

12 OSZTOTT RENDSZEREK ARCHITEKTÚRÁI

TÉMAKÖRÖK

Ezen fejezet célja az osztott rendszerek szoftverarchitektúra-modelljeinek tárgyalása. Mire a fejezet végére érünk:

- meg fogjuk érteni az osztott rendszerek architektúráinak fő előnyeit és hátrányait;
- meg fogjuk érteni az osztott rendszerek architektúrájának két alapvető modelljét, nevezetesen a kliens–szerver rendszereket és az osztott objektumarchitektúrákat;
- meg fogunk ismerkedni az ORB fogalmával és a CORBA-szabványok alapfogalmaival;
- meg fogjuk ismerni a peer-to-peer és a szolgáltatásorientált architektúrákat mint szervezeti osztott rendszerek implementálásának lehetséges módjait.

TARTALOM

- 12.1. Többprocesszoros architektúrák
- 12.2. Kliens–szerver architektúrák
- 12.3. Osztott objektumarchitektúrák
- 12.4. Szervezeti osztott rendszerek

Jelenleg jóformán minden nagy számítógép-alapú rendszer osztott. Az osztott rendszerek olyan rendszerek, amelyekben az információfeldolgozás nem egyetlen gépre korlátozódik, hanem el van osztva több számítógép között. Nyilvánvalóan az osztott rendszerek tervezésének meglehetősen sok köze van az egyéb szoftverek tervezéséhez, de felmerülnek olyan jellegzetes kérdések is, amelyekkel az ilyen rendszerek tervezésekor számolni kell. Néhány ilyen kérdést már felvettem a 11. fejezetben, a kliens–szerver architektúrák bemutatásánál és itt részletesebben szólok róluk.

Coulouris és mások (Coulouris és mások, 2001) az osztott rendszerek fő jellemzőit tárgyalják. Szerintük a rendszer fejlesztésének osztott megközelítése a következő előnyökkel rendelkezik:

1. *Erőforrás-megosztás.* Egy osztott rendszer lehetővé teszi a hálózaton lévő számítógépekhez kapcsolt hardver- és szoftvererőforrások – lemezek, nyomtatók, állományok, fordítóprogramok – megosztását.
2. *Nyíltság.* Az osztott rendszerek általában nyílt rendszerek, ami azt jelenti, hogy olyan szabványos protokollok köré szerveződnek, amelyek lehetővé teszik különböző szállítók eszközeinek és szoftvereinek a kombinálását.
3. *Konkurencia.* Az osztott rendszerekben számos folyamat működhet egyidejűleg a hálózat különböző számítógépein. Ezek a folyamatok működésük során kommunikálhatnak egymással (de ez nem kötelező).
4. *Skálázhatóság.* Az osztott rendszerek skálázhatók – legalábbis elvben –, abban az értelemben, hogy a rendszerrel szemben támasztott új igények kielégítésére a rendszer képességei új erőforrások hozzáadásával növelhetők. A gyakorlatban azonban a skálázhatóságnak korlátot szabhat a rendszer számítógépeit összekapcsoló hálózat. Sok új számítógép csatlakoztatása esetén a hálózati kapacitás elégtelenné válhat.
5. *Hibátűrés.* A számítógépek rendelkezésre állása és az információ többszörözésének potenciálja azt jelenti, hogy az osztott rendszerek bizonyos hardver- és szoftverhibákat képesek eltűnni (20. fejezet). Az osztott rendszerek legtöbbje hibák esetén csupán csökkentett szolgáltatást biztosít, a szolgáltatások teljes megszűnése csak hálózati hiba esetén következik be.

A nagy rendszerek számára ezek az előnyök azt jelentik, hogy az osztott rendszerek nagyrészt az 1980-as és 1990-es években fejlesztett központosított őrendszerek helyébe léptek. Azonban az egyetlen processzoron vagy processzorfürtön futó rendszerekkel szemben az osztott rendszereknek számos hátránya van:

1. *Bonyolultság.* Az osztott rendszerek sokkal bonyolultabbak, mint a központosítottak. Ez megnehezíti felmerülő tulajdonságainak megértését és a rendszerek tesztelését. Például ahelyett, hogy egy rendszer teljesítménye egyetlen processzor végrehajtási sebességétől függne, a hálózat sávszélességétől, valamint a hálózatban lévő különböző processzorok sebességétől függ. Az erőforrások a rendszer egyik részéből a másikba történő átmozgatása gyökeresen befolyásolhatja a rendszer teljesítményét.
2. *Biztonság.* A rendszer több, különböző számítógépről érhető el, és a hálózatot is lehallgathatják. Ezáltal nehezebb biztosítani, hogy a rendszer fenntartja az adatok integritását, és hogy szolgáltatásai nincsenek kitéve szolgáltatás-visszautasítást eredményező támadásoknak.
3. *Kezelhetőség.* A rendszer különböző számítógépei különböző típusúak lehetnek, és rajtuk különböző verziójú operációs rendszerek futhatnak. Az egyik gépben bekövetkező hibák beláthatatlan következményekkel terjedhetnek át a többi gépre. Ez azt jelenti, hogy a működő rendszer kezelése és karbantartása több erőfeszítést igényel.
4. *Megjósolhatatlanság.* Mint ahogyan azt a www minden használója tudja, az osztott rendszerek válaszai megjósolhatatlanok. A válasz függ a rendszer teljes terhelésétől, szervezési módjától és a hálózati terheléstől. Mivel ezek igen

rövid idő alatt megváltozhatnak, egy felhasználói kérésre adott felelet válasz-ideje egyik kérésről a másikra drasztikusan megváltozhat.

Az osztott rendszerek tervezői számára kihívás olyan hardver és szoftver tervezése, amely rendelkezik az osztott rendszerektől elvárt tulajdonságokkal és ugyanakkor minimalizálja az ezekkel a rendszerekkel kapcsolatos problémákat. Ehhez meg kell érteni a különböző osztott rendszerek architektúrájának előnyeit és hátrányait. Itt az osztott rendszerek architektúrája két általános típusával foglalkozunk:

1. *Kliens–szerver architektúrák.* Ebben a megközelítésben a rendszer nem más, mint kliensek számára biztosított szolgáltatások halmaza. Ezek a rendszerek különbözőképpen kezelik a szervereket és a klienseket.
2. *Osztott objektumarchitektúrák.* Ebben az esetben nincs különbség a szerverek és kliensek között, és a rendszert olyan egymással kölcsönhatásban lévő objektumok halmazaként képzelhetjük el, amelyek helye lényegtelen. Nem különböztetjük meg a szolgáltatókat és a szolgáltatások felhasználóit.

Az iparban mind a kliens–szerver, mind pedig az osztott objektumarchitektúrákat széles körben használják, de az alkalmazások osztottsága általában egyetlen szervezeten belül értendő. Az így támogatott osztottság így szervezeten belüli. Tárgyak két további osztott architektúrafajtát is, amelyek a szervezeten kívüli megosztás számára megfelelőek: a peer-to-peer (p2p) és a szolgáltatásorientált architektúrákat. A peer-to-peer rendszereket leginkább személyes rendszerekben alkalmazták eddig, de most kezdik már az üzleti alkalmazásokban is használni. A könyv írásának idején a szolgáltatásorientált rendszerek éppen bevezetésre kerültek, de ez a megközelítés valószínűleg csak 2005-re válik majd igazán fontosá.

Az osztott rendszerek különböző komponensei különböző programozási nyelveken implementálhatók és különféle processzorokon futtathatók. Az adatmodellek, az információ ábrázolásmódja és a kommunikációs protokollok mind-mind eltérhetnek. Egy osztott rendszernek ezért olyan szoftverre van szüksége, amely a különféle részeket kezelni tudja, és biztosítja a kommunikáció és az adatcsere lehetőségét. Az ilyen szoftvereket *köztes termékeknek* hívjuk – a rendszer osztott komponensei között középen helyezkednek el.

Bernstein (Bernstein, 1996) összefoglalja az osztott számítások támogatása céljából elérhető különböző típusú köztes termékeket. A köztes termékek olyan általános célú szoftverek, amelyeket általában kulcsrakész formában vásárolnak meg ahelyett, hogy az alkalmazás fejlesztői írják meg azokat. Köztes termékek például az adatbázisokkal való kommunikációt kezelő szoftverek, a tranzakciókezelők, az adatkonverterek, a kommunikációvezérlők stb. A köztes termékek egyik nagyon fontos osztályáról, az osztott rendszerek keretrendszereiről, a fejezet későbbi részében szólok.

Az osztott rendszereket általában objektumorientált módon fejlesztik. Ezek a rendszerek olyan lazán integrált, független részekből állnak, amelyek mindegyi-

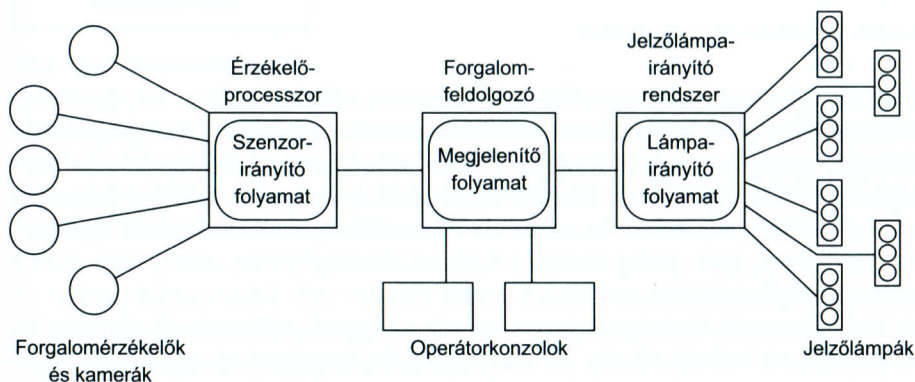
ke közvetlenül a felhasználókkal, illetve a rendszer egyéb részeivel léphet kapcsolatba. A rendszer részeinek esetleg független eseményekre kell válaszolniuk. A szoftverobjektumok kifejezik ezeket a jellegzetességeket, ily módon az osztott rendszerek komponenseinek természetes absztrakciói.

12.1. TÖBBPROCESSZOROS ARCHITEKTÚRÁK

Az osztott rendszer legegyszerűbb modellje a többprocesszoros rendszer, ahol a különböző folyamatok különálló processzorokon futhatnak (de nem szükségszerűen futnak is). Ez a modell nagyméretű valós idejű rendszerek esetén gyakori. Mint ahogy az a 15. fejezetben olvasható, ezek a rendszerek információt gyűjtenek, amely alapján döntéseket hoznak, majd jelzéseket küldenek a rendszer környezetét megváltoztató működtetőknek.

A folyamatok az információ összegyűjtésével, a döntéshozatallal és a működtető vezérlésével foglalkoznak, és ezek mindegyike képes egyetlen processzoron, egy ütemező által vezérelve futni. Több processzor használata növeli a rendszer teljesítményét és rugalmasságát. Az egyes folyamatok elosztása processzorokra történhet előre meghatározott módon (kritikus rendszerekben ez gyakori) vagy egy ütemező segítségével, amely eldönti, hogy az egyes processzorokhoz mely folyamatokat kell hozzárendelni.

Egy ilyen rendszer példája látható a 12.1. ábrán. Ez egy közlekedésirányító rendszer egyszerűsített modellje. Osztott érzékelők segítségével információt gyűjt a forgalomról, majd a vezérlőterembe küldés előtt helyileg feldolgozza azt. Az operátor ennek az információnak a felhasználásával dönt, és utasításokat ad a közlekedési lámpákat vezérlő folyamatnak. Ebben a példában különálló folyamat kezeli az érzékelőket, a vezérlőtermet és a közlekedési lámpákat. Ezek a logikai folyamatok egyszerű folyamatok vagy folyamatcsoportok lehetnek. A példában a folyamatok külön processzorokon futnak.

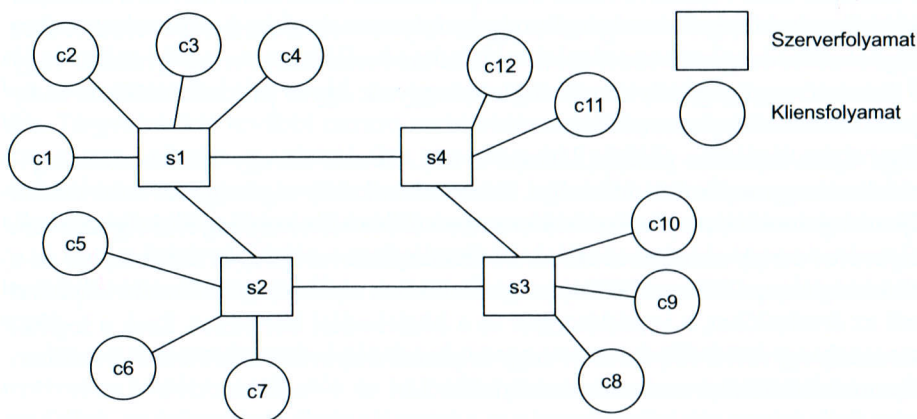


12.1. ábra. Egy többprocesszoros közlekedésirányító rendszer

A több folyamatból álló szoftverrendszerek nem szükségképpen osztottak. Egnél több processzor esetén megvalósítható az osztottság, de a rendszertervezőknek a tervezési folyamat során nem mindig kell törődniük az osztottság kérdéseivel. Az ilyenfajta rendszerek tervezési szempontjai lényegében azonosak a 15. fejezetben leírt valós idejű rendszerekéivel.

12.2. KLIENS-SZERVER ARCHITEKTÚRÁK

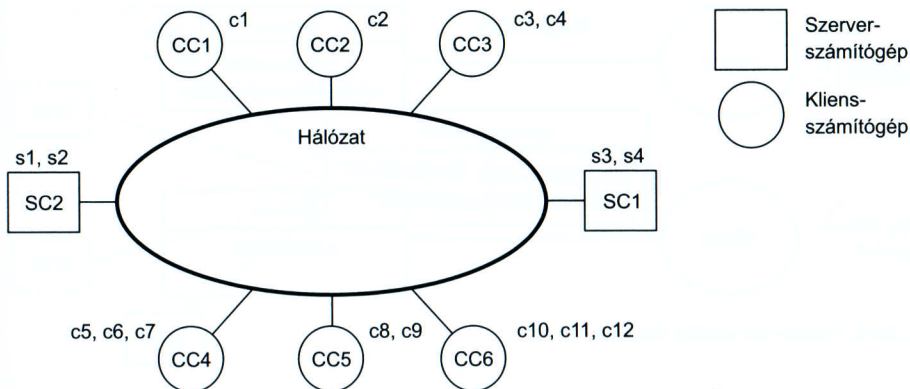
A kliens-szerver architektúrák alapfogalmait a 11. fejezetben már bemutattam. A kliens-szerver architektúrában az alkalmazást szerverek által biztosított szolgáltatásoknak, valamint az ezeket igénybevevő klienseknek a halmazaként modellezzük (Orfali és Harkey, 1998). A klienseknek tudniuk kell az elérhető szerverekről, de általában nem tudnak a többi kliens létezéséről. A kliensek és szerverek – mint ahogy az az osztott kliens-szerver architektúra logikai modelljét bemutató 12.2. ábrán is látható – különálló folyamatok.



12.2. ábra. Egy kliens-szerver rendszer

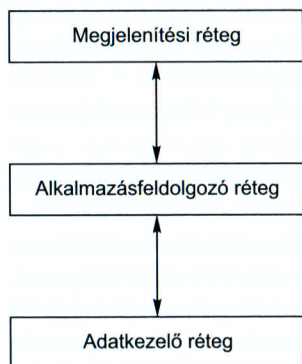
Számos folyamat futhat egyetlen processzoron, ezért a rendszer folyamatai és processzorai közötti kapcsolat számossága nem szükségképpen 1:1. A 12.3. ábrán hat kliens- és két szerverszámítógéppel felálló rendszer fizikai felépítése látható, amelyek a 12.2. ábrán látható kliens- és szerverfolyamatok futtatására képesek. Általában, amikor *kliensekről* és *szerverekről* beszélünk, ezekre a logikai folyamatokra gondolunk, nem pedig azokra a fizikai számítógépekre, amelyeken ezek a folyamatok végbemennek.

A kliens-szerver rendszerek tervezésének az éppen fejlesztett alkalmazás logikai szerkezetét kell tükröznie. Az alkalmazások vizsgálatának egyik módja látható a 12.4. ábrán, amely egy három rétegbe szerveződő alkalmazást mutat be. A megjelenítési réteg az információk a felhasználó számára történő megjelenítésével és a felhasználói interakciókkal foglalkozik. Az alkalmazásfeldolgozó



12.3. ábra. Kliens–szerver hálózatba kötött számítógépek

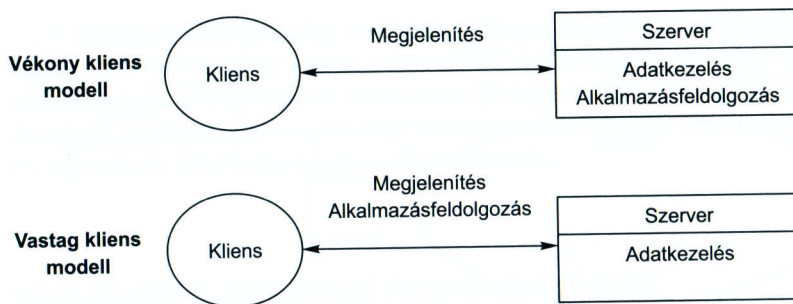
réteg az alkalmazáslogika implementálásával, míg az adatkezelő réteg az adatbázis-műveletekkel foglalkozik. Központosított rendszerek esetén ezeknek nem fontos tisztán elkülönülniük. Osztott rendszerek tervezése esetén azonban szét kell őket választani, mivel lehetővé válik az egyes rétegek különböző számítógépekre történő elosztása.



12.4. ábra. Alkalmazásrétegek

A legegyszerűbb kliens–szerver architektúra a *kétrétegű kliens–szerver architektúra*, ahol az alkalmazás egy szerverből (vagy több azonos szerverből) és több kliensből épül fel. Ahogy az a 12.5. ábrán látható, a kétrétegű kliens–szerver architektúráknak két formája van:

1. *Vékony kliens modell.* Egy vékony kliens modellben minden alkalmazásfeldolgozó és adatkezelő művelet a szerveren megy végbe. A kliens csupán a megjelenítő szoftver futtatásáért felelős.
2. *Vastag kliens modell.* Ebben a modellben a szerver csak az adatkezeléssel törődik, az alkalmazáslogikát és a felhasználóval történő kapcsolattartást a klientsen futó szoftver valósítja meg.



12.5. ábra. Vékony és vastag kliensek

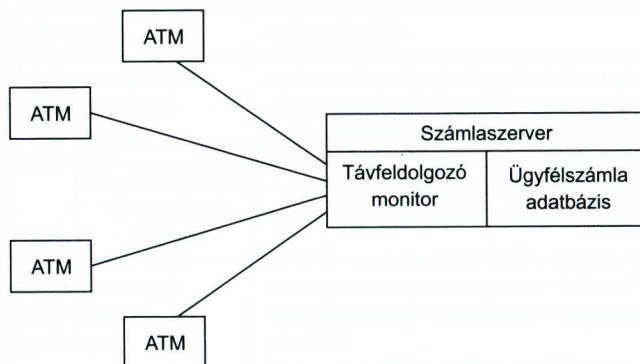
Központosított ősrendszerek (2. fejezet) kliens–szerver felépítésű rendszereké alakítása esetén legegyszerűbb egy vékony klienssel rendelkező, kétrétegű architektúrát használni. Ezen rendszerek felhasználói felületét kell átültetni PC-kre, és maga az alkalmazás szerverként üzemel, kezelve az alkalmazás feldolgozásának és az adatok kezelésének műveleteit. A vékony kliens modell abban az esetben is kialakítható, ha a kliensek egyszerű hálózati eszközök, nem pedig PC-k vagy munkaállomások. A hálózati eszköz futtat egy internetböngészőt, amin keresztül a felhasználói felület implementálva van.

A vékony kliens modell egyik fő hátránya, hogy mind a szervert, mind pedig a hálózatot nagy terhelésnek teszi ki. Minden számításért a szerver a felelős, és ez a kliens és a szerver között jelentős hálózati forgalmat generálhat. A modern PC-kben sok olyan feldolgozási tartalék van, amelyet a vékony kliens architektúra nagyrészt kihasználatlanul hagy.

Ezzel szemben a vastag kliens modell kihasználja ezt az elérhető feldolgozási tartalékot, és mind az alkalmazáslogika feldolgozásának, mind pedig a megjelenítésnek a feladatát a kliensre osztja. A szerver lényegében egy tranzakciószerver, amely az adatbázis-tranzakciókat kezeli. Ennek az architektúrának jól ismert példái a banki ATM-rendszerek, ahol az ATM a kliens, a szerver pedig az ügyfél számlaadatbázisát futtató nagyszámítógép. A pénzkidó automata hardvere sok, a tranzakcióval kapcsolatba hozható, ügyféllel kapcsolatos feldolgozást elvégez.

A 12.6. ábra ezt az osztott ATM-rendszert ábrázolja. Figyeljük meg, hogy az ATM-ek nem közvetlenül az ügyfél adatbázisához, hanem egy távoli feldolgozást felügyelő úgynevezett távfeldolgozó monitorhoz kapcsolódnak. Egy távfeldolgozó monitor vagy tranzakciókezelő olyan köztes termék, amely szervezi a távoli kliensek kommunikációját és szériázza a kliens tranzakcióit, hogy azokat az adatbázis fel tudja dolgozni. A szériázott tranzakciók használata azt jelenti, hogy a rendszer hiba esetén adatvesztés nélkül helyreállítható.

A vastag klienseken alapuló modell ugyan hatékonyabban osztja el a feldolgozást, mint a vékony klienseken alapuló modell, de a rendszermenedzselés bonyolultabb vastag kliensek használata esetén. Az alkalmazás funkcionalitása sok különböző számítógép között van szétszórva. Ha az alkalmazást meg kell változtatni, akkor az magával vonja a kliensekre való újratelepítést. Ez több száz klienssel rendelkező rendszer esetén igen nagy költségekkel járhat.

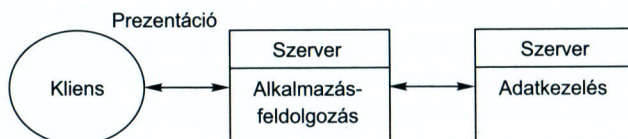


12.6. ábra. Egy kliens–szerver ATM-rendszer

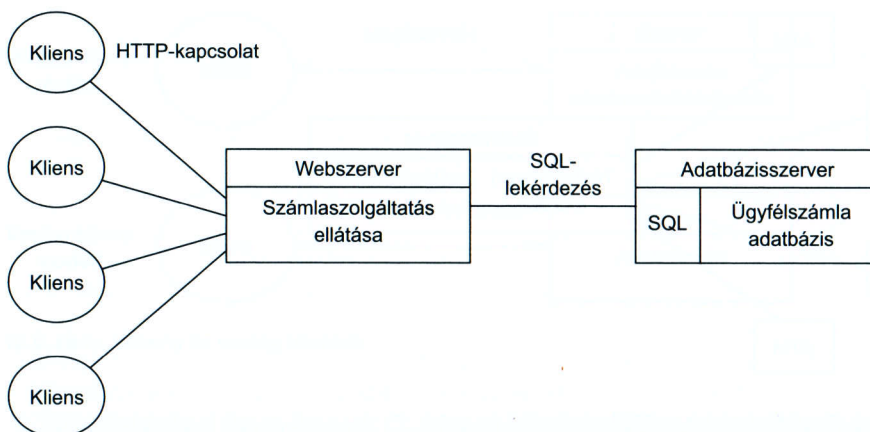
A szerverről kliensre letölthető mobil kód (mint például Java-appletek és ActiveX-vezérlők) megjelenése olyan kliens–szerver modell kifejlődését tette lehetővé, amely valahol a vékony és a vastag kliens modellek között helyezkedik el. Az alkalmazást futtató szoftverek mobil kódként letölthetők a kliensre, így módon könnyítve a szerver terhelésén. A felhasználói felület egy, a letöltött kódokat futtatni képes webböngésző segítségével épül fel.

A kétrétegű kliens–szerver szemléletmód alapvető problémáját az jelenti, hogy a három logikai réteget – megjelenítési, alkalmazásfeldolgozási és adatkezelési – két számítógéprendszerre képezi le. Problémák adódhatnak továbbá vékony klienseken alapuló modellek esetén a skálázhatósággal és a teljesítménnyel, vastag kliens modell esetében pedig a rendszermenedzseléssel. Ezeknek a problémáknak az elkerülésére egyfajta megoldási mód a *háromrétegű kliens–szerver architektúra* használata (12.7. ábra). Ebben az architektúrában a megjelenítés, az alkalmazásfeldolgozás és az adatkezelés különböző processzorokon futó logikailag különálló folyamatok.

Egy háromrétegű kliens–szerver architektúra segítségével megvalósított rendszerre példa egy internetes banki rendszer (12.8. ábra). A bank ügyfeleinek adatbázisa (amely általában egy nagyszámítógépen helyezkedik el) biztosítja az adatkezelési szolgáltatásokat, egy webszerver biztosítja az alkalmazás szolgáltatásait, például a készpénzátutalásra, kimutatások generálására, számlafizetésre stb., a felhasználó saját számítógépe pedig egy internetböngészővel rendelkező kliens. Ez a rendszer skálázható, mivel az ügyfelek számának növekedésével együtt viszonylag könnyen bővíthető új webszerverek hozzáadásával.



12.7. ábra. Egy háromrétegű kliens–szerver architektúra



12.8. ábra. Egy internetes banki rendszer osztott architektúrája

A háromrétegű architektúra használata ebben az esetben lehetővé teszi a webszerver és az adatbázisszerver közötti információáramlás optimalizálását. Az ilyen rendszerek közötti kommunikációra gyors, alacsonyabb szintű kommunikációs protokollokat is használhatunk. Az információ adatbázisból történő kinyerésére egy SQL nyelvű adatbázis-lekérdezéseket támogató hatékony köztes termék használható.

Néhány esetben célszerű a háromrétegű kliens–szerver modell többrétegűvé bővítése, ahol a rendszerhez további szerverek kapcsolódnak. A többrétegű rendszerek akkor használhatók, ha az alkalmazásoknak különböző adatbázisokhoz kell hozzáférniük és azokat használniuk. Ebben az esetben az alkalmazásszerver és az adatbázisszerverek közé egy integrációs szervert kell elhelyezni, amely összegyűjti az osztott adatokat, és úgy jeleníti meg az alkalmazás számára, mintha azok egyetlen adatbázisból valók volnának.

Az alkalmazásfeldolgozást több szerver között elosztó három- és többrétegű kliens–szerver architektúrák természetüknél fogva skálázhatóbbak a kétrétegű architektúráknál. A vékony kliens elvén működő kétrétegű architektúrákhoz képest kisebb a hálózati forgalom. Az alkalmazásfeldolgozás a rendszer legváltozékonyabb része és központi elhelyezkedése miatt könnyen frissíthető. A feldolgozás, bizonyos esetekben, elosztható az alkalmazáslogikát végrehajtó és az adatkezelést végző szerverek között, ami még gyorsabb válaszadást tesz lehetővé a kliensek kéréseire.

A kliens–szerver architektúrák tervezőinek a legmegfelelőbb architektúra kiválasztásakor számos tényezőt kell figyelembe venniük. A 12.9. ábrán azok a szituációk láthatók, amelyekben az általam tárgyalt kliens–szerver architektúrák valószínűleg megfelelőek.

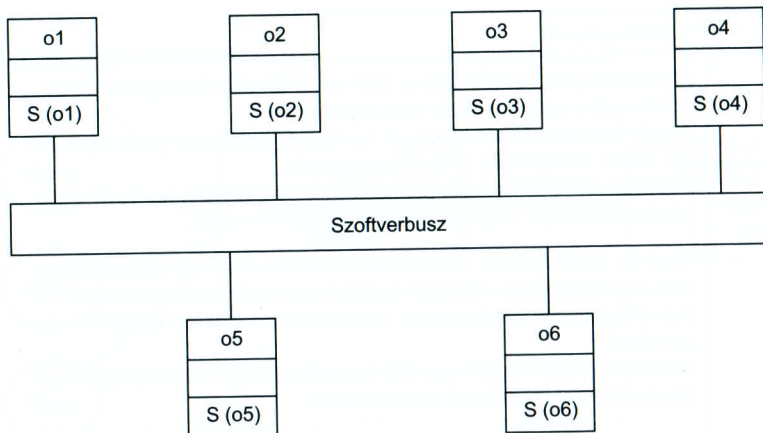
Architektúra	Alkalmazások
Vékony kliens elven működő kétrétegű kliens–szerver architektúra	Az ősrendszerek alkalmazásai, ahol az alkalmazásfeldolgozás és az adatkezelés szétválasztása nem célravezető. A számításintenzív alkalmazások kevés adatkezeléssel vagy adatkezelés nélkül, mint például a fordítóprogramok. Adatintenzív alkalmazások (böngészés és lekérdezés) kevés alkalmazásfeldolgozással vagy alkalmazásfeldolgozás nélkül.
Vastag kliens elven működő kétrétegű kliens–szerver architektúra	Azok az alkalmazások, amelyekben a kliensoldali alkalmazásfeldolgozást egy kulcsrakész rendszer (például Microsoft Excel) biztosítja. Számításintenzív alkalmazások adatkezeléssel (például grafikus megjelenítés). Viszonylag stabil végfelhasználói funkciókkal és jól megalapozott rendszerkezeléssel rendelkező alkalmazások.
Három- vagy több-rétegű kliens–szerver architektúra	Kliensek százaival vagy ezreivel rendelkező széles körű alkalmazások. Azok az alkalmazások, ahol mind az adatok, mind pedig az alkalmazás változékonyak. Különböző forrásokból származó adatok integrálását végző alkalmazások.

12.9. ábra. A különböző kliens–szerver architektúrák alkalmazási területei

12.3. OSZTOTT OBJEKTUMARCHITEKTÚRÁK

Egy osztott rendszer kliens–szerver modelljében a kliensek és a szerverek különböznek egymástól. A kliensek a szerverek, nem pedig a többi kliens szolgáltatásait vehetik igénybe; a szerverek működhetnek kliensként más szerverek szolgáltatásainak igénybevételével, de nem kérhetnek szolgáltatásokat kliensektől; a klienseknek ismerniük kell a specifikus szerverek által kínált szolgáltatásokat és tudniuk kell, hogy hogyan tudnak kapcsolatba lépni ezekkel a szerverekkel. Ez a modell sokféle alkalmazás esetén jól működik, azonban korlátozza a rendszertervezők mozgásterét abban, hogy el kell dönteniük, hogy a szolgáltatások hol legyenek biztosítva. Meg kell tervezniük a skálázhatóságot is, azaz valamilyen módon biztosítaniuk kell a szerverek közötti – a kliensek rendszerhez történő hozzáadásával kialakuló – terhelés elosztását.

Az osztott rendszerek tervezésének általánosabb megközelítése a kliensek és szerverek közötti megkülönböztetés megszüntetése és a rendszer felépítésének osztott objektumarchitektúráként történő megtervezése. Egy osztott objektumarchitektúrában (12.10. ábra) az alapvető rendszerkomponensek az objektumok, amelyek interfészeket biztosítanak a saját maguk által nyújtott szolgáltatásokhoz. A többi objektum ezeket a szolgáltatásokat hívja anélkül, hogy a kliens (azaz a szolgáltatás igénybe vevője) és a szerver (a szolgáltatás nyújtója) között bármilyen logikai megkülönböztetést tenne.



12.10. ábra. Osztott objektumarchitektúra

Az objektumok egy hálózaton belül több számítógép között oszthatók el és köztes termékeken keresztül kommunikálnak egymással. Ezt a köztes terméket ORB-nek hívjuk. Ennek feladata egy észrevehetetlen, objektumok között húzódó interfész biztosítása. Az ORB-kről a 12.3.1. alfejezetben lesz szó.

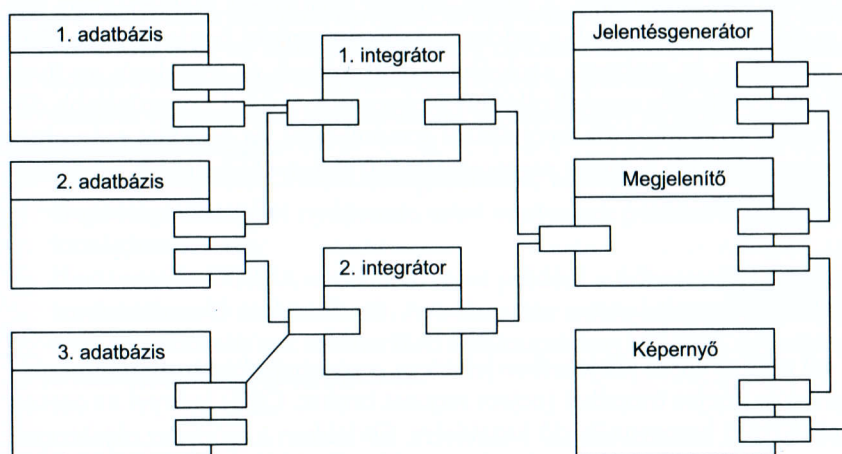
Az osztott objektummodell a következő előnyökkel bír:

- Megteremti a rendszertervező számára a szolgáltatásnyújtás helyére és módjára vonatkozó döntések elhalasztásának lehetőségét. A szolgáltatást biztosító objektumok a hálózat bármely csomópontján elhelyezkedhetnek, ezért a vastag, illetve vékony kliens elvén alapuló modellek közötti különbségtétel elveszti lényegét, mivel így nem szükséges előre eldönteni, hogy az alkalmazáslogika objektumai hol helyezkedjenek el.
- Ez egy nagyon nyílt rendszer-architektúra, amely lehetővé teszi az új erőforrások igény szerinti hozzáadását. Mint ahogy azt a következő szakaszban látni fogjuk, olyan szoftverbusz-szabványokat fejlesztettek ki és valósítottak meg, amelyek lehetővé teszik, hogy különböző programozási nyelven írt objektumok kommunikáljanak egymással és nyújtsanak szolgáltatásokat egymásnak.
- A rendszer rugalmas és skálázható. A rendszer olyan különböző példányai, amelyek különböző vagy többszörözött objektumok által ugyanazt a szolgáltatást biztosítják, úgy is létrehozhatók, hogy azok különböző rendszerterheléssel birkózzanak meg. A rendszer terhelésének növekedésével a rendszer többi objektumának megváltoztatása nélkül adhatók az új objektumok a rendszerhez.
- Lehetőség van a rendszer dinamikus újrakonfigurálására az objektumok hálózaton keresztül történő igény szerinti vándoroltatásával. Ez fontos lehet ott, ahol a szolgáltatásokra vonatkozó igény igen változó. Egy szolgáltatást biztosító objektum átvándorolhat ugyanarra a processzorra, mint ahol a szolgáltatást igénybe vevő objektumok elhelyezkednek, így módon növelve a rendszer teljesítményét.

Az osztott objektumarchitektúra használható logikai modellként, amely lehetővé teszi a rendszer strukturálását és szervezését. Ebben az esetben azon kell elgondolkoznunk, hogy hogyan tudjuk az alkalmazás funkcionalitását kizárólag a szolgáltatásoknak, illetve azok kombinációinak segítségével biztosítani. Ezt követően ki kell dolgozni egy módszert arra, hogy hogyan lehet biztosítani a szolgáltatásokat osztott objektumokkal. Ezen a szinten a megtervezett objektumok általában szakterület-specifikus szolgáltatásokat nyújtó durva szemcsézettségű objektumok (ezeket néha *üzleti objektumoknak* is nevezik). Például egy kiskereskedelmi alkalmazás tartalmazhat az árukészlet ellenőrzésével, az ügyfelekkel való kommunikációval, az árucikkek rendelésével stb. foglalkozó üzleti objektumokat. Természetesen ez a logikai modell később implementációs modellként megvalósítható.

Egy osztott objektumarchitektúrát emellett kliens–szerver rendszerek megvalósítására is használhatunk. Ebben az esetben a rendszer logikai modellje egy kliens–szerver modell, de mind a kliensek, mind pedig a szerverek osztott objektumok, amelyek egymással egy szoftverbuszon keresztül kommunikálnak. Ez könnyen változtathatóvá teszi a rendszert, például kétrétegűről többretegűre. Ebben az esetben a szerver vagy a kliens nem implementálható egyetlen osztott objektumként, de tartalmazhat több, kisebb objektumot, amelyek mindegyike szakosított szolgáltatásokat nyújt.

Az osztott objektumarchitektúrára példa az adatbányászati rendszer, ahol a különböző adatbázisokban elhelyezkedő adatok közötti összefüggések felderítése a cél (12.11. ábra). Adatbányászati alkalmazás például egy olyan kiskereskedelmi cég, amelynek különféle (mondjuk élelmiszer és vegyes iparcikk) boltjai vannak, és kapcsolatot próbál keresni a különböző ételfajták és a különböző iparcikkek vásárlása között. Például a bébietelt vásárló emberek bizonyosfajta tapétát részesítenek előnyben. Ennek a tudásnak a birtokában a cég kimondottan a bébietelt vásárlókat tudja kombinált ajánlatokkal megcélozni.



12.11. ábra. Egy adatbányászati rendszer osztott architektúrája

Ebben a példában minden adatbázis egy olyan interfésszel rendelkező osztott objektumként tekinthető, amely csak olvasható hozzáférést biztosít annak adataihoz. Az integrátorobjektumok mindegyike kapcsolatok bizonyos típusaival foglalkozik, összegyűjtik az összes adatbázisból az információkat, és megpróbálják kikövetkeztetni a kapcsolatokat. Létezhet egy olyan integrátorobjektum, amely az árucikkek eladásának időszakos változásaival törődik, és egy másik, amely a különböző típusú árucikkek közötti kapcsolatokkal foglalkozik.

A megjelenítő objektumok együttműködnek az integrátorobjektumokkal a feltárt kapcsolatok megjelenítése vagy az azokról szóló jelentések elkészítése céljából. Mivel nagy mennyiségű adat kerül kezelésre, a megjelenítő objektumok a feltárt kapcsolatok megjelenítését normál körülmények között grafikus módon végzik. A grafikus információmegjelenítést a 16. fejezetben tárgyalom.

Az ilyen alkalmazások számára az osztott objektumarchitektúra három okból megfelelőbb, mint a kliens-szerver architektúra:

1. Ellentétben mondjuk egy banki ATM-rendszerrel, a rendszer logikai modellje nem valamilyen szolgáltatás biztosítása, amelyben megkülönböztetett adatkezelési szolgáltatások vannak.
2. Lehetséges az elérhető adatbázisok számának növelése a rendszer megbontása nélkül. Minden adatbázis egyszerűen egy osztott objektum lesz. Az adatbázis-objektumok az adatok elérését egyszerű interfészekon keresztül biztosítják. Az elért adatbázisok különböző gépeken helyezkedhetnek el.
3. Új integrátorobjektumok hozzáadásával lehetővé teszi újfajta kapcsolatok kibányászását is. A cég specifikus kapcsolatokban érdekelt részei képesek kibővíteni a rendszert olyan integrátorobjektumok hozzáadásával, amelyek a cég egyes részlegeinek gépein működnek anélkül, hogy a máshol használt integrátorokra vonatkozó bármilyen tudásra szükség lenne.

Az osztott objektumarchitektúrák fő hátránya az, hogy tervezésük nehezebb a kliens-szerver rendszerekénél. A kliens-szerver rendszerek meglehetősen természetes módon gondolkodnak a rendszerekről. Az emberi ügyintézés módjait tükrözik, amelyben az emberek szolgáltatásokat kérnek és fogadnak az ilyen szolgáltatások nyújtására specializálódott emberektől. Sokkal bonyolultabb dolog általánosabb szolgáltatások nyújtásáról gondolkodni, és egyelőre még nincs túl nagy tapasztalatunk a durva szemcsézettségű üzleti objektumok tervezésében és fejlesztésében.

12.3.1. CORBA

Ahogy azt már az előző alfejezetben jeleztem, az osztott objektumarchitektúra implementációja köztes terméket (object request broker, ORB) igényel az osztott objektumok közötti kommunikáció kezelésére. Elviekben a rendszer objektumai különféle programozási nyelveken implementálhatók, különböző platformokon futhatnak, és a rendszer többi objektumainak nem kötelező ismerni a neveiket. A

„ragasztó” köztes terméknek ezért rengeteg tennivalója van az objektumkommunikáció észrevétlenségének biztosítására.

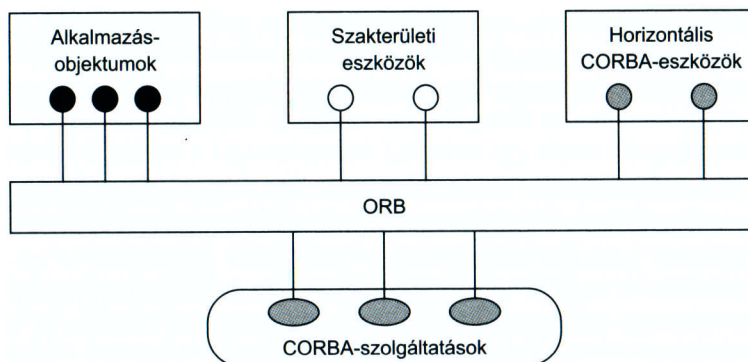
Az osztott objektumok támogatását végző köztes termékekre két szinten van szükség:

1. A logikai kommunikáció szintjén a köztes termék olyan funkciókat biztosít, amelyek lehetővé teszik, hogy a különböző számítógépeken elhelyezkedő objektumok adatokat és vezérlési információkat cseréljenek egymással. A különböző platformokon a logikai objektumkommunikáció megkönnyítésére különböző szabványokat dolgoztak ki, mint amilyen például a CORBA és a COM (Pritchard, 1999).
2. A komponens szintjén a köztes termék alapot nyújt kompatibilis komponensek fejlesztéséhez. Az EJB- és CORBA-komponensek és az ActiveX-vezérlők (Szyperski, 2002) az egyéb komponensek által lekérdezhető és felhasználható szabványos módszerek segítségével megvalósítandó komponensek számára nyújtanak alapvető eszközöket. Komponensszabványokról a 19. fejezetben írok.

Ebben a szakaszban a logikai objektumkommunikációra szolgáló köztes termékekre helyezzük a hangsúlyt, és megvizsgáljuk, hogyan támogatják ezt a CORBA-szabványok. Ezeket a szabványokat az objektumorientált fejlesztéssel kapcsolatos szabványokat alkotó Object Management Group (OMG) definiálta. A definiált szabványok az OMG webhelyén a nyilvánosság számára ingyenesen hozzáférhetők.

A 12.12. ábra az OMG osztott alkalmazásokra vonatkozó elképzelését ábrázolja, amelyet az Object Management Architecture Siegel-féle diagramjából (Siegel, 1998) vettem át. Ez az osztott alkalmazások alábbi komponensekből történő felépítését javasolja:

1. Az alkalmazás számára tervezett és implementált alkalmazásobjektumok.
2. Az OMG által az adott szakterületre definiált szabványos objektumok. Ezek között a szakterületi objektumszabványok között megtaláljuk a pénzügyi/biztosítási, elektronikus kereskedelmi, egészségügyi és egyéb szakterületek objektumszabványait.
3. Azok az alapvető CORBA-szolgáltatások, amelyek osztott számítástechnikai alapszolgáltatásokat nyújtanak, mint amilyenek például a könyvtárak, a biztonságkezelés stb.
4. Horizontális CORBA-eszközök, mint például a felhasználói felület eszközei, rendszerkezelő eszközök stb. A *horizontális eszköz* kifejezés azt sugallja, hogy ezek az eszközök sok szakterületi alkalmazásban közösek, és ezért különböző alkalmazásokban használhatók.



12.12. ábra. Egy CORBA-alapú osztott alkalmazás struktúrája

A CORBA-szabványok ezt az elképzelést minden szempontból lefedik. Ezek a szabványok négy fő elemből állnak:

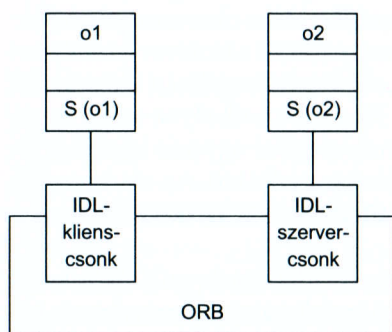
1. Az alkalmazás objektumainak egy objektummodelljéből, ahol egy CORBA-objektum IDL-ben (Interface Definition Language) leírt jól definiált, nyelvfüggetlen interfésszel rendelkezik.
2. Egy ORB-ből, amely az objektumszolgáltatásokra vonatkozó kérélmeket kezeli. Az ORB megállapítja a szolgáltatást biztosító objektum helyét, felkészíti a kérelemre, elküldi a szolgáltatásra vonatkozó kérélmeket, és az eredményt pedig visszaadja a kérelmezőnek.
3. Olyan általános objektumszolgáltatások halmazából, amely szolgáltatásokra valószínűleg sok osztott alkalmazás fog igényt tartani. Erre példa a Directory, a Transaction és a Persistence szolgáltatás.
4. Az alkalmazások által igényelhető, ezeken az alapszolgáltatásokon felépülő, közös komponensek halmazából. Ezek lehetnek vertikális, szakterület-specifikus komponensek, illetve sok alkalmazás által használt horizontális, általános célú alkalmazások is.

A CORBA objektummodellje egy objektumot attribútumok és szolgáltatások egységbe zárásának tekint, mint ahogyan az objektumok esetén ez szokásos. A CORBA-objektumoknak azonban rendelkezniük kell különálló interfészdefiniációval, amely az objektum nyilvános adatait és műveleteit definiálja. A CORBA-objektumok interfészei szabványos, nyelvfüggetlen interfészdefiniációs nyelven adhatók meg. Ha egy objektum egy másik objektum által biztosított szolgáltatásokat szeretné igénybe venni, ezeket a szolgáltatásokat az IDL-interfészen keresztül tudja elérni. A CORBA-objektumok rendelkeznek egy Interoperable Object Reference (IOR) nevű egyedi azonosítóval. Ezt az IOR-t használják, amikor egy objektum szolgáltatást kér egy másiktól.

Az ORB ismeri a szolgáltatásokat kérő objektumokat és azok interfészeit, és kezeli az objektumok közötti kommunikációt. A kommunikációt folytató objektumoknak sem azt nem kell tudniuk, hogy hol van a másik objektum, sem pedig azt,

hogyan van implementálva. Ugyanúgy, ahogy az IDL-interfész elszigeteli az objektumokat az ORB-től, lehetőség van az objektum implementációjának teljesen átlátszó módon történő lecserélésére. A hívások között az objektum helye megváltozhat, és a rendszer többi objektuma számára ez is teljesen átlátszó.

A 12.13. ábra azt mutatja be, hogy az o1 és o2 objektumok hogyan kommunikálhatnak az ORB-n keresztül. A hívó objektumhoz (o1) hozzá van rendelve egy IDL-klienscsonk, amely definiálja az igényelt szolgáltatást biztosító objektum interfészét. Az o1 implementálását végző programozó az objektum implementációjában ezt a klienscsonkot hívja meg, ha egy szolgáltatásra szükség van. Az IDL a C++ kibővítése, így ez a klienscsonk nagyon könnyen elérhető akkor, ha C++-ban programozunk, és viszonylag könnyen elérhető, ha C-ben vagy Javában dolgozunk. Olyan nyelvekre is létezik IDL-leképezés, mint például az Ada és a COBOL.

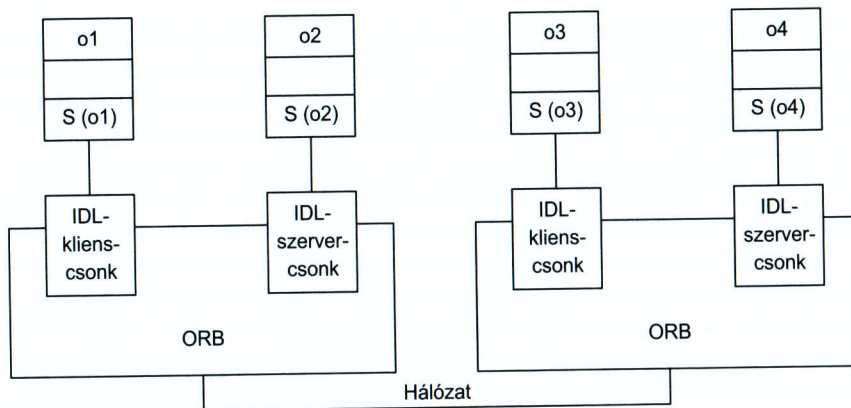


12.13. ábra. Objektumok kommunikációja ORB-n keresztül

A szolgáltatást biztosító objektumhoz egy IDL-szervercsonk van rendelve, amely összekapcsolja az interfészt a szolgáltatások implementációjával. Leegyszerűsítve, amikor az interfészen keresztül egy szolgáltatást hívnak, az IDL-szervercsonk ezt lefordítja az alkalmazott implementációs nyelvű szolgáltatáshívássá. A módszer vagy eljárás végrehajtása után az IDL-szervercsonk az eredményeket visszafordítja IDL-re, így azokhoz a hívó objektum is hozzáférhet. Ha egy objektum egyszerre biztosít szolgáltatásokat más objektumoknak és vesz igénybe máshol szolgáltatásokat, akkor szüksége van IDL-szervercsonkra és IDL-klienscsonkra egyaránt. IDL-klienscsonkra minden objektumnak szüksége van.

Az ORB-eket általában nem különálló folyamatokként implementálják, hanem ezek olyan könyvtárak összessége, amelyek összekapcsolhatók az objektumok implementációjával. Éppen ezért egy osztott rendszerben minden, osztott objektumokat futtató számítógépnek saját ORB-je lesz, ami az objektum helyi hívásait kezeli. Abban az esetben azonban, ha egy távoli objektum által biztosított szolgáltatásra vonatkozó kérés érkezik, ORB-közi kommunikációra van szükség.

Ez a helyzet látható a 12.14. ábrán. Ebben a példában az o1 és o2 objektumok valamelyike az o3 vagy o4 valamelyikétől igényel szolgáltatást; ekkor az összekapcsolt ORB-k kommunikációja szükséges. Egy CORBA-implementáció az ORB-ORB kommunikációt oly módon támogatja, hogy minden ORB számára



12.14. ábra. ORB-közi kommunikáció

hozzáférést biztosít az összes IDL-interfészhez, és implementálja a Generic Inter-ORB Protocol (GIOP) nevű OMG-szabványt. Ez a protokoll olyan szabványos üzeneteket definiál, amelyeket az ORB-k ki tudnak cserélni egymás között, a távoli objektumhívás és információátvitel megvalósítása céljából. Az alacsonyabb szintű TCP/IP-protokollal kombinálva a GIOP lehetővé teszi az ORB-k számára az interneten keresztül történő kommunikációt.

A CORBA-kezdemenyezés az 1980-as években kezdődött és a CORBA korai verziói csak az osztott objektumok támogatásával foglalkoztak, a szabványok fejlődésével azonban átfogóbbá váltak. Az osztott objektumok kommunikációjának mechanizmusán kívül, a CORBA-szabványok már néhány, az osztott objektum-orientált alkalmazásokat segítő szabványos szolgáltatást is nyújtanak.

A CORBA-szolgáltatásokra úgy tekinthetünk, mint olyan eszközökre, amelyeket valószínűleg sok osztott rendszer igényel. A szabványok körülbelül 15 gyakori szolgáltatást definiálnak. A következő felsorolás ezen általános szolgáltatások közül mutat be néhányat:

1. Naming és Trading szolgáltatások, amelyek a hálózat többi objektumára történő hivatkozást és azok felfedezését teszik lehetővé. A Naming szolgáltatás olyan könyvtárszolgáltatás, amely az objektumok elnevezését és más objektumok által történő felfedezését teszi lehetővé. Ez egy telefonkönyvhöz hasonlít. A Trading szolgáltatásokat a szaknévsorhoz hasonlíthatjuk: az objektumok megtudhatják, hogy milyen más objektumok vannak bejegyezve a Trading szolgáltatásnál, és hozzáférhetnek az objektumok specifikációjához.
2. Notification szolgáltatások, amelyek lehetővé teszik az objektumoknak, hogy bizonyos események bekövetkezése esetén értesítsenek más objektumokat. A szolgáltatás segítségével az objektumok regisztrálhatják egy bizonyos eseményre vonatkozó érdeklődésüket, és így automatikusan értesülnek az adott esemény bekövetkezéséről. Tegyük fel például, hogy rendszerünk tartalmaz egy nyomtatási puffert, amely sorba állítja a nyomtatandó dokumentumokat, valamint több nyomtatóobjektumot. A nyomtatási puffer bejegyzí a nyomtató-

objektum „nyomtatás vége” eseményére vonatkozó érdeklődését. A Notification szolgáltatás tudatja vele, hogy a nyomtatás kész, és így az adott nyomtató képes a következő dokumentum fogadására.

3. Transaction szolgáltatások, amelyek támogatják az atomi tranzakciókat és a hiba esetén történő visszagörgetést. A tranzakciók olyan hibatűrő eszközök (18. fejezet), amelyek támogatják a frissítési műveletek során bekövetkező hibákból történő helyreállítást. Ha egy objektum frissítési művelete kudarcba fullad, az objektum állapota visszagörgethető a frissítés kezdete előtti állapotba.

A CORBA-szabványok horizontális és vertikális komponensek széles köre számára tartalmaznak olyan interfész-definíciókat, amelyeket az osztott alkalmazások készítői használhatnak. A vertikális komponensek a szakterületi alkalmazások jellegzetes komponensei. A horizontális komponensek olyan általános célú komponensek, mint például a felhasználói felületek komponensei. A horizontális és vertikális CORBA-komponensek specifikációjának elkészítése hosszú folyamat, a jelenleg elérhető specifikációk az OMG weblapján megtalálhatók.

12.4. SZERVEZETKÖZI OSZTOTT RENDSZEREK

Biztonsági és együttműködési okokból osztott rendszereket elsősorban szervezeti szinten készítenek. Egy szervezet számos szerverrel rendelkezik, amelyek között a számítási terhelés elosztható. Mivel ezek ugyanazon szervezeten belül helyezkednek el, helyi szabványok és működési folyamatok alkalmazhatók. Habár webalapú rendszerek esetén a kliensszámítógépek sokszor kívül esnek a szervezeti határokon, ezek feladata csupán a felhasználói felület futtatása.

Azonban már léteznek az osztott feldolgozásnak olyan újabb modelljei, amelyek nemcsak szervezeten belüli, hanem szervezeten kívüli osztott feldolgozást is lehetővé tesznek. Ebben az alfejezetben két ilyen is tárgyalunk. A peer-to-peer feldolgozás önálló hálózati csomópontok által végzett feldolgozáson alapulnak. A szolgáltatásorientált rendszerek osztott objektumok helyett osztott szolgáltatásokon alapulnak, az adatcserére pedig XML-alapú szabványokat használnak.

12.4.1. Peer-to-peer architektúrák

A peer-to-peer (p2p) rendszerek olyan decentralizált rendszerek, ahol a feldolgozást a hálózat bármely csomópontja végezheti, és – legalábbis elviekben – nem különböztetünk meg klienseket és szervereket. Peer-to-peer alkalmazások esetén a teljes rendszert úgy tervezzük meg, hogy az kihasználja a számítógépek hatalmas hálózata által biztosított számítási teljesítmény és tárolókapacitás előnyeit. A csomópontok közötti kommunikációt lehetővé tévő szabványok és protokollok magába az alkalmazásba vannak beágyazva, és minden csomópont ennek egy példányát futtatja.

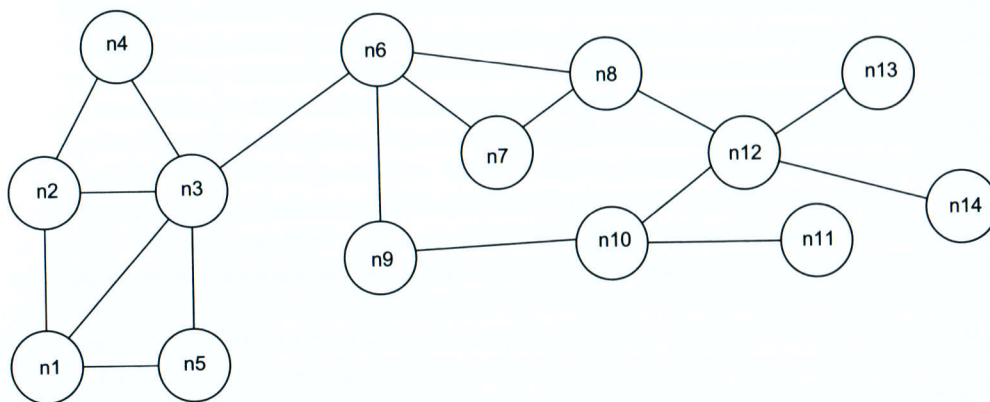
A könyv írásának idején a peer-to-peer technológiát leginkább személyes rendszerek (Oram, 2001) esetén használják. Például a Gnutella és Kazaa protokollokon alapuló állománycserélő rendszerek segítségével a felhasználó megoszthatja a saját PC-jén lévő állományokat, az olyan azonnali üzenetküldő rendszerek, mint az ICQ vagy a Jabber, pedig a felhasználó köztes szerver beiktatása nélküli közvetlen kommunikációját biztosítják. A SETI@home egy olyan hosszú lefutású projekt, amely a rádióteleszkópokból érkező adatokat otthoni PC-k segítségével dolgozza fel a földöntúli élet jelei után kutakodva. A Freenet egy decentralizált adatbázis, amelyet úgy terveztek meg, hogy könnyebb legyen névtelenül információkat publikálni, így nehezítve a hatóságok dolgát, hogy azok elhallgassanak bizonyos információkat.

Ezt a technológiát azonban az üzleti életben is egyre inkább alkalmazzák, hogy kihasználják a PC-k alkotta hálózatban rejlő erőforrásokat (McDougall, 2000). Az Intel és a Boeing egyaránt p2p rendszert készített számításgényes alkalmazásai számára. Az osztott munkavégzést támogató kooperatív alkalmazások számára ez tűnik a leghatékonyabb technológiának.

A p2p alkalmazások architektúráját két szempontból vizsgálhatjuk. A logikai hálózati architektúra a rendszer elosztásának architektúrája, míg az alkalmazás-architektúra az egyes alkalmazásfajták komponenseinek általános elrendezését jelenti. Ebben a szakaszban a logikai hálózati architektúra két alapvető fajtája kerül a középpontba: a decentralizált architektúrák és a félig centralizált architektúrák.

A peer-to-peer rendszerekben elvileg a hálózat minden csomópontja tud az összes többi csomóponttól, kapcsolatot tud velük létesíteni, és adatokat tud velük cserélni. A gyakorlatban azonban ez lehetetlen, ezért a csomópontokat „lokalitásokba” szervezzük, és néhány csomópont majd az egyéb lokalitásokhoz vezető hídként jelenik meg. Ez a decentralizált p2p architektúra látható a 12.15. ábrán.

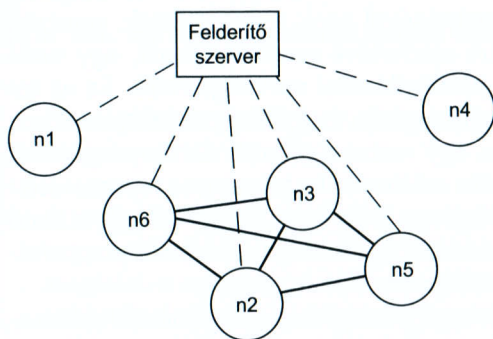
Egy decentralizált architektúrában a hálózat csomópontjai nem csupán funkcionális elemek, hanem kommunikációs kapcsolóelemek is, amelyek képesek az adatokat és vezérlőjeleket egyik csomópontból a másikba juttatni. Tegyük fel pél-



12.15. ábra. Decentralizált p2p-architektúra

dául, hogy a 12.15. ábra egy decentralizált dokumentumkezelő rendszert ábrázol. Ezt a rendszert kutatók egy csoportja használja dokumentumok megosztására, és minden kutató a saját dokumentumtárát tartja karban. Azonban ha valaki lekért egy dokumentumot, az a többi csomópont számára is elérhetővé válik. Akinek valamelyik dokumentumra szüksége van, kiad egy keresési parancsot, amely az adott „lokalitásba” tartozó csomópontokhoz lesz elküldve. Ezek a csomópontok megnézik, hogy rendelkeznek-e az adott dokumentummal, és ha igen, akkor eljuttatják azt a kérelmezőnek. Ha nincs meg nekik, akkor továbbítják ezt a keresést további csomópontok felé, és ha a dokumentum végül előkerül, a csomópont el tudja irányítani azt az eredeti kérelmezőhöz. Ezért ha az $n1$ csomópont egy $n10$ -nél tárolt dokumentumot keres, a keresés az $n3$, $n6$, $n9$ csomópontok érintésével jut el $n10$ -be.

Ennek a decentralizált architektúrának nyilvánvaló előnye, hogy nagyon redundáns, ezért hibátűrő, jól viseli, ha bizonyos csomópontok lecsatlakoznak a hálózatról. Azonban az, hogy ugyanazt a keresést több különböző csomópontnak is el kell végeznie, nyilvánvaló többletterhelést ró a rendszerre. A p2p modell egyik alternatívájaként használhatunk egy félig centralizált architektúrát, ahol a hálózat egy vagy több csomópontja szerverként dolgozva könnyíti meg a csomópontok közötti kommunikációt. Ezt a modellt a 12.16. ábra mutatja be.



12.16. ábra. Egy félig centralizált p2p-architektúra

Egy félig centralizált architektúrában a szerver(ek) szerepe a hálózatban lévő felek kapcsolatfelvételének elősegítése és a számítási eredmények koordinálása. Ha például azt feltételezzük, hogy a 12.16. ábra egy azonnali üzenetküldő rendszert ábrázol, akkor a hálózati csomópontok a szerverrel kommunikálva (ezt jelzik a szaggatott vonalak) határozzák meg, hogy mely további csomópontok érhetőek el. Ha ezt felderítette, akkor kialakítható a közvetlen kapcsolat, így a szerverrel való kapcsolattartás szükségtelenné válik. Így az $n2$, $n3$, $n5$ és $n6$ csomópontok közvetlen kommunikációban állnak egymással.

Egy olyan – számításra kihegyezett – p2p rendszerben, ahol processzorintenzív számítások oszlanak meg sok csomópont között, bizonyos csomópontok kiüntetett szereppel rendelkeznek, ők osztják szét a munkát a többi csomópontnak, valamint gyűjtik és ellenőrzik a számítások eredményeit.

Annak ellenére, hogy a peer-to-peer rendszerek nyilvánvaló többletterhelést okoznak, ez sokkal hatékonyabb megközelítés a szervezetközi feldolgozás elvégzésére, mint a következő szakaszban bemutatásra kerülő szolgáltatásorientált megközelítés. A szervezetközi feldolgozás során használt p2p megközelítéssel azonban még vannak bajok: a biztonság és a megbízhatóság kérdése még nem megoldott. Ez azt jelenti, hogy a p2p rendszerek leginkább csak nemkritikus információs rendszerek esetén alkalmazhatók, vagy olyan rendszerek esetén, ahol a szervezetek között már működő kapcsolat van.

12.4.2. Szolgáltatásorientált architektúra

A www fejlődése azt eredményezte, hogy a kliensszámítógépek saját szervezetükön kívül elhelyezkedő távoli szerverekhez is hozzáférnek. Ha ezek a szervezetek információikat HTML-formátumra hozták, akkor azok elérhetővé váltak. Azonban ez a hozzáférés kizárólag csak webböngészőn keresztül volt lehetséges, és az információtárolókhoz történő közvetlen hozzáférés nem volt megvalósítható. Ez azt eredményezte, hogy nem volt mód olyan szerverek közötti kapcsolatokra, amikor mondjuk egy program több katalógust is le szeretne kérdezni.

Ennek a problémának a kiküszöbölése érdekében alakult ki a webszolgáltatások fogalma. Egy webszolgáltatás segítségével azok a szervezetek, amelyek információikat más programok számára elérhetővé szeretnék tenni, egy webszolgáltatás-interfész definiálásával és publikálásával ezt megtehetik. Ez az interfész definiálja az elérhető adatokat, és megadja, hogy hogyan lehet azokhoz hozzáférni. Általánosabban fogalmazva, egy webszolgáltatás a más programok által használható számítási és információs erőforrások szabványos reprezentációja. Így például lehetőségünk nyílik arra, hogy létrehozzunk egy adóügyi iktató szolgáltatást, amely a felhasználók által kitöltött adóbevallási formanyomtatványokat automatikusan ellenőrzi, és továbbítja az adóhatóságnak.

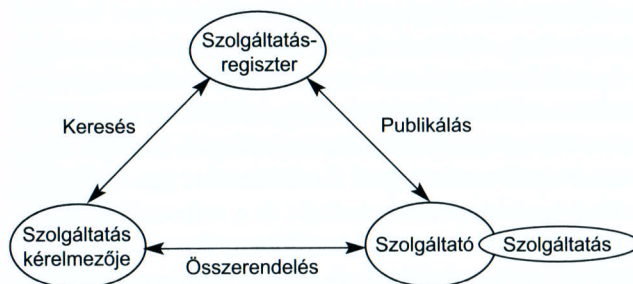
A webszolgáltatás fogalma az általánosabb szolgáltatás fogalomból származik, amely Lovelock és mások (Lovelock és mások, 1996) szerint:

...egyik fél egy másik számára felajánlott tevékenysége. Habár a folyamat egy fizikai termékhez köthető, a cselekmény lényegében eszmei, és normál körülmények között nem eredményezi a termelési tényezők tulajdonjogát.

Egy szolgáltatás lényege, hogy a szolgáltatás biztosítása független a szolgáltatást igénybe vevő alkalmazástól (Turner és mások, 2003). A szolgáltatók specializált szolgáltatásokat fejleszthetnek ki annak érdekében, hogy azokat felajánlják a különböző szervezetektől érkező különféle felhasználóknak. Az alkalmazásokat különböző szolgáltatók szolgáltatásainak vagy valamilyen szabványos programozási nyelv, vagy egy olyan speciális szolgáltatás-összeállító nyelv segítségével végzett összekapcsolásával készíthetjük el, mint amilyen a BPEL4WS.

Különféle szolgáltatásmodellek léteznek, a JINI-től (Kumaran, 2001) kezdve a webszolgáltatásokon (Stal, 2002) át egészen a hálózati szolgáltatásokig (Foster

és mások, 2002). Fogalmilag ezek mindegyike a 12.17. ábrán látható modellnek megfelelően működik, amely a Kreger (Kreger, 2001) által leírt fogalmi webszolgáltatás-modell általánosítása. A szolgáltató interfészük megadásával és funkcionalitásuk implementálásával szolgáltatásokat biztosít. A szolgáltatás igénybe vevője az adott szolgáltatást hozzárendeli saját alkalmazásához. Ez azt jelenti, hogy a kérelmező alkalmazása tartalmazza a szolgáltatás hívásának kódját, valamint feldolgozza a hívás eredményét. Annak biztosítása érdekében, hogy a szolgáltatáshoz külső felhasználók is hozzáférhessenek, a szolgáltató bejegyzi a szolgáltatást egy szolgáltatásregiszterbe, amely azt az információt tartalmazza, hogy a szolgáltatás hogyan érhető el és mit csinál.



12.17. ábra. Egy szolgáltatásorientált rendszer fogalmi architektúrája

A szolgáltatásmodell és az osztott objektumok használata között az alábbi különbségek vannak:

- A szolgáltatásokat bármelyik szolgáltató felkínálhatja, a szervezeten kívül vagy belül. Feltételezve, hogy ez bizonyos (később tárgyalt) szabványoknak megfelel, a szervezetek több szolgáltató szolgáltatásainak integrálásával készíthetik el alkalmazásaikat. Egy gyártással foglalkozó cég például közvetlenül kapcsolódhat a beszállítói által nyújtott szolgáltatásokhoz.
- A szolgáltató a szolgáltatásról szóló információkat publikussá teszi, így minden arra feljogosított felhasználó használhatja azokat. A szolgáltatónak és a felhasználónak nem kell az alkalmazásba történő integráció előtt arról tárgyalniuk, hogy az adott szolgáltatás mit is csinál.
- Az alkalmazások a szolgáltatások hozzárendelését egészen a telepítésig vagy akár a végrehajtásig elhalaszthatják, így például egy részvényár-szolgáltatás a rendszer végrehajtása során dinamikusan szolgáltatót válthat.
- Lehetséges új szolgáltatásokat pillanatnyi szükségletek alapján létrehozni. Egy szolgáltató létező szolgáltatások innovatív módon történő összekapcsolásával új szolgáltatásokat állíthat elő.
- A felhasználók a szolgáltatások igénybevételeért, nem pedig azok biztosításáért fizethetnek. Így egy ritkán használt, drága komponens megvásárlása helyett az alkalmazás készítője olyan külső szolgáltatást használhat, amelyért csak akkor kell fizetni, ha azt használják.

- Az alkalmazások kisebbek lehetnek (ami különösen akkor fontos, ha más eszközökbe kerülnek beágyazásra), mivel a kivételkezelést külső szolgáltatásként valósíthatják meg.
- Az alkalmazások reaktív módon tudják működésüket hozzáigazítani a környezetükhöz, mivel annak megváltozása esetén különböző szolgáltatásokkal tudják magukat összerendelni.

Ezen sorok írása idején ezek az előnyök óriási érdeklődést váltottak ki a lazán kapcsolódó osztott alkalmazások készítésének alapjául szolgáló webszolgáltatások iránt. Egyelőre azonban a szolgáltatásorientált architektúrákkal kapcsolatban kevés gyakorlati tapasztalat áll rendelkezésünkre, így még nem tudható, hogy ez a megközelítés milyen gyakorlati következményekkel jár.

A szoftverek újrafelhasználhatósága hosszú ideje kutatások témájául szolgál, de – ahogyan azt a 18. és 19. fejezetben tárgyalom – az újrafelhasználás során még mindig sok gyakorlati problémával találjuk szembe magunkat. A fő problémák egyike az, hogy az újrafelhasználható komponensek szabványait csak az utóbbi időben alakították ki, és több ilyen szabvány is használatban van. Azonban a webszolgáltatások kialakulását szabványok vezérelték, és a webszolgáltatások sok aspektusát lefedő szabványok most vannak készülóban. Az alábbi három alapvető szabvány teszi lehetővé a webszolgáltatások közötti kommunikációt:

1. *SOAP (Simple Object Access Protocol – egyszerű objektumelérési protokoll)*. Ez a protokoll a webszolgáltatások közötti strukturált adatcsere módját határozza meg.
2. *WSDL (Web Services Description Language – webszolgáltatás-leíró nyelv)*. Azt határozza meg, hogy hogyan kell a webszolgáltatások interfészeit leírni.
3. *UDDI (Universal Description, Discovery and Integration – univerzális leírás, felderítés és integráció)*. Ez egy olyan felderítési szabvány, amely megadja, hogy hogyan lehet a szolgáltatás kérelmezői által a szolgáltatások felderítésére használt szolgáltatásleíró információkat szervezni.

Ezen szabványok mindegyike az XML-en – egy ember és gép által is érthető jelölőnyelven (Skonnard és Gudgin, 2002) – alapszik. A webszolgáltatások fogalmának megértéséhez azonban nincs szükség ezen szabványok részleteinek ismeretére.

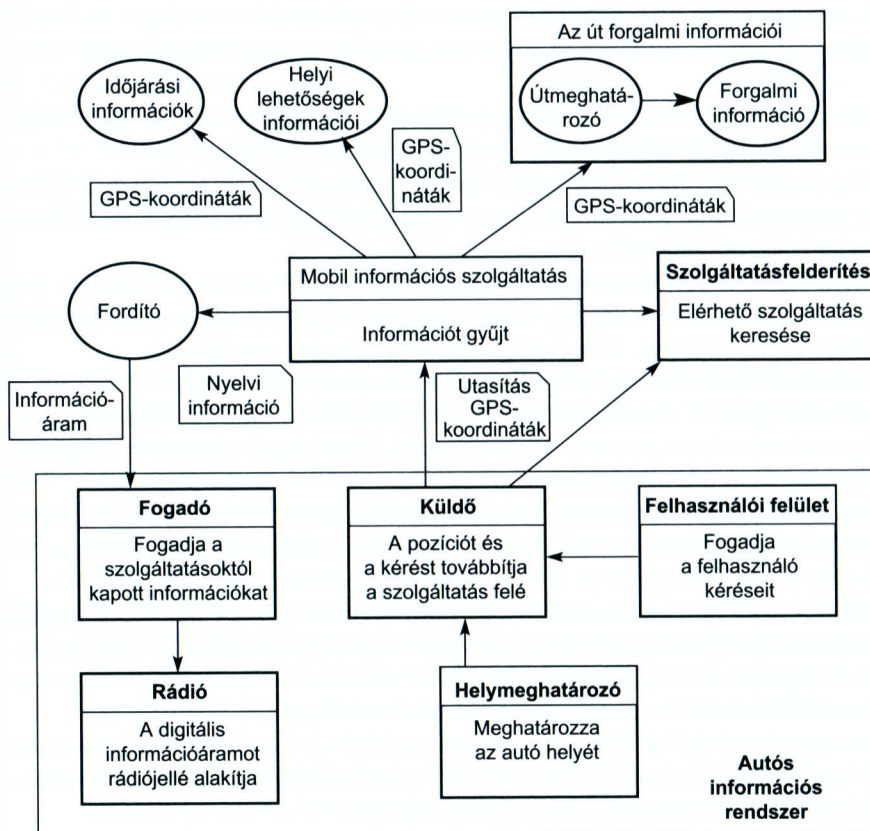
A webszolgáltatás-architektúrák olyan lazán kapcsolódó architektúrák, amelyekben a szolgáltatás-hozzárendelés a végrehajtás során változhat. Bizonyos rendszereket kizárólag webszolgáltatások felhasználásával készítenek, míg más rendszerek ötvözik a webszolgáltatásokat a helyileg kifejlesztett komponensekkel. Annak bemutatása céljából, hogy hogyan lehet ilyen alkalmazásokat felépíteni, tekintsük a következő forgatókönyvet:

Egy autós információs rendszer időjárési, forgalmi, helyi stb. információkat biztosít a sofőröknek. A rendszer kapcsolatban van az autórádióval, így ezeket az információkat egy speciális csatornán leadott jelként juttatja el. Az autó fel van szerelve egy GPS-

vevővel, amelynek segítségével felderíthető a pozíciója, és e pozíció alapján a rendszer számos információs szolgáltatást el tud érni. Az információközlés a sofőr által megadott nyelven történik.

A 12.18. ábrán egy ilyen rendszer lehetséges felépítése látható. A szoftver öt modulból áll. Ezek egy része a sofőrrel, az autó helyzetét jelentő GPS-vevővel és az autórádióval való kommunikáció kezelését végzi, míg a Küldő és Fogadó modulok a külső szolgáltatásokkal tartják a kapcsolatot.

Az autó egy külső mobil információs szolgáltatással kommunikál, amely több más, például az időjárás és forgalmi helyzetről, valamint a helyi lehetőségekről információt nyújtó szolgáltatás által biztosított információkat gyűjti össze. Különböző helyeken különböző szolgáltatók nyújtják ezt a szolgáltatást, az autó rendszere pedig egy felderítő szolgáltatás segítségével határozza meg a megfelelő szolgáltatás helyét, és végzi el az összerendelést. A mobil információs rendszer szintén ezt a felderítő szolgáltatást használja, amikor a megfelelő időjárás, forgalmi és helyi lehetőségeket biztosító szolgáltatás összerendelését végzi. A szolgáltatások egymással SOAP-üzeneteket váltanak, amelyek tartalmazzák a GPS-



12.18. ábra. Egy szolgáltatásalapú autós információs rendszer

pozíciót, amit a szolgáltatások a megfelelő információ kiválasztására használnak. Az összegyűjtött információ az információ nyelvét a sofőr nyelvére fordító szolgáltatáson keresztül jut el az autóhoz.

Ez a példa a szolgáltatásorientált megközelítés egyik fő előnyét mutatja be. Nem szükséges a rendszer programozásakor, sőt telepítésekor eldöntenünk, hogy mely szolgáltatót, és annak melyik konkrét szolgáltatását vesszük igénybe. Ahogy az autó változtatja a helyét, az autó szoftvere szolgáltatásfelderítés segítségével megkeresi a leginkább megfelelő szolgáltatást, és elvégzi az összerendelést. A fordítást végző szolgáltatás miatt a határokon átlépve azok az emberek is hozzáférnek a helyi információkhoz, akik nem beszélik a helyi nyelvet.

A szolgáltatásorientált feldolgozás víziója a jelenlegi webszolgáltatásokkal még nem megvalósítható, mert ezekben a szolgáltatások alkalmazásokhoz rendelése még meglehetősen statikusan zajlik. A jövőben azonban egyre több dinamikus összerendeléssel és alkalmazásarchitektúrával találkozunk majd, és a dinamikus, szolgáltatásorientált rendszerek víziója realizálódik. A téma fontossága miatt erről a 31. fejezetben, a kialakuló technológiákról szóló új fejezetben bővebben írok.

Kulcsfogalmak

- Az osztott rendszerek támogatják az erőforrásmegosztást, a nyíltságot, a konkurenciát, a skálázhatóságot, a hibatűrést és az átlátszóságot.
- A kliens-szerver rendszerek olyan osztott rendszerek, amelyekben a rendszert szerverek által kliensfolyamatok számára biztosított szolgáltatások összességéként modellezzük.
- A kliens-szerver rendszerekben a felhasználói felület mindig a kliensen fut, míg az adatkezelést mindig egy osztott szerver végzi. Az alkalmazás funkcionalitása akár a kliensen, akár pedig a szerveren implementálható.
- Egy osztott objektumarchitektúrában nem teszünk különbséget a kliensek és a szerverek között. Az objektumok a többi objektum által hívható általános szolgáltatásokat biztosítanak. Ez a megközelítés kliens-szerver rendszerek implementálására is alkalmazható.
- Az osztott objektumrendszerek az objektumok kommunikációjának kezelése, valamint az objektumok rendszerhez történő hozzáadásának és rendszerből való eltávolításának lehetővé tétele céljából köztes termékeket igényelnek.
- A CORBA-szabványok az osztott objektumarchitektúrákat támogató köztes termékek szabványai, amelyek objektummodell-definíciókat, egy ORB definícióit és a gyakori szolgáltatások definícióit tartalmazzák. A CORBA-szabványoknak különböző implementációi állnak rendelkezésre.
- A peer-to-peer architektúrák olyan decentralizált architektúrák, amelyekben nem teszünk különbséget kliensek és szerverek között. A számításokat különböző szervezetek rendszerei között oszthatjuk el.

- A szolgáltatásorientált rendszerek a különféle szolgáltatók által biztosított szoftverszolgáltatások összekapcsolásával jönnek létre. A szolgáltatásorientált architektúrák fontos jellemzője, hogy a szolgáltatások architekturális komponensekhez történő hozzárendelése elhalasztható a rendszer telepítéséig vagy futtatásáig.

További irodalom

„Turning software into a service”. Ez a cikk jó áttekintést nyújt a szolgáltatásorientált feldolgozás alapelveiről. A témában született sok egyéb cikkel szemben nem rejti el ezeket az alapelveket a kapcsolódó szabványok tárgyalása mögé. (Turner, M. és mások, *IEEE Computer*, 36(10), October, 2003.)

Peer-to-peer: Harnessing the Power of Disruptive Technologies. Habár nem az egész könyv szól a p2p architektúrákról, remek bevezetőként szolgál a p2p feldolgozásba, és számos p2p rendszer felépítését is tárgyalja. (Oram, A. (ed.), 2001, O'Reilly and Associates, Inc.)

Distributed systems: Concepts and Design, 3rd ed. Az elosztott rendszerek tervezésének és implementálásának minden aspektusát tárgyaló, átfogó könyv. Az első két fejezet különösen az itt leírtak témájába vág. (Coulouris, G. és mások, 2001, Addison-Wesley.)

„Middleware: A model for distributed systems services”. Nagyszerű áttekintő cikk, összegzi a köztes termékek osztott rendszerekben betöltött szerepét, és kifejti a nyújtható köztes termék szolgáltatások körét. (Bernstein, P. A., *Comm. ACM*, 39(2), February 1996.)

Feladatok

- 12.1. Magyarázza el, hogy az osztott rendszerek természetüknél fogva miért skálázhatóbbak, mint a központosított rendszerek. Mik egy rendszer skálázhatóságának valószínű határai?
- 12.2. Mik az alapvető különbségek a kliens-szerver rendszerek fejlesztésének vékony, illetve vastag kliens elvű megközelítései között? Magyarázza el, hogy miért mosódik el ez a különbség abban az esetben, ha implementációs nyelvként a Javát választjuk.
- 12.3. Az ön ügyfele egy részvényinformációkat kezelő rendszer kifejlesztésében érdekelt, amelyben a kereskedők cégekre vonatkozó információhoz férhetnek hozzá, különféle befektetési forgatókönyveket értékelhetnek ki egy szimulációs rendszer használatával. Az egyes kereskedők ezt a szimulációt különbözőféleképpen használják attól függően, hogy mekkora tapasztalattal rendelkeznek, illetve hogy milyen típusú részvényekkel kereskednek. Javasoljon a rendszernek egy olyan kliens-szerver architektúrát, amely megmutatja, hogy a funkcionalitás hol helyezkedik el. Választását indokolja is meg.

- 12.4. A 12.4. ábrán látható alkalmazásmodellre hivatkozva bocsátkozzon fejtegetésekbe egy 1980-as évekbeli, nagyszámítógépen működő, biztosítási kötvényeket feldolgozó őrsrendszer kliens-szerver architektúrájává történő átalakításakor esetlegesen felbukkanó problémákról.
- 12.5. Melyek az ORB által kötelezően biztosítandó alapeszközök?
- 12.6. Magyarázza meg, hogy az osztott objektumok ORB-vel való használata miért egyszerűsíti le a skálázható kliens-szerver rendszerek implementációját. Válaszát szemléltesse példával.
- 12.7. Hogyan segíti a CORBA IDL a különböző programozási nyelveken implementált objektumok közötti kommunikációt? Magyarázza el, hogy ez a megközelítési mód miért okozhat teljesítménybeli problémákat akkor, ha az objektumok implementációs nyelvei között gyökeres eltérés van.
- 12.8. Tegyen javaslatot egy olyan színházjegyfoglaló rendszer osztott objektumarchitektúrájára, ahol a felhasználók ellenőrizhetik a színházak egyes székeinek foglaltságát, és foglalást is végezhetnek. A rendszernek kezelnie kell a visszaváltott jegyeket is, amelyeket az utolsó pillanatban megpróbálnak újra értékesíteni.
- 12.9. Sorolja fel a decentralizált és a félig centralizált peer-to-peer architektúrák 2-2 előnyét és hátrányát.
- 12.10. Milyen előnyei vannak a dinamikus összerendelésnek egy szolgáltatásorientált rendszer esetében?
- 12.11. Az autós információs rendszerhez kapcsolódóan magyarázza el, hogy miért az a legjobb megoldás, hogy az autós szoftver, ahelyett hogy közvetlenül az információs szolgáltatással létesítene kapcsolatot, egy információgyűjtő szolgáltatással kommunikál. A kérdés megválaszolása során gondoljon a kommunikáció megbízhatóságára is.
- 12.12. A szolgáltatásorientált modell fejlődése szabványok korai specifikációin és alkalmazásain alapul. Beszéljen a szabványosításnak a verseny segítésének és korlátozásának, valamint a szoftverpiaci innovációban betöltött általános szerepéről.