explicit meg kell határoznia az adatforrást, ami azt jelenti, hogy egy új adatforrás bevezetése nagyon költséges feladat, mivel az alkalmazás újra kell írni.

3.2.3. Grid Data Virtualizációs szolgáltatások

A 3.4. ábra mutatja meg az OGSA-DAI elképzelést az adatszolgáltatás architektúráról. Az ábrán láthatjuk az adatforrások és a Grid-es alkalmazások



3.4. ábra. Grid adat szolgáltatások

közötti virtualizációs szolgáltatások szintjeit. Ezek a szintek különböző szempontok szerint virtuálissá teszik a Gridet a felhasználó számára. Az adatbázisgrid alapvető feltételezése, hogy az alkalmazások többsége ezeken a virtualziációs felületeken keresztül éri el az adatokat, meghagyván a "pass-thru" lehetőséget. A "pass-thru" azt jelenti, hogy az alkalmazás átmenetileg tárolja egy korábbi, virtualizációs szolgáltatástól jövő választ, majd közvetlenül az alsóbb rétegekhez továbbítja az új kérést. Az egyes szolgáltatások részletes vizsgálata előtt a 3.5. ábrán láthatunk egy összefoglalást arról, hogy melyik szolgáltatás milyen átlátszóságot valósít meg, illetve kellene megvalósítania. Mint ahogy az ábrán is látszik, a szolgáltatások két fő csoportra bonthatóak:

- Alap
- Kiegészítő

3.2.4. Alap szolgáltatások

Discovery (Felfedező)

Ez a szolgáltatás valósítja meg az adatok név- és előfordulás átlátszóságát. Így az alkalmazásoknak elegendő bizonyos adattartományokra hivatkozniuk,

	Átlátszóság			
	Heterogenitás	**Szétosztottság**	**Név**	**Párhuzamosság**
Alap Szolgáltatások				
Felfedező			Lefordítja a logikai tartományokat az aktuális adatbázisokra. Kiválasztja a legmegfelelőbb helyet és másolatot.	
Egységesített hozzáférés	Adatformátum és implementáció független adathozzásférés	Egységes felületet nyújt az adatok hozzáféréséhez	A fájlok hely átlátszóságát is némileg megvalósítja.	
Konzisztencia karbantartó		Ugyel a szétosztott adatok konzisztenciájára		
Együttműködési		Karbantartja az adatok frissítését a többszörözött helyeken-		
Munkafolyamat karbantartó				Automatikusan párhuzamosítja a kéréseket
Kiegészítő szolg.				
Authorizációs				
Másolat			Migrálja az adatokat igény szerint a megfelelő helyre	
Schema karbantartó	Adatformátumok közötti fordítás			

3.5. ábra. Grid átlátszóság

ahelyett, hogy a pontos helyét adnák meg. Például: Keressük azokat a *betegeket*, akik bizonyos tünetekkel rendelkeznek és egy megadott korhatár alá esnek. Ebben az esetben a Discovery szolgáltatásnak a betegek logikai tartományt le kell képzenie a megfelelő fizikai adatforrásra (pl.: kórházak).

Amellett, hogy az egyes adatforrásokat nyilvántartja, más virtualizációs szolgáltatások elérhetőségét is nyilvántartja. Ha egy virtualizációs szolgáltatásnak több példánya is elérhető, akkor különböző döntő tényezők alapján, mint költség, sebesség, stb dönthet arról, hogy az adott Grides alkalmazásnal mi a legmegfelelőbb. Természetesen több felfedező szolgáltatás is lehet, amiket el lehet érni közvetlen cím alapján, vagy centralizáltan, azaz kapcsolódunk egy központi Felfedező szolgáltatáshoz, aki automatikusan átirányít a legmegfelelőbbhöz.

Az adatok frissítése és publikálása az alábbi módon történhet.

Hirdetés Ebben az esetben az adatforrás maga küldi el a schemáját a virtualizációs szolgáltatásoknak, illetve maguk a virtualizációs szolgáltatások továbbítják a Discovery-nek az általuk felügyelt adatforrásokat.

Begyűjtés Ebben az esetben a Discovery szolgáltatás bizonyos időközönként maga tesz fel kérdéseket adatforrásoknak, virtualizációs szolgáltatásoknak, hogy felderítse az elérhető egészt.

Federated Access Service (FAS, egyesített hozzáférési szolgáltatás)

Mint ahogy már a bevezetőben is említettük, sok adat-intenzív alkalmazás használja a Gridet. Ezek az alkalmazások komplex munka- és adat elemző folyamatokat határoznak meg. Ezeknek a folyamatoknak a nagy-többsége megengedi a párhuzamos végrehajtást. Időveszteséget és hatékonyság csökkentést eredményezhet ha minden egyes alkalmazáshoz külön-külön írják meg az ütemezőt. Ehelyett az alkalmazásoknak le kell írniuk a munkafolyamataik közötti függőséget, amit ez a szolgáltatás a rendelkezésre álló Grid csúcsok között szétoszt, ütemez. A szükséges adatokat rendelkezésére bocsátja, átmenetileg tárolja és biztonságos állapotba viszi a folyamatot esetleges csúcsponthibák esetén.

Consistency Management Service (konzisztencia karbantartó szolgáltatás)

Gyakran előfordul, hogy az adatokat, több csúcsponton is elhelyezik, a hatékonyabb elérhetőség kedvéért. Ebben az esetben szükség van egy szolgáltatásra, ami karbantartja, konzisztenssé teszi a különálló részeket.

Collaboration Service(Együttműködési szolgáltatás)

Az adat-intenzív szolgáltatások többsége szereti megosztani adatait más Grid csúcspontokkal, más felhasználókkal. Azért, hogy a felhasználók értesüljenek a frissítésekről illetve, hogy elkerüljék a különböző konfliktusokat, szükség van egy szolgáltatásra, ami ezt automatikusan elvégzi. Ennek a szolgáltatásnak az elsődleges feladata, hogy az egymástól független, szétosztott adatbázis frissítéseket virtualizálja, illetve ha érkezik hozzá egy kérés valamely Grid csúcsponttól, akkor legyen képes a legfrissebb változatát odaadni, esetlegesen speciális kérésre valamely régebbi változatát a kívánt adatnak.

3.2.5. Kiegészítő szolgáltatások

Az előzőekben említett szolgáltatások mind az alapszoláltatások körébe tartoztak. Definiáltak néhány kiegészítő szolgáltatást, amely teljessé teszi az adatbázisgrid elméletét.

Authorizációs szolgáltatás

Minden adatbáziskezelőnek megvan a saját azonosítási/hozzáférési mechanizmusa. Már az OGSA-ban definiáltak egy egyszerű authorizációs módszert. Ez a szolgáltatás szolgál arra, hogy létrehozza a kapcsolatot a Grid-es authorizációs és az egyes adatforrások authorizációja között.

Schema karbantartó szolgáltatás

Napjainkban nagyon sokféle adattárolási mód létezik. Nagy erőfeszítések folynak azért, hogy ezek között megteremtsék a kapcsolatot, és így az adattranszformációt a különböző formák között.

Másolat szolgáltatás

A hatékony hozzáférés érdekében készülhetnek átmeneti vagy állandó másolatok az adatok egészéről vagy részéről. Ha egy csomópont túlterhelődik, akkor a Grid workload managere kérheti a Replika szoláltatást másolat készítésére egy kevésbé terhelt csomópontra, így mentesítve az előzőt. Az egységesített hozzáférési szolgáltatás (FAS) ezeket a másolatokat virtualizálhatja a felhasználó felé.

3.2.6. Kapcsolat más OGSA szolgáltatásokkal

A fentebb leírt szolgáltatások szorosan együtt működnek azokkal a szolgáltatásokkal, amiket az OGSA definiált.

Registry Az OGSA definiált nyilvántartó helyeket, ahol számos információt lehet nyilvántartani az egyes erőforrásokról. A Felfedező szolgáltatás több információ tárolására is használhatja ezeket a helyeket. Az első, hogy megfeleltesse a logikai tartományokat és állításokat a megfelelő adatforrásnak. A másik, hogy lekérdezze és frissítse az egyes adatforrások képességeinek, tulajdonságainak listáját.

Authentikációs szolgáltatás A Grid biztonsági szolgáltatásnak rendelkeznie kell sokféle biztonsági mechanizmussal. Mivel az egyes adatbáziskezelők saját biztonsági mechanizmusokat használnak jó ötletnek tűnik központi helyen nyilvántartani és kezelni a biztonsági módozatokat.

Notification (Értesítő) szolgáltatás Az adatbázis Grid virtualizációs szolgáltatásai felhasználhatják a Grid Notification Service-ét, hogy értesítést kapjanak az alábbi adatforrás változásokról:

Schéma változás Az egységesített lekérdező előfeldolgozó számára fontos erről értesülni. Ha egy külső adatforrás schémája megváltozott, akkor az esetleges lekérdezési terveket újra ki kell értékelni. A replika szolgáltatásoknak is fontos értesülni ezekről a változásokról, ha konzisztens adatbázist akarnak fenntartani.

Állapot változás Előfordulhatnak olyan esetek, amikor megsérül az adatforrás. Ebben az esetben az egységesített hozzáférési szolgáltatásnak értesülnie kell a hibáról, hogy megfelelő választ küldhessen egy lekérdezés eredményeként. A Replika szolgáltatás számára is hasznos, ha értesül ezekről a hibákról, hiszen így lépéseket tehet annak érdekében, hogy kijavítsa a hibát, vagy folytassa a replika készítését a sérült résztől.

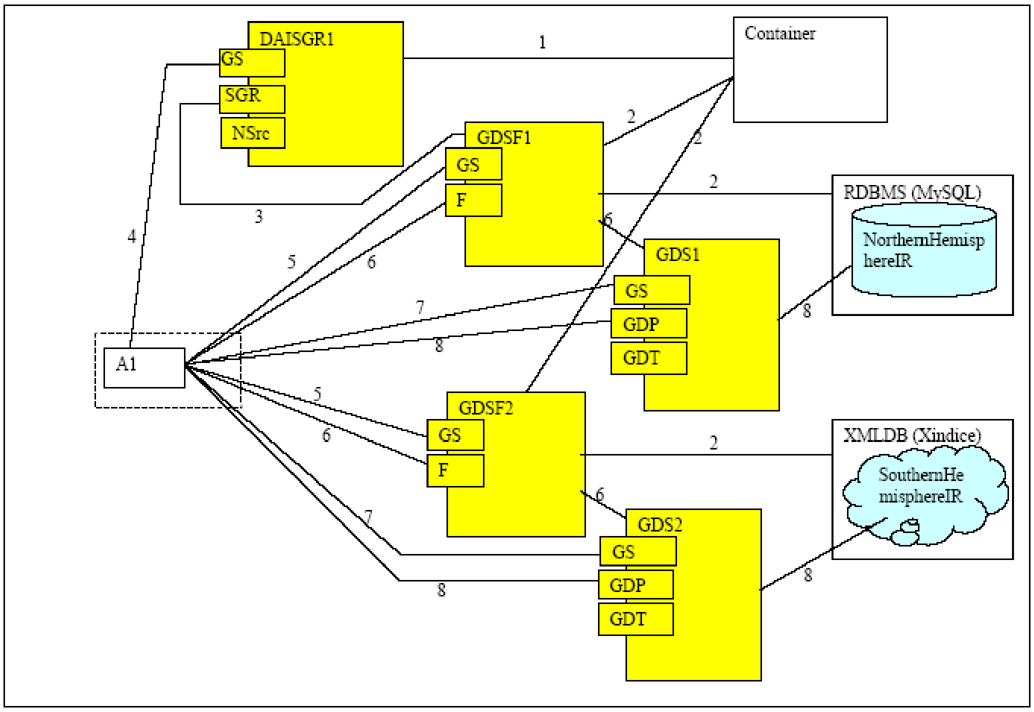
3.2.7. OGSA-DAI implementáció

Ebben a részben OGSA-DAI implementáció logikai felépítését ismertetem. A 3.6 ábrán látható az egyes szolgáltatások egymáshoz kapcsolódó viszonya. Az éleken látható számok megfeleltethetőek az egyes végrehajtási lépések logikai sorrendjének. A sorrend a következő:

1. A Grid szolgáltatás tároló (service hosting container) egy állandó szolgáltatás csoport regisztrálót (DAI Service Group Registry-t) hoz létre. (DAISGR1).

2. A Grid szolgáltatás tároló létrehoz állandó Grid adat szolgáltató központot (Grid Data Service Factory-t) (GDSF1, GDSF2), melyek létrehozáskor úgy konfigurálódnak, hogy reprezentáljanak egyedi adatforrásokat, melyeket hagyományos egyszerű adatbáziskezelőkön keresztül lehet elérni. A 3.6 ábrán két adatbázis van: egy relációs adatbázis kezelő illetve egy xml adatbázis kezelő. i

3. A GDSF1 regisztrálja magát a DAISGR-nél, így hirdetvén magát a potenciális kliensek felé.



3.6. ábra. Az együttműködő OGSA-DAI szolgáltatások

4. Az A1 nevű kliens kapcsolódik a DAISGR-hez keresvén olyan GDSF-t, mely képes létrehozni olyan GDS-t, ami kiszlgálja az ő kérését a megfelelő adatforráshoz kapcsolódva.

5. A kapott válasz alapján eldönti, hogy melyik GDSF-hez forduljon, majd az adott GDSF Factory::CreateService műveletét meghívva megkapja a kívánt GDS-t (3.2.7. alfejezet), ami már a megfelelő adatbázishoz kapcsolódik és rajta keresztül megkapja a szükséges információkat, illetve végrehajtja a kívánt műveleteket.

6. A kliens lekérdezéseket küldhet a GDS-nek, amik egyenlőre a GDS képességeire vonatkoznak.

7. XML dokumentumba csomagolva a GridDataPerform::Perform művelet segítségével lekérdezéseket és más műveleteket küld a kliens. A válasz szintén XML dokumentumba csomagolva érkezik.

DAISGR

Az OGSA-DAI Service Group Registry egy speciális regisztrációs szolgáltatási az összes OGSA-DAI szolgáltatás számára, melyek egy vagy több OGSA-

DAI portType-ot implementálnak. Ha egy szolgáltatás regisztrálja magát a DAISGR-nél, akkor az alábbi tulajdonságokkal lesz gazdagabb:

- A kliensek tudomást szerezhetnek a szolgáltatás létezéséről.
- A kliensek lekérdezhetik a szolgáltatás képességeit, illetve megtudhatják mely adatforrásokat tartja karban.
- Több kliens között is megosztódik.

A DAISGR egy állandó szolgáltatás, így az életciklusát a következő szabályok határozzák meg:

- A kilensek részéről bármilyen próbálkozás a szolgáltatás befejezésére, a GridService::Destroy művelet meghívása, figyelmen kívül lesz hagyva.
- Bármilyen próbálkozás a befejezési idő megváltoztatására a GridService::RequestTerminationBefore illetve a GridService::RequestTerminationAfter műveletek meghívása figyelmen kívül lesz hagyva.

Akkor jön létre, amikor az őt tartalmazó szolgáltatás tároló létrejön és akkor fejezi be a működését, amikor az őt tartalmazó szolgáltatás tároló megszűnik. A klienseknek megengedi, hogy lekérdezzék az állapotát illetve információkat kapjanak a hozzá regisztrált szolgáltatásokról. Ez utóbbit a GridService::FindServiceData művelet meghívásával tehetik meg. Szolgáltatások regisztrálása és a regisztráció megszüntetése több módon is történhet. Szoláltatás regisztrálása:

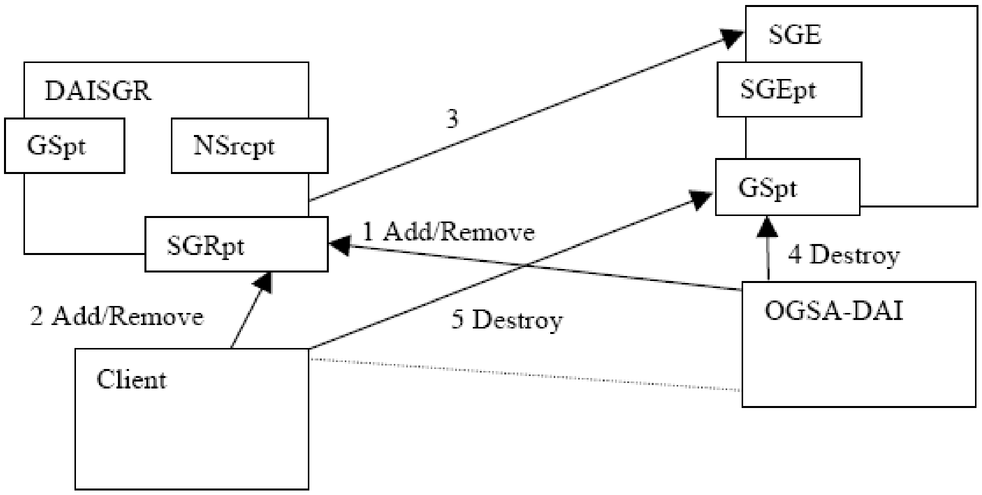
- A szolgáltatás regisztrálhatja magát a DAISGR-nél.
- Egy kliens vagy más OGSA-DAI szolgáltatás regisztrálja az adott szolgáltatást a DAISGR-nél.

Az egyes szolgáltatások regisztrálásához létrejön egy SGE (OGSI ServiceGroupEntry Service), ami karbantartja a szolgáltatás regisztrálást. A regisztráció megszüntetése az alábbi módokon történhet:

- A szolgáltatás maga kezdeményezi a regisztráció megszüntetését.
- Egy másik OGSA-DAI szolgáltatás illetve egy kliens kezdeményezi az adott szolgáltatás regisztrációjának megszüntetését.
- A DAISGR is kezdeményezheti az adott szolgáltatás regisztációjának megszüntetését, ha lejárt annak az érvényességi ideje.

- OGSA-DAI szolgáltatás vagy egy kliens meghívja a GridService::Destroy műveletét a SGE-nek, így implicit kezdeményezheti az adott szolgáltatást regisztrációjának megszüntetését.

A 3.7 ábrán látható a teljes regisztrációs részfolyamat. A szolgáltatás meg-



3.7. ábra. OGSA-DAI szolgáltatás (De)regisztrációja

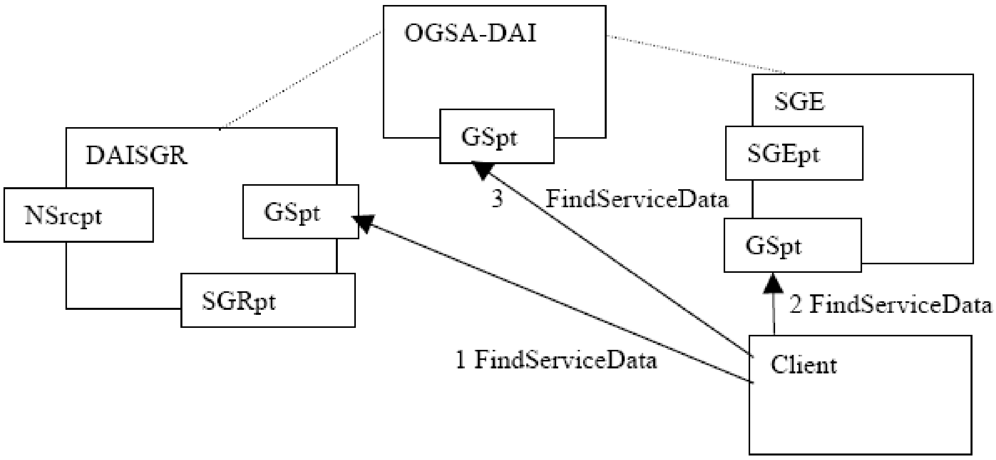
találása/felfedezése is több módon történhet. Első lépésként a DAISGR-hez kell fordulni és őt megkérdezni hol található az adott szolgáltatás, ami az igényeinknek megfelel. Ezek után fordulhatunk a szolgáltatáshoz magához vagy az ő regisztrációját karbantartó SGE-hez. A 3.8 ábrán látható a teljes részfolyamat.

A DAISGR-ben történő változások értesítési mechanizmusnak is megvannak a maga szabályai és szekvenciája.

- A kliens feliratkozik a DAISGR-nél a NotificationSource::Subscribe műveleten keresztül. A feliratkozás során specifikálja, hogy milyen események bekövetkezéséről kér értesítést (pl.: új szolgáltatás regisztrálása), illetve meghatározza azt a végpontot, ahova az értesítést kézbesíteni kell. Ennek a végpontnak a NotificationSink portType kiterjesztettjének kell lennie.

- A DAISGR létrehoz egy szolgáltatást, ami a NotificationSubscription portType kiterjesztése és karbantartja a jelentkezéseket.

- A DAISGR értesíti a klienst a szolgáltatás létrejöttéről és azonosítójáról.



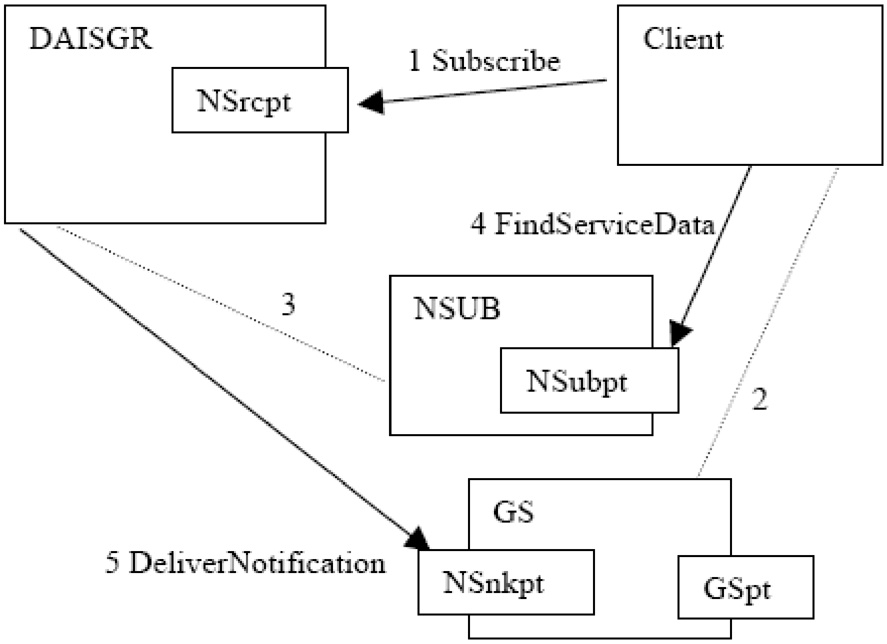
3.8. ábra. OGSA-DAI szolgáltatás megtalálása

- A kliens most már használhatja a létrejött szolgáltatást, lekérdezheti a feliratkozást és karbantarthatja rajta keresztül az érvényességi idejét.
- A DAISGR értesíti a kliens által meghatározott szolgáltatást a bekövetkezett változásokról a NotificationSink::DeliverNotification műveleten keresztül.

GDSF

A Grid Data Service Factory egy speciális szolgáltatás, ami felfedi az adat erőforrásokat és létre tud hozni új Grid Data Service-ket (GDS), melyeken keresztül a kliensek az adat erőforrásokhoz csatlakozhatnak. Ezen felül le is lehet kérdezni a GDSF-ket, hogy milyen erőforrások csatlakoznak hozzá, azaz tényleg megfelelnek e a kliens által támasztott igényeknek. A GDSF egy állandó szolgáltatás, melynek életciklusára és állapotára ugyanazok érvényesek, mint a DAISGR életciklusára. Mint ahogy már a fentiekben leírtuk, a GDSF-nek regisztrálniuk kell magukat a DAISGR-nél, hogy megtalálhatóak és lekérdezhetőek legyenek. A kliens kérhet egy GDS-t a GDSF Factory::CreateService műveletének meghívásával, amin keresztül elérheti az adat erőforrást. A GDSF GridService::FindServiceData műveletével a kliensek megkaphatják a GDSF által kezelt adatbázis leírását. Amikor kliens kér a DAISGR-től egy olyan GDSF azonosítót, mely létre tud hozni olyan GDS-t mely olyan erőforráshoz kapcsolódik, ami megfelel az igényeinek, akkor a következő válaszok lehetségesek:

- Nincs ilyen GDSF regisztrálva.



3.9. ábra. OGSA-DAI szolgáltatás változásnak értesítése

- Pontosan egy ilyen GDSF van regisztrálva. Ebben az esetben a kliens kérhet további infokat a GDSF-ről, csatlakozhat hozzá, a Factory::CreateService művelet segítségével létre hozathatja a GDS-t és közvelenül kommunikálhat a GDS-sel.
- Több lehetséges találat van. Ebben az esetben újabb, pontosabb kéréssel szűkíteni lehet a találati listát. Majd a megfelelő választás után az előző pont szerint csatlakozni lehet egy létrejött GDS-hez.

GDS

A Grid Data Service (GDS) a legfontosabb OGSA-DAI komponens. Ő adja az adatelérést, az adatintegrációs függvényeket és az adatkézbesítési függvényeket a GDSPortType portType-on keresztül, ami a GridDataPerform (GDP) portType és a GridDataTransport (GDT) portType kiterjesztése. A GDS által támogatott függvények részhalmazai annak, amik például a JDBC-ben megtalálhatóak. A GDP egy XML dokumentum-orientált interfész a klienstől jövő adatbáziskérések számára. A kérés a Grid Data Service Perform (GDS-

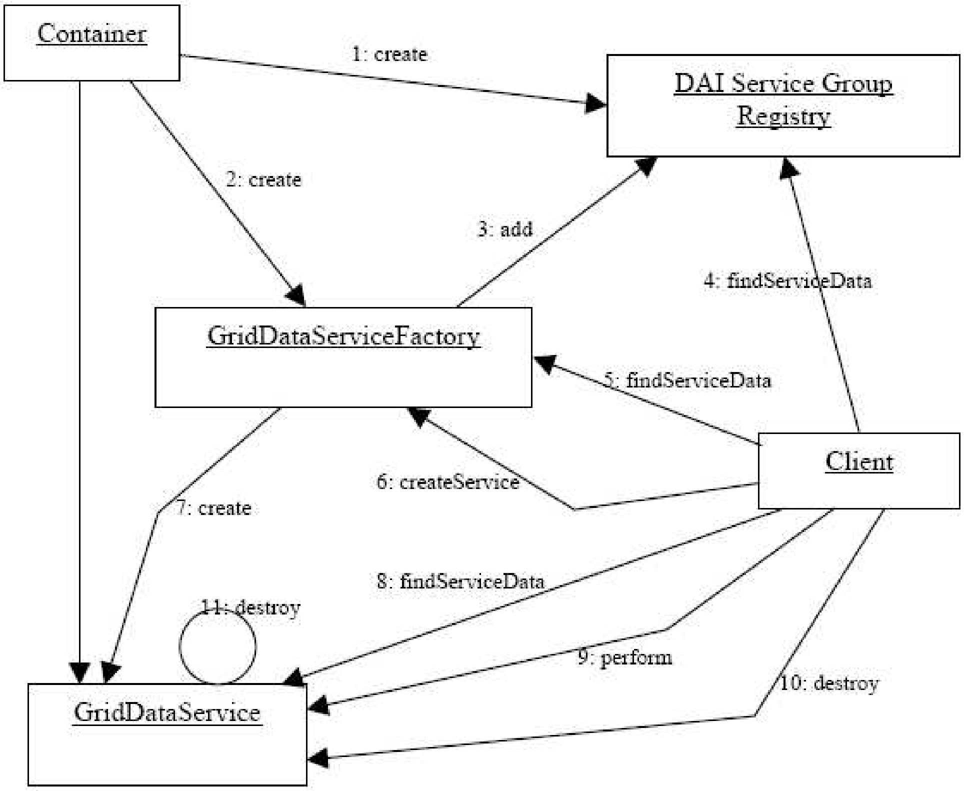
Perform) dokumentumban jön, a válasz pedig Grid Data Service Response (GDS-Response) dokumentumban megy vissza.

A GDS életciklusa az előzőektől eltérően csak ideiglenes.

- Amikor a GDSF létrehozza kap egy érvényességi/lejárati időt.
- Amikor az idő lejár, a GDS a továbbiakban nem lesz elérhető.
- A GridService::Destroy művelettel explicit lehet a GDS megszüntetését, működésének befejezését kérni.

A 3.10 ábrán látható, hogy az egyes OGSA-DAI szolgáltatások, hogyan kapcsolódnak egymáshoz a műveleteiken keresztül.

Egy időben egyszerre csak egy kérést tud a GDS végrehajtani. Minden egyes Perform dokumentumot névvel kell ellátnia a kliensnek. A kérés végrehajtása szinkron és aszinkron módon is lefolyhat.



3.10. ábra. OGSA-DAI szolgáltatások művelet szintű kapcsolatai

Példa egy Perform dokumentumra.

```
<gridDataServicePerform
  xmlns="http://ogsadai.org.uk/namespaces/2003/07/gds/types">
  <documentation>
    This example performs a simple select statement to
    retrieve one row from the test database. The results
    are delivered within the GDS-Response document.
  </documentation>
  <sqlQueryStatement name="statement">
    <sqlParameter position="1" from="idvalue" />
    <expression>
      select * from littleblackbook where id=?
    </expression>
    <webRowSetStream name="statementOutput"/>
  </sqlQueryStatement>
</gridDataServicePerform>
```

Az OGSA-DAI Perform dokumentuma az alábbi SQL részeket támogatja.

- sqlQueryStatement - lekérdezések
- sqlUpdateStatement - adatmódosítások
- sqlStoredProcedure - tárolt eljárás hivás
- relationalResourceManagement - adatbázis létrehozás és törlés

Mint ahogy azt a példán is láttuk paraméterezhetőek, kivéve a relációs erőforrás karbantartó (relationalResourceManagement). A paraméterek lehetnek konkrét értékek, vagy mint ahogy a példán is láthattuk egy másik szálból, lekérdezés, végrehajtás eredményéből nyerhető ki. Az egyes dokumentumok nagyon hasonlóak a példában láthatóhoz, csak a különböző esetekhez más és más elemnevek tartoznak.

Jelenleg a relációs adatbázisokból ismert SQL melett támogatja az XML szabvány alábbi műveleteit is:

- xmlCollectionManagement - új gyűjtemény létrehozása, törlése, létező gyűjtemények listája
- xmlResourceManagement - új bejegyzés(adatbázisban tárolandó XML dokumentum) létrehozása, törlése, létező bejegyzések listája
- xPathStatement - adott bejegyzések xPath-szal elérhető részének lekérdezése

- xUpdateStatement - adott bejegyzés XUpdate-tel történő adatmódosítása

Nem csak relációs és XML adatbázis műveleteteket támogat a GDS, hanem úgynevezett transzformációs műveleteket is. Ezek a következőek:

- gzipCompression - az eredmény a szerveren a UNIX-os világból ismert gzip-pel tömörített.
- xslTransform - az lekérdezés mellé csatolt xslt[1] transformációs file alkalmazása az eredményre. Így az eredmény adatból a szerveren elő lehet állítani például html formátumú kimenetet.
- zipArchive - normál zip tömörítési mód az eredményre, már a szerveren.

Az OGSA-DAI arra is lehetőséget ad, hogy megmondjuk hogyan és hova kézbesítse a választ. Az egyes lekérdezések bemenő paramétereit vehetjük a nagyvilágból http, https, file, ftp protokollokon keresztül vagy valamely másik nevesített lekérdezés eredményéből. A kimenetet pedig hasonlóan átirányíthatjuk a http, https, file, ftp protokollokra.

Példa egy átirányított bemenetre, a bemenet adatait http protokollon keresztül kapjuk meg.

```
<deliverFromURL name="d1">
  <fromURL>http://ogsadai.org.uk/resultsets/data.txt</fromURL>
  <toLocal name="datatoinsert" />
</deliverFromURL>
```

A lehetséges kézbesítési módok az alábbiak:

deliverToURL Az eredményt egy megadott URL-re továbbítja.

deliverFromURL A bemenő adatokat egy megadott URL-ről veszi.

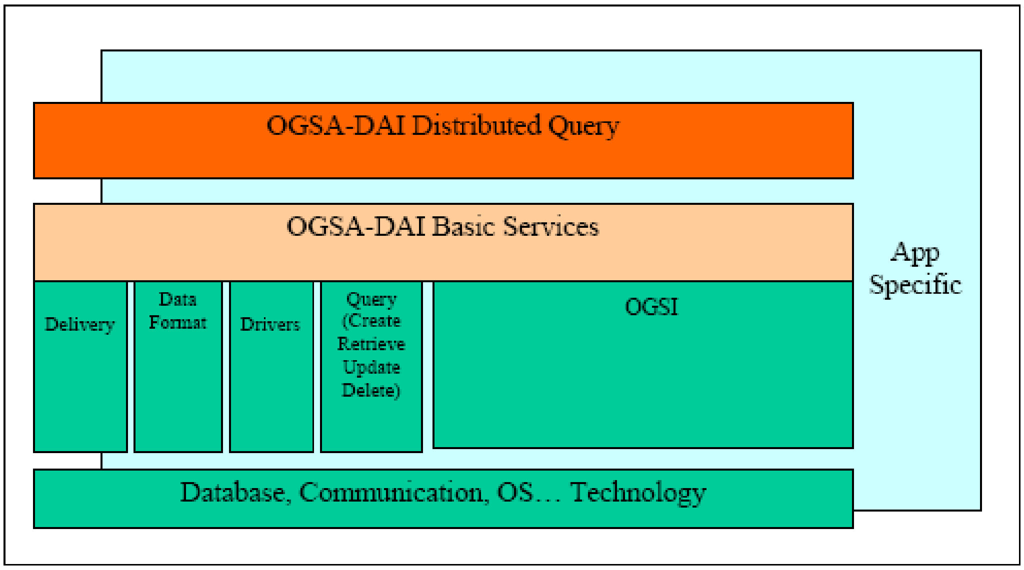
deliverFromGDT A bemenő adatokat egy olyan szolgáltatástól veszi, amelyik GridDataTransport portType-ot megvalósítja és képes adatokat szolgáltatni. Az adatszolgáltatás ebben az eseten történhet blokkosítva vagy egy nagy ömlesztett adat formájában.

deliverToGDT Az előzőhöz hasonló csak az irány a fordított, viszont ugyanazoknak a feltételeknek kell teljesülniük.

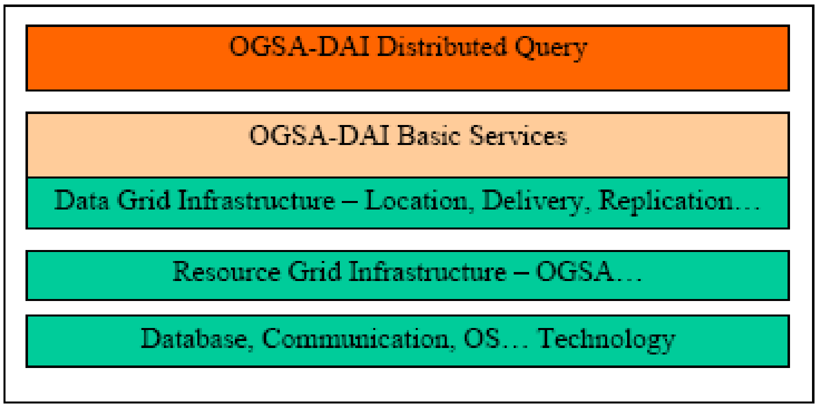
deliverFromGFTP A deliverFromURL-hez hasonló, de itt a használt protokoll a speciális gridftp-n alapul.

deliverToGFTP Az előzőhöz hasonló, csak az irány fordított.

[1] Az XSL-ről bővebben a http://www.w3.org/Style/XSL/ címen olvasható



3.11. ábra. OGSA-DAI napjainkban



3.12. ábra. OGSA-DAI tervezett architektúrája

3.2.8. Distributed Query Processing

Az OGSA-DAI Distributed Query Processing az ogsadai feletti újabb absztrakciós réteg. Az OGSA-DQP két szolgáltatással egészíti ki magát az OGSA-DAI-t.

- Grid Distributed Query Service (GDQS)
- Grid Query Evaluation Service (GQES)

A GDQS az elsődleges interfész a felhasználó felé. Egy koordinátor, ami megteremti a kapcsolatot az lejjebb lévő lekérdezés fordító és optimalizáló között, illetve a GQES példányok között. A GDQS megvalósítja a már említett GDS port type-ot. A GQES szintén implementálja a GDS-et, de ez nem kötődik közvetlenül egy adatforráshoz. A GQES-t futási időben használja a rendszer lekérdezés tervek (query plan) kiértékelésére. A GQES példányok számát és elhelyezkedését a Gridben a GDQS határozza meg, aki a lekérdezés optimalizátor kapott információk alapján (pl.: időzített lekérdezés részek, rész tervek) dönt. Az alábbiakban a GDQS-t részletezem, mivel a GQES létrehozását, életciklusát a GDQS határozza meg, a felhasználó számára transzparens módon.

GDQS

A GDQS egy példa adatok magas szintű integrációjára. Az OGSA-DAI keretrendszer szolgáltatásait használja, hogy elérje a különböző heterogén adatforrásokat. Az alábbi pontokon kapcsolódik az OGSA-DAI-hoz.

- Az elindulási, konfigurációs fázisában Serivce Data Element-eket (SDEs) kap a Grid Data Service Factory-tól (GDSF), ami ismerteti az adatforrások adatbázisschémáját.
- A lekérdezések végrehajtása alatt konfigurálja a GQES-eket, így képesek átadni a válaszokat (*Request Document*-eket) a GDS példányoknak, hogy megkapják a kívánt adatokat.

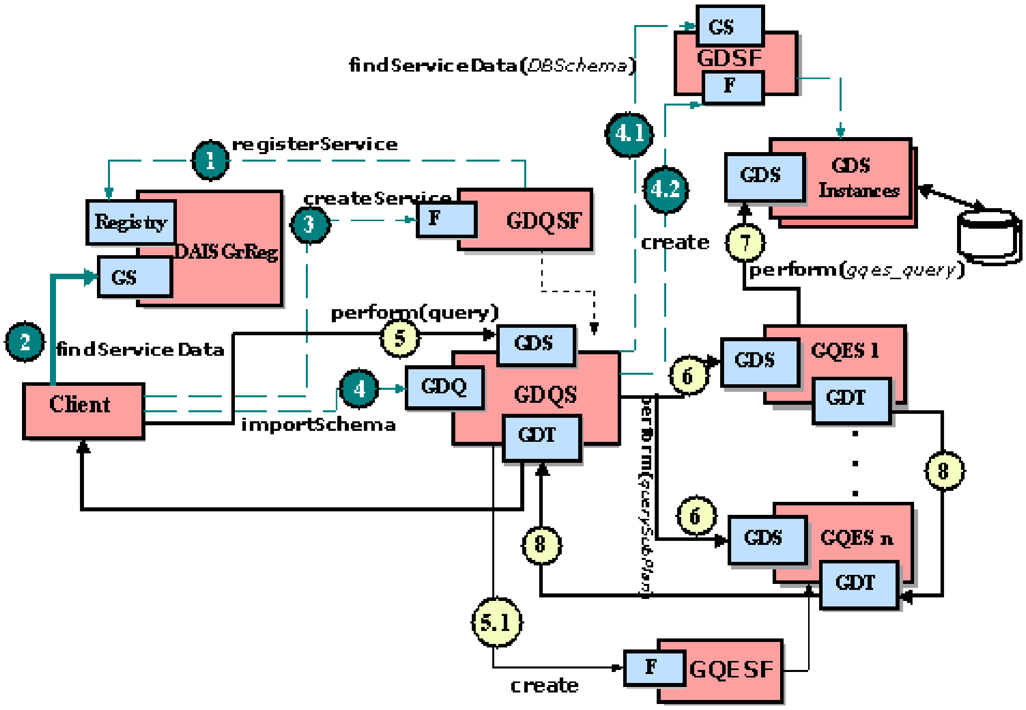
A GDQS az adatok elemzését egyesíti az adateléréssel és integrációval. Mivel a GDQS implementálja a GDS portType-ot, ezért hasonlóan hozzá a felhasználó felé egy dokumentum orientált lekérdező felületet nyújt. A lekérdezéseket a Grid Data Service Perform dokumentumban kell megfogalmazni.

A GDQS-sel való kapcsolatot két részre lehet felosztani.

- Elindítási fázis (Set up)
- Lekérdezési fázis (Query submission)

Elindítási fázis

A GDQS egy további portType-ot is implementál, hogy a felhasználó megadhassa, beállíthassa az adatforrásokat és az analizáló szolgáltatásokat a lekérdezésekhez. Ez a Grid Distrbuted Query portType, ami az importSchema műveletet definiálja. Az adatforrásokhoz azonosítani kell a GDSF kezelőt,



3.13. ábra. A DQP komponensek és az OGSA-DAI komponensek kapcsolata

ami kezeli az igényelt adatforrásokat, az analizáló szolgáltatásokhoz pedig azonosítani kell az URL-t, ami meghatározza azon WSDL dokumentumot, ami leírja az igényelt szolgáltatást. Mind a GSDF kezelőt, mind a WSDL URL-t egy XML dokumentumban kell listaszerűen felsorolni, ami megfelel a GDQDataSourceList.xsd schémának.

A DQP jelenlegi verziója még nem támogatja a schémakonfliktusok feloldását, ezért minden importált tábla nevének egyedinek kell lennie.

Lekérdezési fázis

A GDQS a lekérdezéseket a GDS portType-on keresztül fogadja. A lekérdezést be kell ágyazni egy hagyományos GDS Perform dokumentumba, de van néhány eltérés, ezek a következőek:

- GDQS kizárólag az OQL-t támogatja, mint lekérdező nyelvet.
- Habár a GDQS támogatja az összes továbbítási műveletet, nem támogatja az alábbi standard GDS műveleteket:
  - sql_query_statement

  - sql_storedprocedure
  - sql_update_statement
  - xml_collection_management
  - xml_resource_management
  - xpath_statement
  - xquery_statement
  - xupdate_statement
  - relational_resource_management

- Nem lehet az OQL lekérdezéseket paraméterezni, azaz a dokumentumban a teljes lekérdezést kell előállítani.
- Az OQL lekérdezést tartalmazó Perform dokumentumban szerepelnie kell az OQLQueryStatment műveletnek.

Életciklus

A GDQS egy ideiglenes szolgáltatás, aminek az élettartamára a következő tulajdonságok állnak fenn:

- Amikor a GDSF által létrejön, kap egy TTL (Time To Live) értéket, ennek értéke lehet nem definiált.
- Amikor a TTL érték elévül, a GDS nem lesz a továbbiakban elérhető.
- A GridService::Destroy művelettel a szolgáltatás befejezését lehet kérni, például ha nem definiált TTL-lel kértük a GDQS-t.

Mivel egy lekérdezés nagyon hosszú ideig tarthat a szétosztott volta miatt, ezért javasolt nem definiált TTL-lel kérni a GDQS-t, és a végén, amikor már senkinek sincs szüksége rá, explicit terminálással megszüntetni.

GDQS az OGSA-DAI-ban

A 3.13 ábrán láthatjuk egy lehetséges kapcsolat kiépítését a kezdetektől.

1. GDQSF regisztrálja magát a DAI Service Group Registry-nél.
2. A kliens megkérdezi a Registry-t arról, hogy hol található a GDQSF.
3. A Grid Distributed Service Factory felhasználásával létrejön egy GDQS példány.

4. A kliens meghívja a GDQS importSchema műveletét.
   (a) Az adatbázis schémát a GDSF-től megszerzi a GDQS.
   (b) A GDQS létrehoz egy GDS példányt a GDSF-en keresztül.
5. A kliens most már lekérdezéseket tud végrehajtani GDS perform műveleteken keresztül. A hívás lefordul és optimalizálódik szétosztott lekérdezési tervvé.
   (a) A GDQS felhasználja ezeket az adatokat a GQES példányok megfelelő csomópontokon való létrehozásához.
6. Ezek után a GDQS átadja az egyes GQES példányoknak a hozzájuk tartozó rész terveket.
7. Néhány GQES példány kapcsolódik más GDS példányokhoz adatok elérése céljából.
8. Az eredmény a GQES példányokon keresztül a klienshez visszajut.

A 2.2.1. fejezetben részletes információt találunk a OQL-ről. Az alábbiakban a DQP által támogatott OQL elemek olvashatóak:

- Szelekció
- Bizonyos típusú join (egyenlőségen alapuló, ami hash-joinnal és hash-loop joinnal implementált)
- Beágyazott lekérdezés
- Néhány aggregátum (count, sum, max, min, avg)
- Beágyazatlan lekérdezés
- Konjunkció (AND operátorral összekapcsolt elemek)
- Összehasonlító operátorok (=, !=, >, <, <=, >=, like)

A perform dokumentumban leírható és az OGSA-DAI által definiált transzformációk közül az alábbiak támogatottak a DQP-ben.

- gzipCompression
- xslTransform
- zipArchive

Mint ahogy már az OGSA-DAI-nál is meglehetett adni Delivery(kézbesítési) opciókat, azaz a választ hogyan és hova kézbesítse a rendszer, úgy itt is lehetőség van rá. A DQP-ben az alábbiak támogatottak:

- deliverToResponse
- deliverFromURL
- deliverToURL
- deliverFromGDT
- deliverToGDT
- deliverFromGFTP
- deliverToGFTP

A válasz megegyezik az OGSA-DAI-ban definiált válasz dokumentummal (response document), ami a GDS GridDataPort::perform művelet hívásának eredménye.

Példa

Példa a DQP Perform dokumentumra.

```
<gridDataServicePerform
  xmlns="http://ogsadai.org.uk/schemas/gds">
    <oqlQueryStatement name="statement">
        <expression>
            % print select struct(A:p.ORF,B:p.sequence)
            from p in protein_sequences;
        </expression>
        <webRowSetStream name="statementresult"/>
    </oqlQueryStatement>
</gridDataServicePerform>
```

Példa egy DQP Response dokumentumra.

```
<?xml version="1.0" encoding="UTF-8"?>
<gridDataServiceResponse
  xmlns="http://ogsadai.org.uk/namespaces/2003/07/gds/types">
    <result name="statement" status="COMPLETE"/>
    <result name="statementresult" status="COMPLETE">
        <![CDATA[<?xml version="1.0" encoding="UTF-8"?>
```

```
        <!-- DOCTYPE RowSet PUBLIC '-//Sun Microsystems,
          Inc.//DTD RowSet//EN'
          'http://java.sun.com/j2ee/dtds/RowSet.dtd' -->
        <RowSet>
        . . . (RowSet results here)
        </RowSet>]]>
    </result>
</gridDataServiceResponse>
```

4. fejezet

Példaprogram

Számos fejlesztésben használják az OGSA-DAI technológiát. Ezek közül csak néhányat említek meg[1]:

AstroGrid Egy angliai projekt, mely célja egy virtuális obszervatórium létrehozatala. OGSA-DAI technológiával akarják kialakítani az adattárházakat, melyen keresztül nagyméretű adatok kinyerése különböző helyeken lévő adatbázisközpontokból egyszerűbb.

eDiaMoND eDiaMoND egy Grid a Röntgen X-Ray Mammography mammográfiához. A mammogrammokat, melyek darabonként 32 Mb méretű objektumok, az IBM Content Managerben tárolják. A beteg adatai és képadatok egy relációs adatbázisban tárolódnak. A feladat egy absztrakciós réteg létrehozása, mely vegyesen tudja kezelni mind a két adatforrást. A projekt része egy wrapper készítése az IBM Content Managerhez.

A példák az élet számtalan területéről kerültek ki, de fontos észrevenni, hogy az egészségügyben és a kutatásokban mennyire fontos és elengedhetetlen szerepet töltenek be.

[1] További referenciák, példák a http://www.ogsadai.org.uk/projects/ cím alatt találhatók.

Az OGSA-DAI technológiát az alábbi programon keresztül vizsgáltam meg.

4.1. Neuron hálók szimulációja

A program (NeuroWeb) a bécsi egyetemmel (Universität Wien) együtt, ERASMUS program keretében, közös projektben készült. Az eredeti, kiindulási program Java nyelven íródott, ami külön megkönnyítette az OGSA-DAI integrációját.

4.1.1. NeuroWeb

A program neuron hálók szimulációjához nyújt környezetet. Megvalósíthatóak benne a legalapvetőbb neuron háló funkciók:

- Létrehozás
- Tanítás
- Kiiértékelés

A szimulátor magába foglalja az alábbi neuron háló példákat:

- BackPropagation
  - BackPropagation kötegelt módban
  - QuickPropagation
  - Resilient Propagation
- Self Organizing Map
  - Counter propagation
  - Hopfield net
  - ART 1
  - ART 2
  - Koholen TSP
- Reccurent Nets
  - Jordan net
  - Elman net

- Cascade Correlation
  - Cascade Correlation with Quickpropagation

A program eredeti verziója az alábbi tulajdonságokkal rendelkezik:

- A szimulációs környezet Java nyelven íródott.
- A rendszer webes alapokon nyugszik a bővülő internetes társadalom számára is elérhetővé téve.
- Az előző két tulajdonság miatt platformfüggetlen.
- A rendszer a statikus és dinamikus részek számára az adatokat adatbázisokban tárolja és onnan is nyeri ki.
- Kiterjeszthető. Mint már említettük, a program egy futtató környezet, melyet jól definiált interfészen keresztül ki lehet terjeszteni újabb neuronháló példákkal.
- Mivel az egyes tanítási mechanizmusok erőforrásigényesek lehetnek, ezért a neuron hálok szimulációja távoli gépeken is futtatható, kihasználva az internet nyújtotta lehetőségeket.
- Az egyes neuronhálók futtatása, akár a lokális gépen, akár a távoli gépen, párhuzamosan történhet.

A program készítői már tervezéskor is olyan célokat tűztek ki maguknak, mint amiről a Grid szól. Innen egyenesen jött az ötlet, hogy a napjainkra kialakult Grid implementációk figyelembe vételével és azok felhasználásával újraírják, átalakítsák a programot. Az alábbi különálló részek átírása volt az új célkitűzés.

Futtatás A Grid felhasználása a távoli gépeken, Grid csomópontokon való futtatásban. Az eredeti verzió a Java RMI eszközkészletét használta fel a távoli futtatásra. A lehetséges távoli gépeken futnia kellett szerver programoknak és "CORBA jellegű" távoli metódushívás valósította meg a távoli futtatást.

Kiterjeszthetőség A program dinamikusan kiterjeszthető új neuronháló szimulációkkal többféleképpen.

- Lokális fejlesztés
- Távoli fejlesztés

Mindegyik esetben a futtató környezet az erre szolgáló adatbázisból veszi az információt a lehetséges szimulációkról. Az utóbbi esetben az internet nyíltságát szemelőtt tartva automatizmussal is megtámogatták, oly módon, hogy tetszőleges ftp szerverekre felrakva megadott leíró állományokban, megadott formátumban lehet az új szimulációt hirdetni. A rendes ftp kiváltása a gsiftp-vel újabb feladat.

Adathozzáférés Mint már említettük a szimulációs környezet, és így az egyes szimulációk is, az adatokat vagy közvetlen felhasználói bemenetről nyerték, vagy adatbázis lekérdezésekből. Ezen felül a szimuláció eredményeit, illetve a tanítási eredményeket adatbázisba menthették. Ennek a résznek az átírása egy újabb feladat, mely magától értetődően a diplomamunka alapjául szolgáló OGSA-DAI technológiával történt.

Jogosultságkezelés Egységes Griden alapuló jogosultság kezelés bevezetése a futtató környezetbe, ami a különböző részek (GLOBUS, OGSA-DAI) eltérő jogosultság kezelése miatt nem valósítható meg értelmesen. Jelenleg az OGSA-DAI egyáltalán nem használ sem saját, sem a GLOBUS által nyújtott lehetőségeket.

A program átírása során figyelembe kellett venni azt a követelményt is, hogy a jelenlegi adathozzásférési mechanizmus – JDBC – megmaradjon, illetve választani lehessen az egyes szimulációknál az adathozzáférési módok között. Ennek a feltételnek a lekezelése egy egyszerű tabbed pane volt, ami mindig az aktuális állapotától függően kapcsolta az új szimulációt össze az adatbázissal.

Lényegesebb változás az adatbázishoz való kapcsolódás illetve a lekérdezések felépítése során történt. A kétféle kapcsolódás eltérő kéréseinek és eredményeinek értelmezésének megkönnyítése végett egy absztrakciós réteget vezettem be a lekérdezések összeállítására illetve az erdemények értelmezésére. A Result osztály a ResultSet interface-t implementálja, mint ahogy a normál datbázis lekérdezések során az eredmény is ilyen típusú osztályból nyerhető ki. A Convert osztály felhasználja a Result osztályt, és a konverzió során a kapott XML Request dokumentumból a szabványos Java dom interfészen keresztül előállítja a ResultSet-nek megfelelő eredményt. Ezzel a megoldással nem kell az eredmények feldolgozását módosítani, csak a lekérdező és adat/adatbázis módosító utasításokat kell egy picit más formába csomagolni.[2] A kapcsolódás a normál kapcsolat kialakításhoz hasonlóan történt, de az OGSA-DAI által specifikált Java interfészen keresztül. Az alábbiakban néhány kapcsolódásiponthoz tartozó kódrészletet mutatok be:[3]

[2] A részletes kódrészletek megtalálhatóak a Result.java, Convert.java fájlokban.

[3] A részletes interfész specifikáció a http://www.ogsadai.org.uk/docs/ címen található.

- Kapocslódás a DAISGR-hez:

```
String urlString = <YOUR_DAISGR_URL>;
try {
  URL url       = new URL(urlString);
  mDAISGRLocator  = new
  DAIServiceGroupRegistrationServiceGridLocator();
  mDAISGR = gdsrLocator
              .getDAIServiceGroupRegistrationPort(url);
} catch (Exception e) {
  // Probléma kezelés.
}
```

- A DAISGR lekérdezése a regisztrált szolgáltatásokról:

```
try {
  // A kérés összeállítása.
  ExtensibilityType query =
  DAIServiceGroupQueryHelper.getPortTypeQuery(portTypes);
  // A lekérdezés elindítása.
  ExtensibilityType result = mDAISGR.findServiceData(query);
  } catch (Exception e) {
    // Probléma kezelés.
  }
```

- Kapcsolódás a GDSF-hez:

```
  String urlString = <YOUR_GDSF_URL>;
  try {
    URL url       = new URL(urlString);
    mGDSFLocator = new
                    GridDataServiceFactoryServiceLocator();
    mGDSF         = mGDSFLocator.
                        getGridDataServiceFactoryPort(url);
  } catch (Exception e) {
    // Probléma kezelés.
  }
```

- Kapcsolódás egy GDS-hez:

```
try {
  GDSServiceGridLocator gds = new GDSServiceGridLocator();
  GDSPortType gdsPt = gds.getGDSPort(gdsLocator.value);
```

```
} catch (Exception e) {
}
```

- Egy Perform dokumetnum végrehajtása a GDS-en keresztül:

```
// A perform dokumentumot tartalmazó xml file neve.
  String fileName = <YOUR_XML_FILE_NAME>;
try {
// Kezelhető XML dokumentum elkészítése
Document doc = XMLUtilities.xmlFileToDOM(fileName, false);
// Dokumentum beillesztése a kérésbe.
ExtensibilityType gds_pd =
AnyHelper.getExtensibility(doc.getDocumentElement());
// A GridDataService::Perform művelet meghívása.
ExtensibilityType gds_rd = mGDSpt.perform(gds_pd);
} catch (GridDataServiceUserExceptionType e) {
// OGSA-DAI Felhasználó kivételek
} catch (GridDataServiceSystemExceptionType e) {
// OGSA-DAI Rendszer kivételek
} catch (Exception e) {
// Más problémák.
}
```

5. fejezet

Eredmények

Az előző fejezetekben az OGSA-DAI elméleti bemutatását és gyakorlati megvalósítását láthattuk. Ez a fejezet az elmélet és a gyakorlati megvalósítás összehasonlításait, illetve különböző szempontok szerinti vizsgálatát mutatja be.

5.1. Átlátszóság

Napjaink osztott rendszereit[DIST] átlátszóság szempontjából az alábbi csoportokba sorolhatjuk:

Hozzáférhetőség A felhasználó elöl elrejti a tényleges hozzáférési módot. Ezt az átlátszóságot teljesíti az OGSA-DAI, mivel egységes felületen keresztül jól definiált módon tud a felhasználó kapcsolódni különböző adatbázisokhoz. Számára ugyanolyannak tűnik az ORACLE-höz, DB2-höz, PostgreSQL-hez, MySQL-hez valamint az Xindice adatbázis kezelőkhöz a kapcsolódás. Persze itt csak szigorúan az elérésről van szó. Az adatok kezelése, karbantartása függ attól, hogy az egyes adatbázis implementációk mennyire támogatják a szabványt, viszont a végrehajtó (perform) műveletek ugyanúgy néznek ki mindegyik esetben.

Elhelyezkedés A felhasználó elöl elrejti az erőforrás fizikai helyét. Ezt az átlátszóságot is teljesíti, hiszen a felhasználónak elegendő a DAISGR címéről tudnia. Rajta keresztül kapcsolódik az adatbázishoz az egész folyamat alatt, nem tudván az adatbázis tényleges fizikai helyéről.

Áthelyezhetőség Elrejti az erőforrás más helyre való áthelyezhetőségét. Ezt a feltételt is teljesíti, hiszen egészen addig amíg a szolgáltatás tárolóhoz (host container) új elérési címet nem rendelnek, illetve az egyes

GDSF-ek máshogy nevezik az általuk reprezentált adaterőforrásokat, a felhasználó semmit sem tud arról, hogy hol van a tényleges fizikai erőforrás.

Mozgathatóság Elrejti a felhasználó elől a fizikai erőforrás átmozgatását, még akkor is, amikor az használatban van. Az előzőnél ez egy jóval erősebb feltétel. Nem támogatja az OGSA-DAI.

Többszörözhetőség Elrejti az erőforrás többszörözöttségét. Jelenleg az OGSA-DAI nem támogatja a másolatok (replikák) kezelését, így ezt az átlátszóságot sem.

Egyidejűség Elrejti a felhasználó elöl az erőforrás versenyhelyzetű megosztottságát. Ezen problémával nem maga az OGSA-DAI foglalkozik, hanem a mögöttes adatbázis kezelő, azaz ebben az értelemben támogatja, de ha azt szeretnénk elérni, hogy például kizárólagosan használjunk (lockoljunk) egy adatbázis táblát, akkor ezt pillanatnyilag nem tudjuk az OGSA-DAI-n keresztül megtenni, tehát nem támogatja.

Meghibásodás Elrejti a felhasználó elöl az erőforrás meghibásodását, és újbóli üzembehelyezését. A válasz nem, mivel jelenleg az OGSA-DAI nem támogatja a másolatokat, ezért hiba esetén nem tud máshova átváltani, valamint az OGSA-DAI mélyebb rétegeiben a fizikai kapcsolódás JDBC-n keresztül történik, ezért csak azt tudja kezelni, amit a JDBC megenged.

5.2. Biztonságosság

Más oldalról megközelítve a vizsgálatot fontos kérdés, hogy mennyire biztonságos, mennyire működik együtt már meglévő biztonságtechnológia módszerekkel. Az OGSA-DAI napjainkban kezd felzárkózni a biztonságot illetően. A legutolsó verziót leszámítva csak látszólag támogatta a GLOBUS-ban támogatott biztonsági módszert, ami nyilvános kulcs titkosításon, X.509-es bizonyítványon és SSL-en alapszik. Hasonló biztonsági módszert támogat a MySQL is, amely első volt a támogatott adatbáziskezelő között, ennek ellenére az OGSA-DAI legutolsó verziójáig nem lehetett használni ezt a biztonsági mechanizmust. De ez a módszer még mindig csak a szolgáltatás tároló és az adatbázis szerver közötti kapcsolatot hitelesíti illetve teszi biztonságossá. A felhasználók azonosítása, hitelesítése, ami a GLOBUS-ban már létezik és használatos, az OGSA-DAI-ban még nem funkcionál.

5.3. Másolatok és konzisztenicia

Mint ahogy már az átlátszóságnál is említettem az OGSA-DAI egyenlőre nem kezel másolatokat, így magunknak kell kezelni a másolatokat, valamint a konzisztenciájukról is magunknak kell gondoskodni.

5.4. Szétosztottság

Az OGSA-DAI elrejti a felhasználó elöl az erőforrások fizikai elhelyezkedését, de minden egyes erőforrást egyedinek tekint. Ebböl következik, hogy a felhasználó is csak egyedileg tudja kezelni őket. Erre a problémára a DQP részleges megoldást ad. Ha nem az OGSA-DAI műveleteivel közelitjük meg a rendszert, hanem a DQP-n keresztül, akkor bizonyos adaterőforrásokat egységesen tudunk kezelni, de egyenlőre ez csak a lekérdezéseket tartalmazza. Létrehozást, módosítást és törlést nem tudunk végrehajtani szétosztott módon.

5.5. Sebesség

Az OGSA-DAI-t fontos megvizsgálni sebesség szempontjából is. Ilyen szempontból az eredmények még nem kielégítőek. Alaposabban megnézve a tényleges implementációt, már az installáció és konfiguráció során is láthatjuk, hogy a relációs adatbázis kezelők vezérlése teljesen JDBC-n, míg az XML adatbázis kezelő vezérlése XML:DB-n alapul. A kapcsolódás folyamata már eleve lassabb, hiszen nem közvetlenül az adatbázishoz kapcsolódunk, hanem egy DAISGR-hez, vagy ezt megkerülve esetleg egy GDSF-hez, és majd csak egy GDS létrehozása után tudunk az adatbázishoz kapcsolódni, ami egy közvetlen JDBC-c kapcsolódáshoz képest időben jelentős. Azoknál a programoknál amelyek futásuk alatt csak egyszer akarnak kapcsolódni, ez az idő nem igazán számottevő. Másik lassító tényező az XML dokumentum alapú interfészben keresendő. Már egy egyszerű GDS létrehozását is XML dokumentumon keresztül kell kérni. Mint ahogy az előzőekben olvashattuk a létrehozó, módósító és lekérdező műveleteket is XML dokumentumba ágyazottan kell végrehajtani. A válasz szintén XML dokumentumba ágyazottan érkezik meg. Az OGSA-DAI által támogatott relációs adatbázis kezelők többségének legújabb verziói már támogatják az XML-es adatformátumok tárolását, keresését, de nem csak az új verziókat támogatja a rendszer, ráadásul szigorúan SQL szabványnak megfelelő módon kezeli. Ebből következik, hogy a rendszer minden egyes alkalommal egy konverziót hajt végre, az adatbáziskezelőtől kapott eredményt XML dokumentumba csomagolja CDATA részbe.

Így amikor a kliens megkapja a válaszadatot, vagy hatékony string kereséssel, vagy kétszeres XML dokumentum készítéssel juthat hozzá az eredményhez. Ezek az állandó konverziók már jelentős lassulást idézhetnek elő, ha sok lekérdezés fordul elő a rendszerben. Nagyméretű válaszadatok esetén, jelentős letöltési sebességnövekedést érhetünk el, ha használjuk a gzipCompression, zipArchive opciókat.

5.6. Összegzés

A célkitűzéseket és a gyakorlati megvalósítást figyelmebe véve egy nagyon hasznos eszközt mutat be az OGSA-DAI. A teljes célkitűzést még nem valósítja meg, de már ez a verziója is a Grid-es szabványoknak megfelelően terjeszti ki a Grid-et implementáló GLOBUS-t.

Irodalomjegyzék

[IBMRED] Bart Jacob, Luis Ferreira: Enabling Applications for Grid Computing with Globus, 2003, ISBN:0738453331

[GGF7-1] Vijayshanker Raman, Inderpal Narang, Chris Crone, Laura Haas, Susan Malaika, Tina Mukai, Dan Wolfson and Chaitan Baru, Services for Data Access and Processing on Grids

[GGF7-2] Malcolm P. Atkinson, Vijay Dialani, Leanne Guy, Inderpal Narang, Norman W. Paton, Dave Pearson, Tony Storey and Paul Watson, Grid Database Access and Integration: Requirements and Functionalities

[GGF7-3] Neil P Chue Hong, Amy Krause, Simon Laws, Susan Malaika, Gavin McCance, James Magowan, Norman W. Paton and Greg Riccardi, Grid Database Service Specification

[DQP] M.Nedim Alpdemir: Grid Distributed Query Service, Document Identifier: OGSA-DAI-USER-UG-GDQS-v1.0, http://www.ogsadai.org.uk/docs/dqp/html/OGSA-DAI-USER-UG-GDQS.htm

[DAISGR] Mike Jackson: DAI Service Group Registry, Document Identifier: OGSA-DAI-USER-UG-DAISGR-v4.0,http://www.ogsadai.org.uk/docs/R3/html/OGSA-DAI-USER-UG-DAISGR.htm

[GDSF] Mike Jackson, Mario Antonioletti, Amy Krause: Grid Data Service Factory, Document Identifier: OGSA-DAI-USER-UG-GDSF-v4.2, http://www.ogsadai.org.uk/docs/R3/html/OGSA-DAI-USER-UG-GDSF.htm

[GDS] Amy Krause, Tom Sugden, Andrew Borley: Grid Data Service, Document Identifier: OGSA-DAI-USER-UG-GDS-v4.1, http://www.ogsadai.org.uk/docs/R3/html/OGSA-DAI-USER-UG-GDS.htm

[OVERVIEW] Mario Antonioletti, Mike Jackson: OGSA-DAI Product Overview, Document Identifier: OGSA-DAI-USER-UG-PRODUCT-OVERVIEW-v4.0, http://www.ogsadai.org.uk/docs/R3/html/OGSA-DAI-USER-UG-PRODUCT-OVERVIEW.htm

[DIST] Andrew S. Tanenbaum - Maarten van Steen: Distributed Systems: Principles and Paradigms, Pearson Education Limited, Panem könyvkiadó, 2002, ISBN: 963-545-387-6

[XML-Schema] Eric van der Vlist: XML Schema, O'Reilly & Associates, 2002, ISBN: 0-596-00252-1

[XML] Neil Bradley: Az XML kézikönyv, Pearson Education Limited, 2000, ISBN: 963-9131-21-0