Elosztott rendszerek: Alapelvek és paradigmák
Distributed Systems: Principles and Paradigms

Maarten van Steen[1] Kitlei Róbert [2]

[1]VU Amsterdam, Dept. Computer Science
[2]ELTE Informatikai Kar

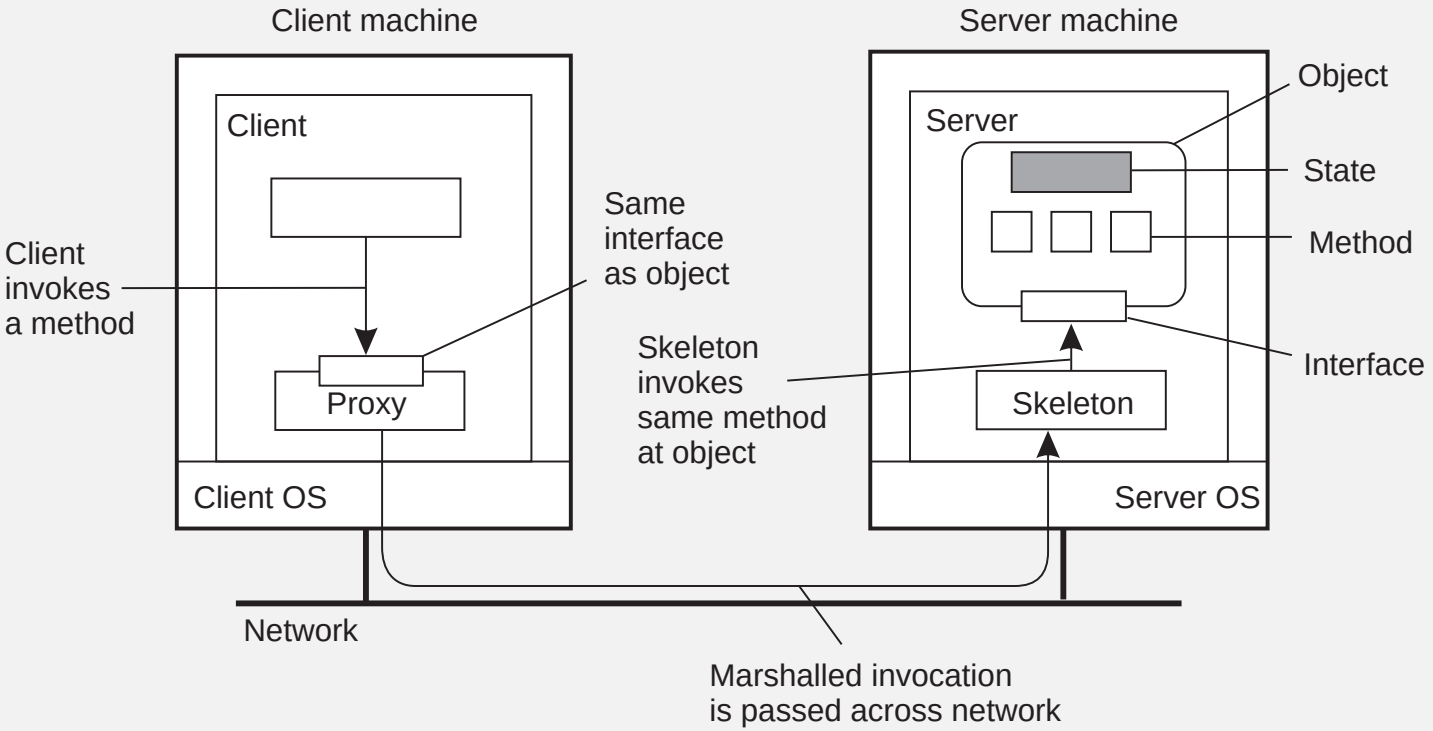
10. rész: Objektumalapú elosztott rendszerek

2015. május 24.

Tartalomjegyzék

Távoli elosztott objektumok

- Objektum: műveleteket és adatokat zár egységbe (enkapszuláció)
- A műveleteket metódusok implementálják, ezeket interfészekbe csoportosítjuk
- Az objektumokat csak az interfészükön keresztül érhetik el a kliensek
- Az objektumokat objektumszerverek tárolják
- A kliensoldali helyettes (proxy) megvalósítja az interfészt
- A szerveroldalon a váz (skeleton) kezeli a beérkező kéréseket



Távoli elosztott objektumok

Objektumok létrehozás ideje alapján

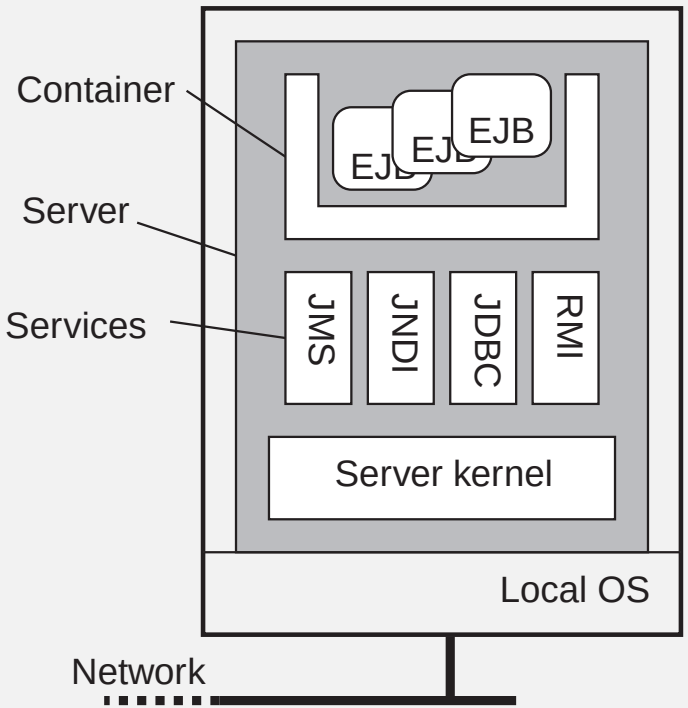
- Fordítási időben létrejövő objektumok: A helyettest és a vázat a fordítóprogram készíti el, összeszerkeszti a kliens és a szerver kódjával. Nem cserélhető le, miután létrejött, és a klienssel/szerverrel azonos a programozási nyelve.
- Futási időben létrejövő objektumok: Tetszőleges nyelven valósítható meg, de objektumadapterre van szükség a szerveroldalon a használatához.

Objektumok élettartamuk alapján

- Átmeneti (tranziens) objektum: Élettartama csak addig tart, amíg be van töltve a szerverbe. Ha a szerver kilép, az objektum is megsemmisül.
- Tartós (perzisztens) objektum: Az objektum állapotát és kódját lemezre írjuk, így a szerver kilépése után is megmarad. Ha a szerver nem fut, az objektum passzív; amikor a szerver elindul, betöltéssel aktivizálható.

Példa: Enterprise Java Beans (EJB)

Az objektumokat alkalmazásszerverek (pl. GlassFish) tárolják, amelyek lehetővé teszik az objektumok különböző módokon való elérését.



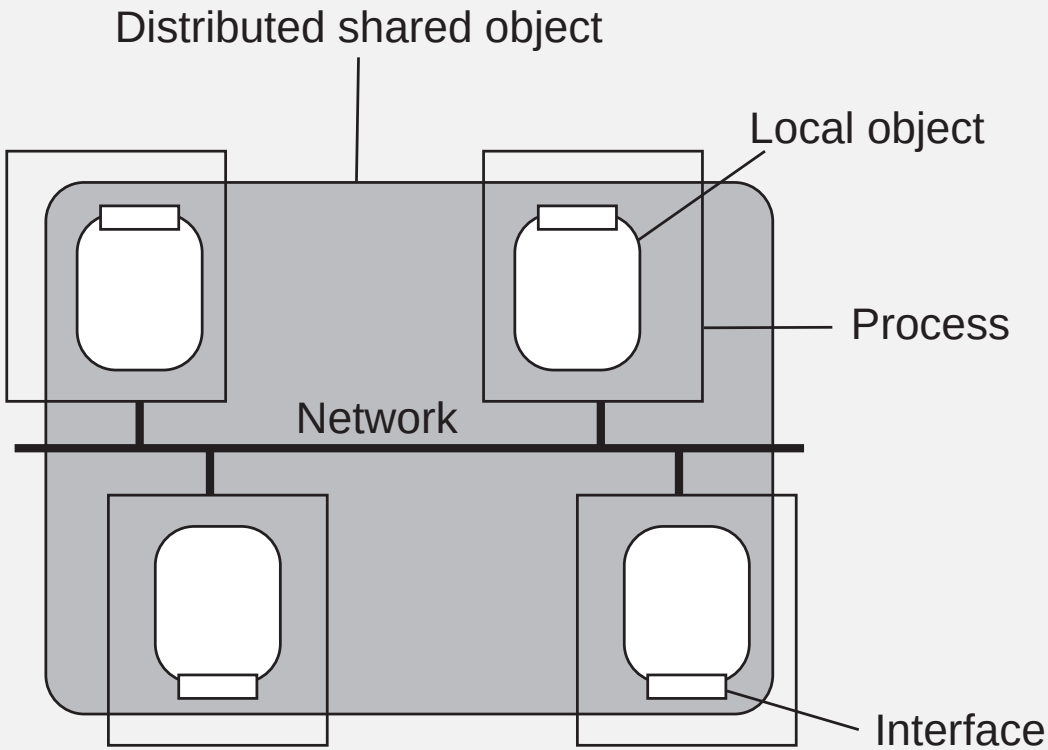
Példa: Enterprise Java Beans (EJB)

EJB-k fajtái

- Stateless session bean: Tranziens objektum, egyszer hívják meg, miután elvégezte a feladatát, megszűnik. Példa: egy SQL lekérdezés végrehajtása, és az eredmény átadása a kliensnek.
- Stateful session bean: Tranziens objektum, de a klienssel egy munkameneten (session) keresztül tartja a kapcsolatot, ezalatt állapotot is tart fenn. Példa: bevásárlókosár.
- Entity bean: Perzisztens, állapottal rendelkező objektum, amely több munkamenetet is ki tud szolgálni. Példa: olyan objektum, amely az utolsó néhány kapcsolódó kliensről tárol adatokat.
- Message-driven bean: Különböző fajta üzenetekre reagálni képes objektum. A publish/subscribe kommunikációs modell szerint működik.

Globe elosztott objektumok

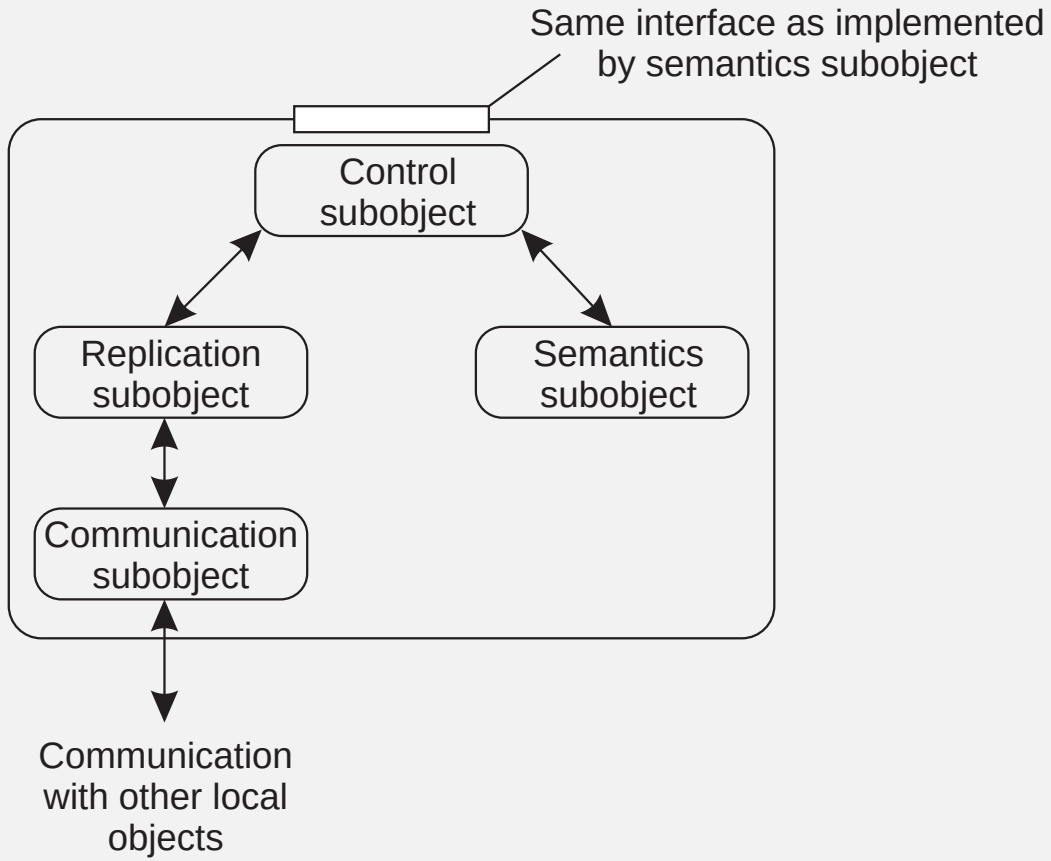
Általában a távoli objektumok nem elosztottak: az állapotukat egy gép tárolja.
A Globe rendszerben az objektumok fizikailag több gépen helyezkednek el: elosztott közös objektum (distributed shared object, DSO).



Globe elosztott objektumok

Az elosztottság támogatásához belső architektúra szükséges, és célszerű, ha ez független attól, hogy a DSO külső felülete milyen szolgáltatásokat nyújt.

A replikációkezelő alobjektum vezérli, hogy hogyan és mikor kell a lokális szemantikus alobjektumot meghívni.



Folyamatok: Objektumszerverek

A rendszer részei a kiszolgálók, a vázak és az adapterek.
A kiszolgálót (servant), amely az objektum működését biztosítja, több paradigma szerint lehet implementálni:

- Függvények gyűjteménye, amelyek adatbázistáblákat, rekordokat stb. manipulálnak (pl. C vagy COBOL nyelven)
- Osztályok (pl. Java vagy C++ nyelven)

A váz (skeleton) a szerveroldali hálózati kapcsolatokat kezeli:

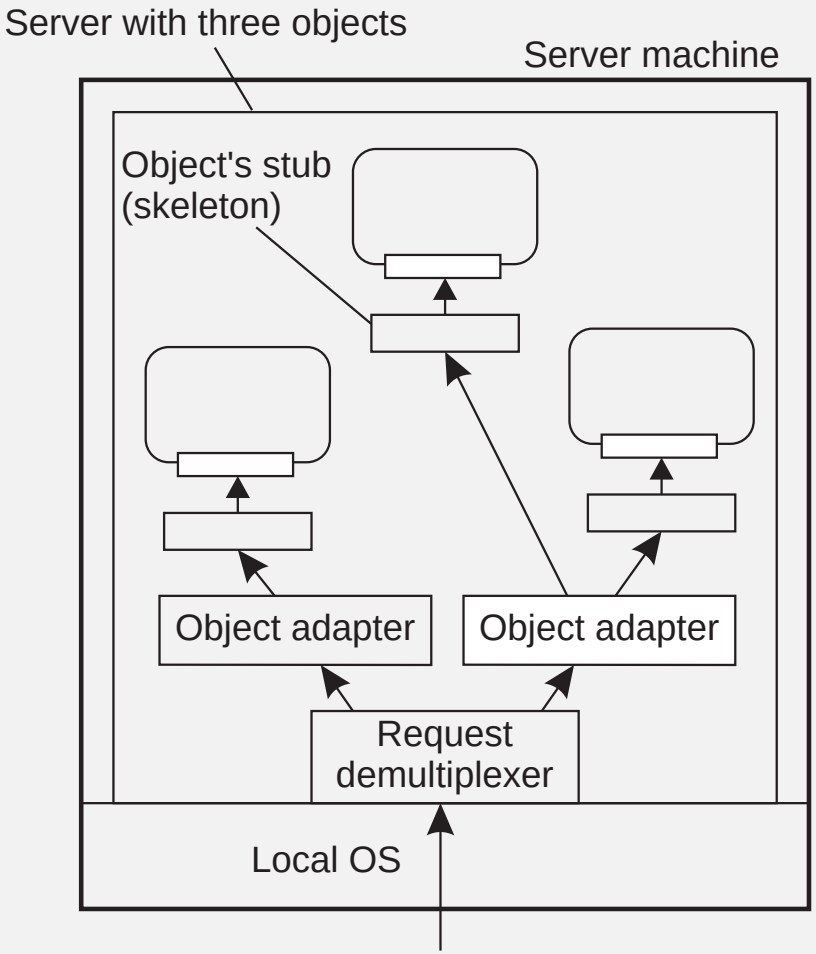
- Kicsomagolja a beérkező kéréseket, lokálisan meghívja az objektumot, becsomagolja és visszaküldi a választ
- Az interfész specifikációja alapján hozzák létre

Az objektumadapter feladata objektumok egy csoportjának kezelése:

- Elsőként fogadja a kéréseket, és azonosítja a releváns kiszolgálót
- Aktivációs házirend (policy) szerint aktiválja a megfelelő vázat
- Az adapter generálja az objektumhivatkozásokat

Folyamatok: Objektumszerverek

Az objektumszerverek vezérlik a tartalmazott objektumok létrehozását.



Példa: Ice

Az Ice (Internet Communications Engine) objektumorientált köztesréteg, szolgáltatásai elérhetőek a legnépszerűbb nyelveken. Az aktivációs házirend megjelenik a köztesréteg szintjén: az adapter `properties` adattagján át lehet módosítani. Ez segíti a rendszer egyszerű kezelhetőségét.

```
main(int argc, char* argv[]) {
  Ice::Communicator ic = Ice::initialize(argc, argv);
  Ice::ObjectAdapter adapter =
   ic->createObjectAdapterWithEndPoints("a","tcp -p 2000");
  Ice::Object object = new MyObject;

  adapter->add(object, objectID);
  adapter->activate();

  ic->waitForShutdown();
}
```

Kliens csatlakoztatása objektumhoz

Objektumhivatkozás

Ha egy kliens birtokol egy referenciát egy objektumra, képes hozzá csatlakozni (bind):

- A hivatkozás előírja, melyik szerveren, melyik objektumot, milyen kommunikációs protokoll szerint lehet rajta keresztül elérni
- A hivatkozáshoz kód tartozik, ezt a konkrét objektum eléréséhez felparaméterezve kapjuk a helyettest

Kétfajta csatlakozás

- Implicit: Magán a hivatkozott objektumon hívjuk meg a műveleteket
- Explicit: A kliens kódjában a csatlakozás explicit megjelenik

Kliens-objektum csatlakozás: implicit/explicit

```
// implicit
Distr_object* obj_ref;
obj_ref = ...;
obj_ref->do_something();
```

```
// explicit
Distr_object* obj_ref;
Local_object* obj_ptr;
obj_ref = ...;
obj_ptr = bind(obj_ref);
obj_ptr->do_something();
```

- A hivatkozás tartalmazhat egy URL-t, ahonnan az implementáció letölthető
- Protokollban rögzíthető, hogyan kell betölteni és példányosítani a letöltött kódot
- A szerver és az objektum ismerete elegendő lehet a távoli metódushívás megkezdéséhez
- Objektumhivatkozások paraméterként is átadhatóak, amik RPC esetében bonyodalmakat okoznak

Távoli metódushívás (Remote Method Invocation, RMI)

Tegyük fel, hogy a helyettes és a váz rendelkezésre áll a kliensnél/szervernél.

1. A kliens meghívja a helyettest
2. A helyettes becsomagolja a hívás adatait, és elküldi a szervernek
3. A szerver biztosítja, hogy a hivatkozott objektum aktív:
   - Külön folyamatot hozhat létre, amely tárolja az objektumot
   - Betöltheti az objektumot a szerverfolyamatba
   - ...
4. Az objektum váza kicsomagolja a kérést, és a metódus meghívódik
5. Ha paraméterként objektumhivatkozást kaptunk, ezt szintén távoli metódushívással éri el a szerver; ebben a szerver kliensként vesz részt
6. A választ hasonló úton küldjük vissza, a helyettes kicsomagolja, és visszatér vele a klienshez

RMI: Paraméterátadás

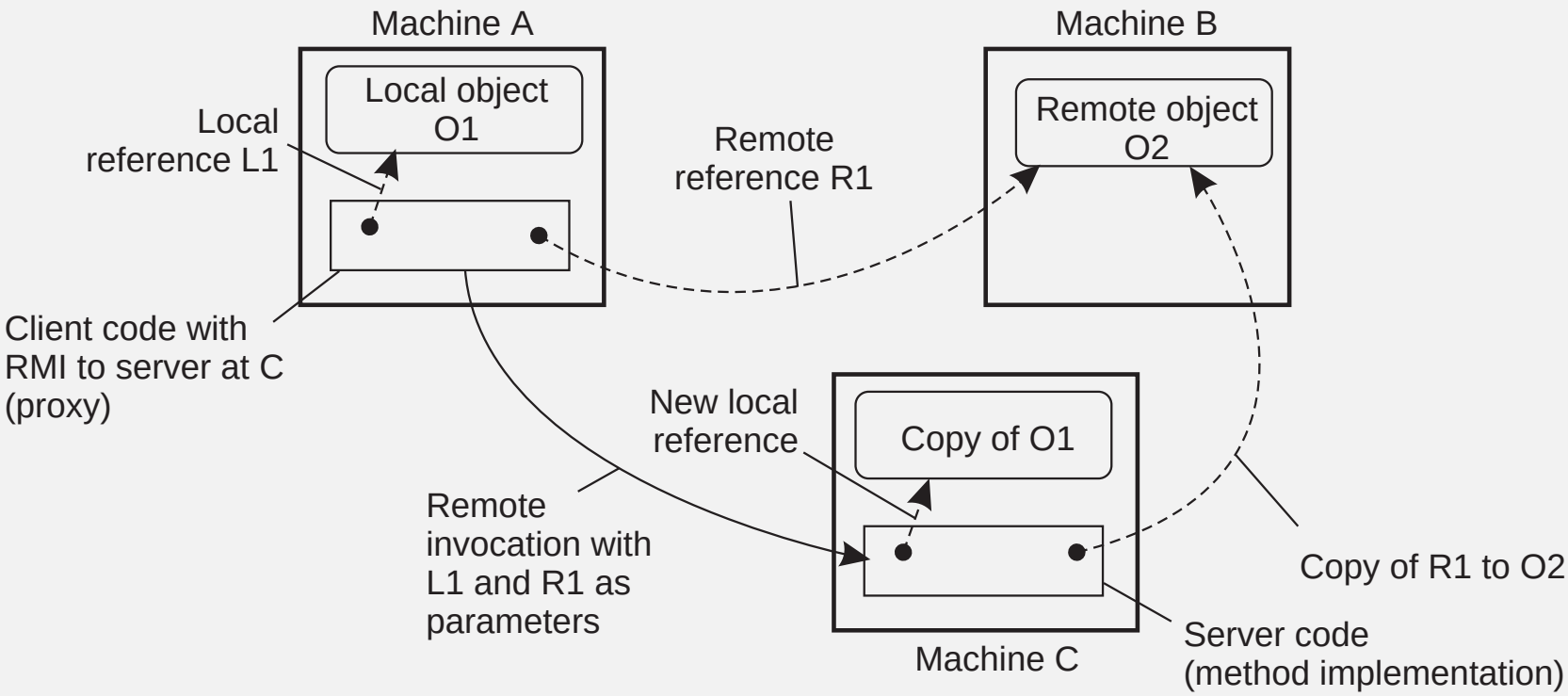
Hivatkozás szerinti paraméterátadás: sokkal egyszerűbb, mint RPC esetén.

- A szerver egyszerűen távoli metódushívással éri el az objektumot
- Ha már nincsen szüksége rá, megszünteti a csatolást (unbind)

Érték szerinti paraméterátadás: RMI esetén ez kevésbé kényelmes.

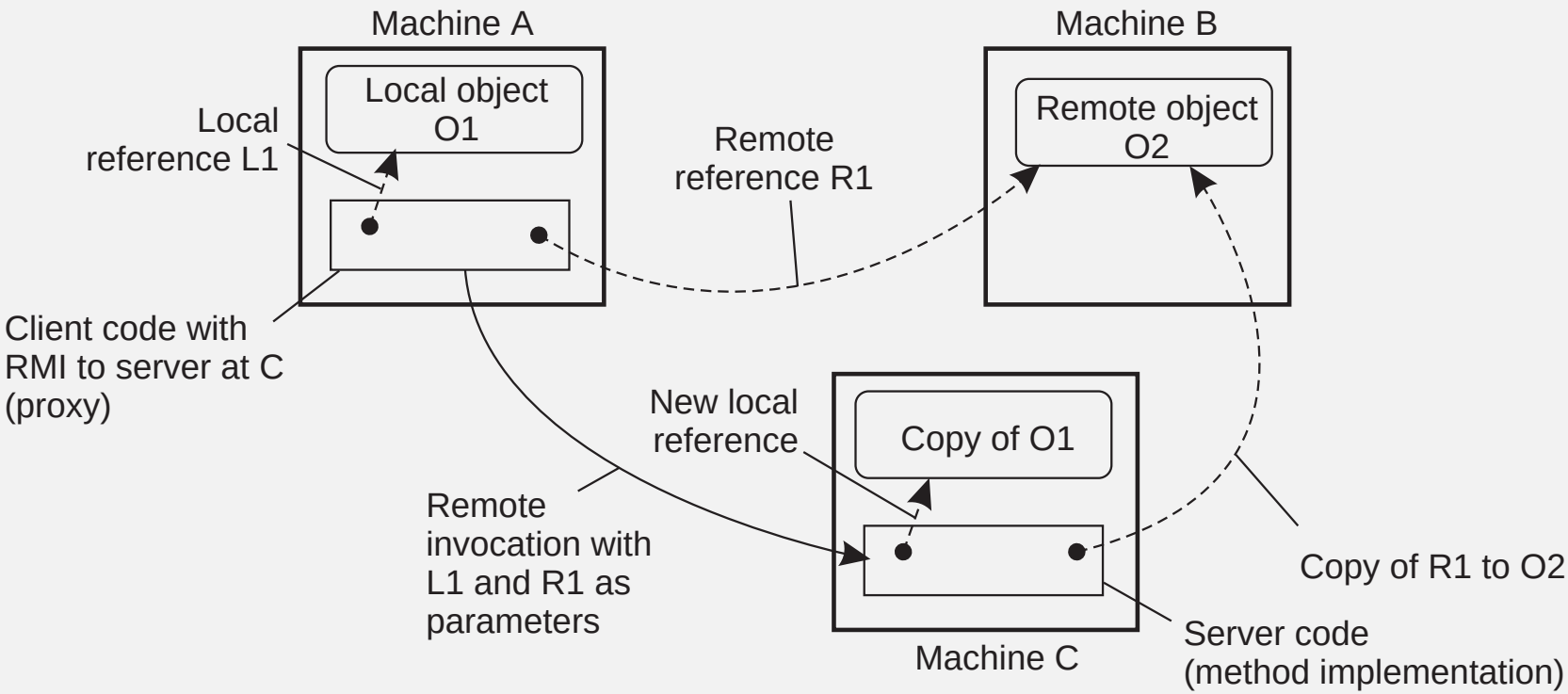
- Szerializálni kell az objektumot
  - Az állapotát
  - A metódusait, vagy hivatkozást olyan helyre, ahol elérhető az implementációjuk
- Amikor a szerver kicsomagolja az objektumot, ezzel másolat készül az eredetiről
- Ez az automatikus másolódás többféle problémát okoz, pl. néha „túl átlátszó”: könnyen több objektumról készíthetünk másolatot, mint amennyiről szeretnénk

RMI: Paraméterátadás



A rendszerszintű objektumhivatkozások általában a szerver címét, adapterének portját és az objektum lokális azonosítóját tartalmazzák. Néha ezekhez további információk is járulnak, pl. a kliens és szerver között használt protokoll (TCP, UDP, SOAP stb.)

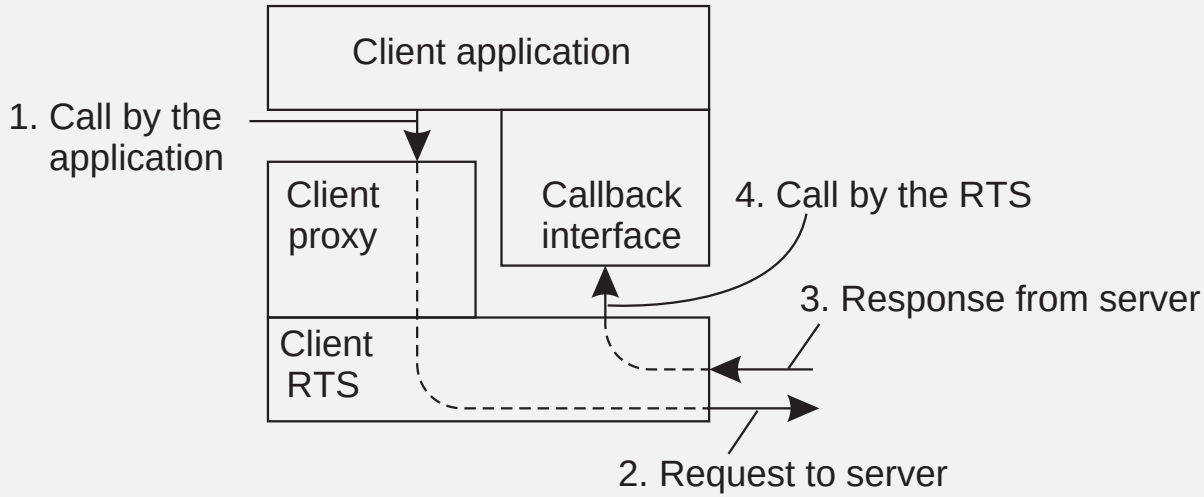
RMI: Paraméterátadás

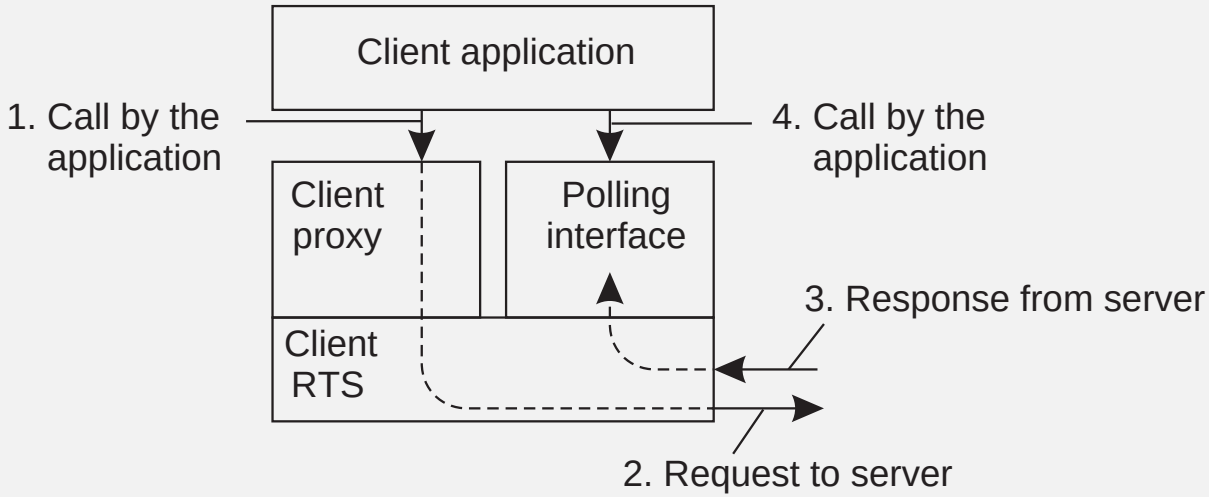


Mivel a helyettesnek mindenféleképpen ismernie kell a hivatkozáshoz szükséges adatokat (cím, port, lokális ID), felhasználhatjuk a helyettest magát mint távoli hivatkozást. Ez különösen előnyös, ha a helyettes letölthető (pl. a Java esetében igen).

Objektumalapú üzenetküldés

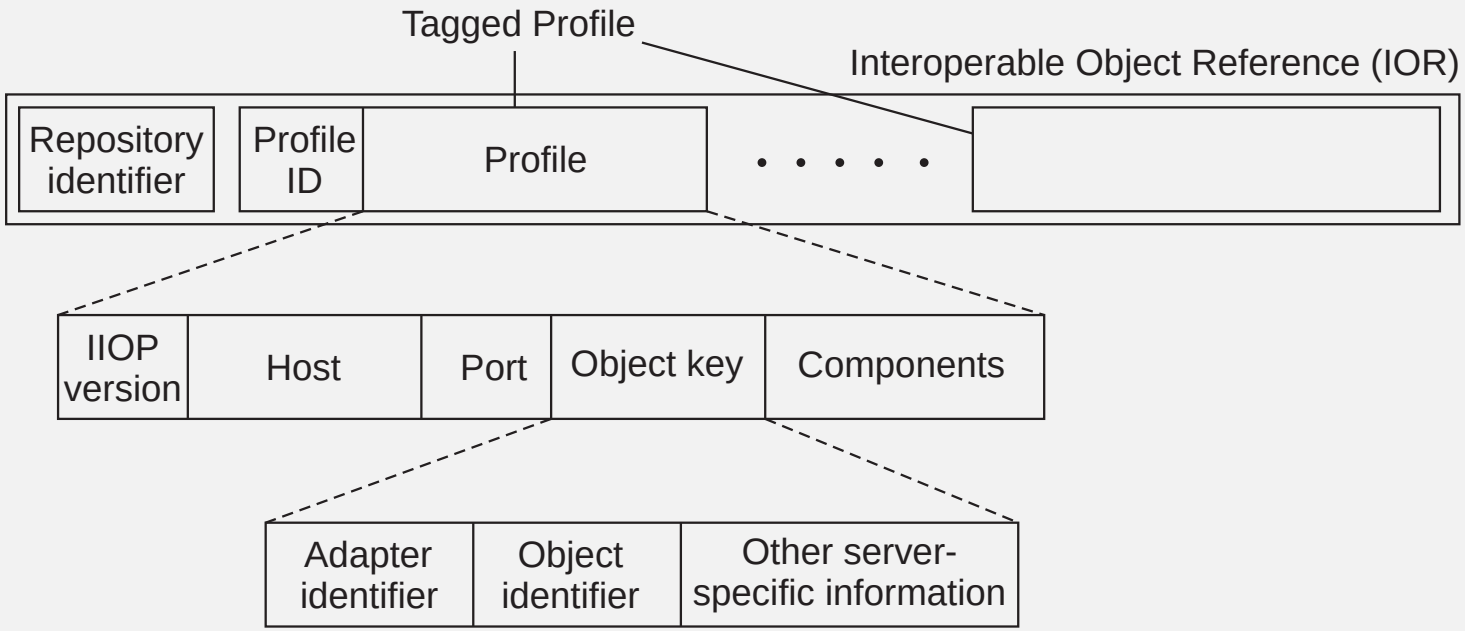
A kliens az üzenetekre várás közben lehet aktív vagy passzív.





Objektumhivatkozások

IIOP[1]: távoli objektumhivatkozásokat kezelő protokoll
CORBA[2]: Objektumalapú köztesréteg, IIOP-t használ
Az alábbi ábrán a CORBA objektumhivatkozásának szerkezete látható.

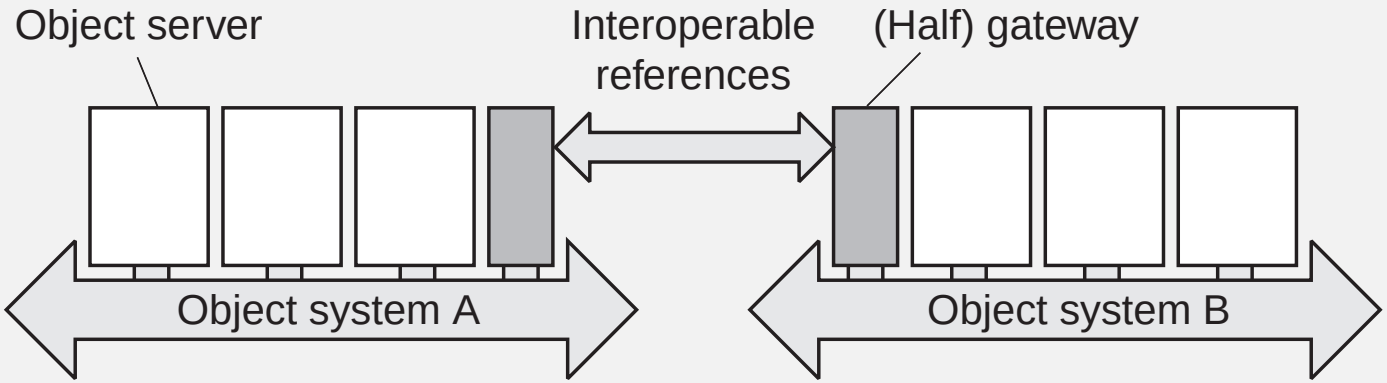


[1] Internet Inter-ORB Protocol
[2] Common Object Request Broker Architecture

Objektumhivatkozások

Különböző objektumkezelő rendszerekben a hivatkozások szerkezete nagymértékben eltérhet.

A rendszerek között átjárók (gateway) biztosíthatják a hivatkozások konvertálását.



Replikáció és konzisztencia

Az objektumok a belépő konzisztencia megvalósításának természetesen adódó eszközei:

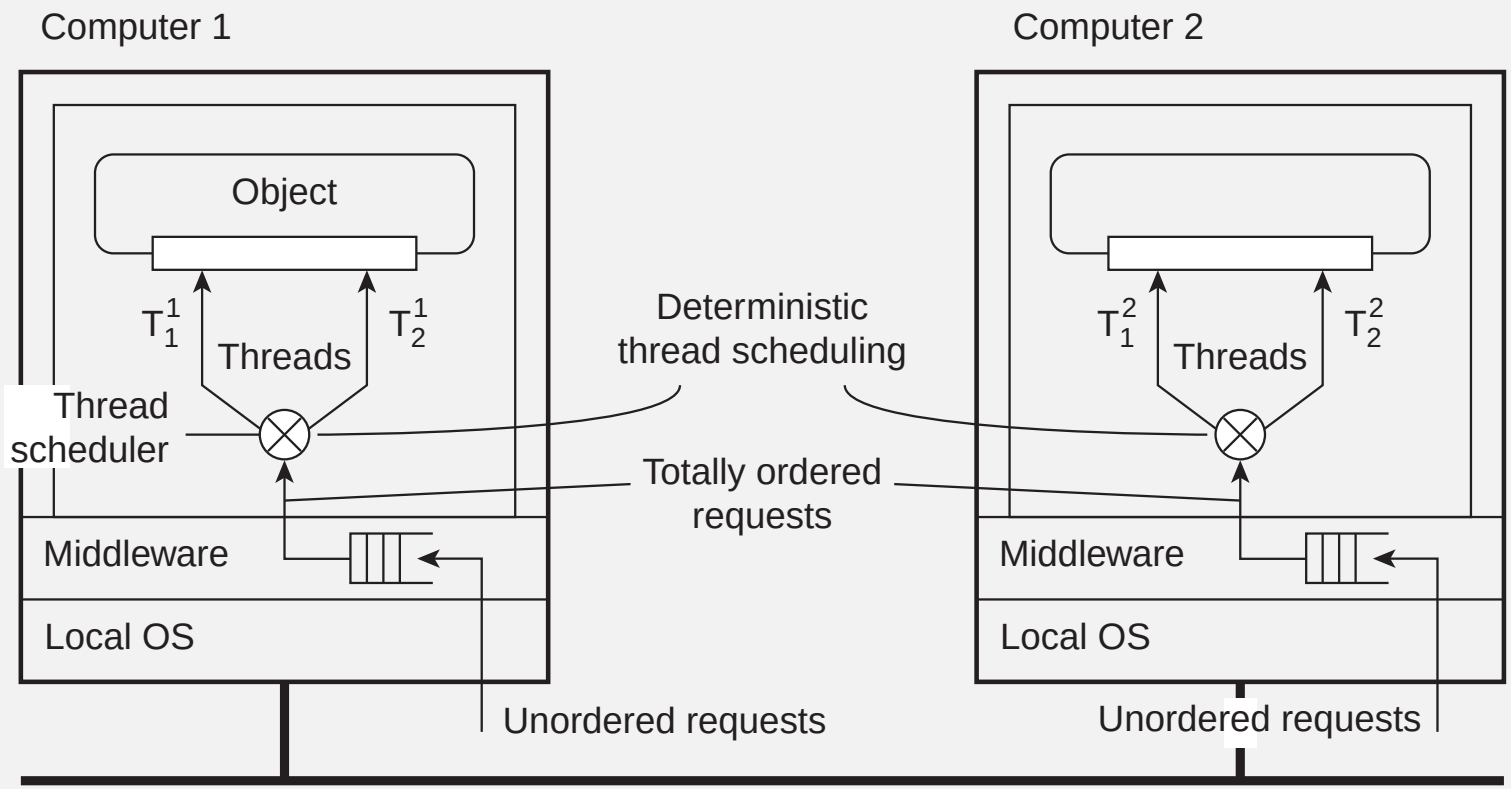
- Az adatok egységbe vannak zárva, és szinkronizációs változóval (zárral) védjük őket
- A szinkronizációs változókat soros konzisztencia szerint érjük el (az értékek beállítása atomi lépés)
- Az adatokat kezelő műveletek összessége pont az objektum interfésze lesz

Replikáció

Mit tegyünk, ha az objektumot replikálni kell? A replikált objektumokon a műveletek végrehajtásának sorrendjének azonosnak kell lennie.

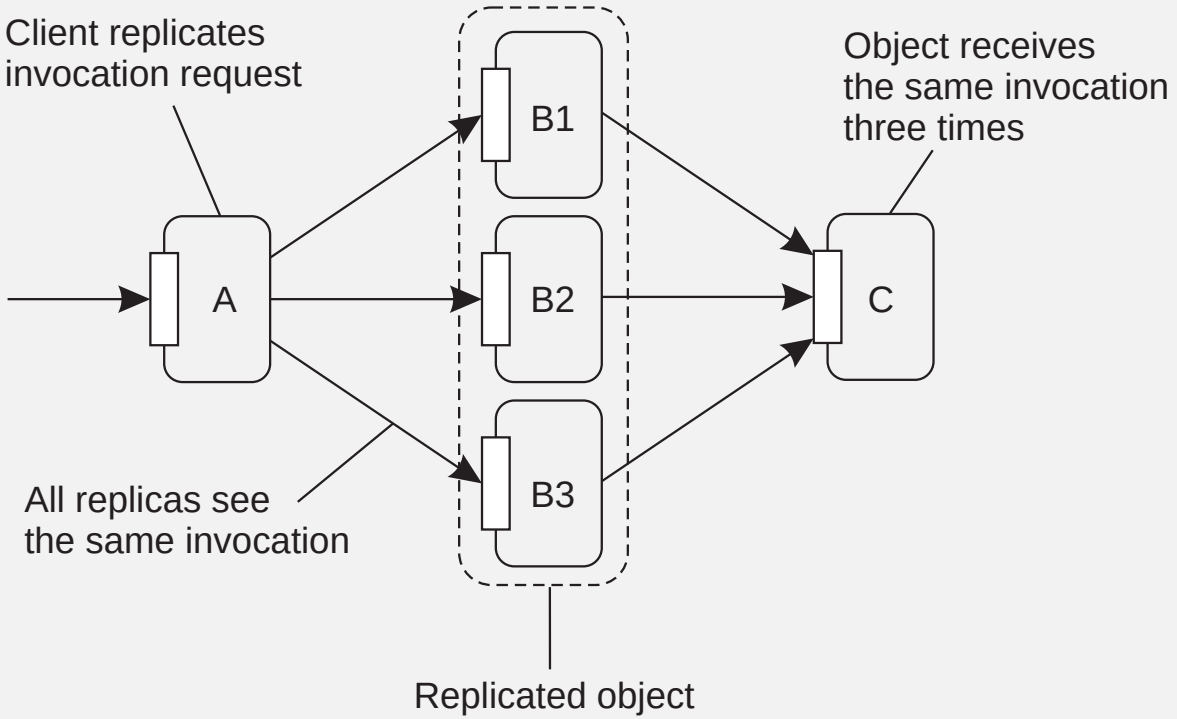
Replikált objektumok

Nemcsak a kéréseknek kell sorrendben beérkezniük a replikátumokhoz; a vonatkozó szálak ütemezésének determinisztikusnak kell lennie.
Egyszerű megoldás lehetne, ha teljesen sorosítva (egyetlen szálon) hajtanánk végre a kéréseket, de ez túl költséges.



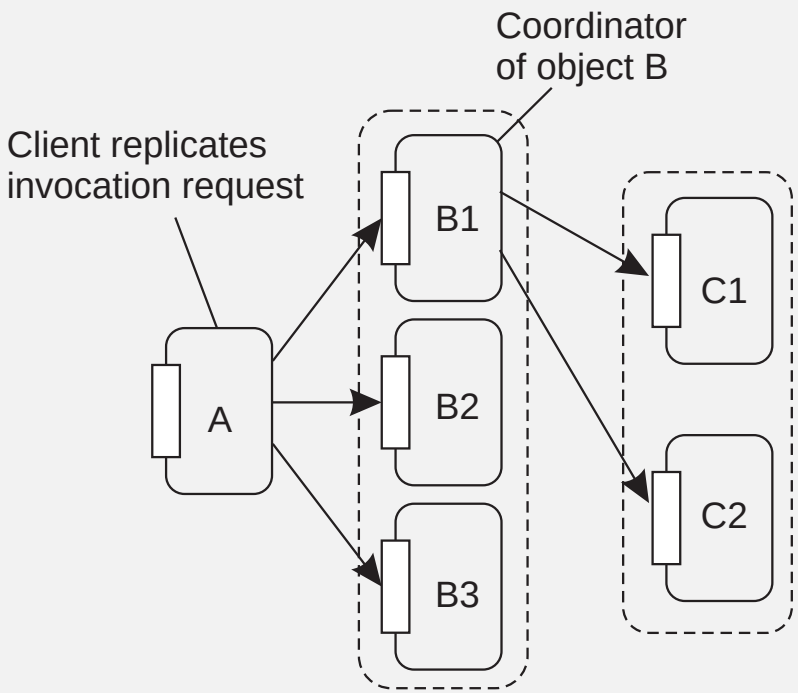
Replikált hívások

Aktív replikáció: ha a replikált objektum hívás során maga is meghív más objektumot, az a kérést többszörözve kapná meg.

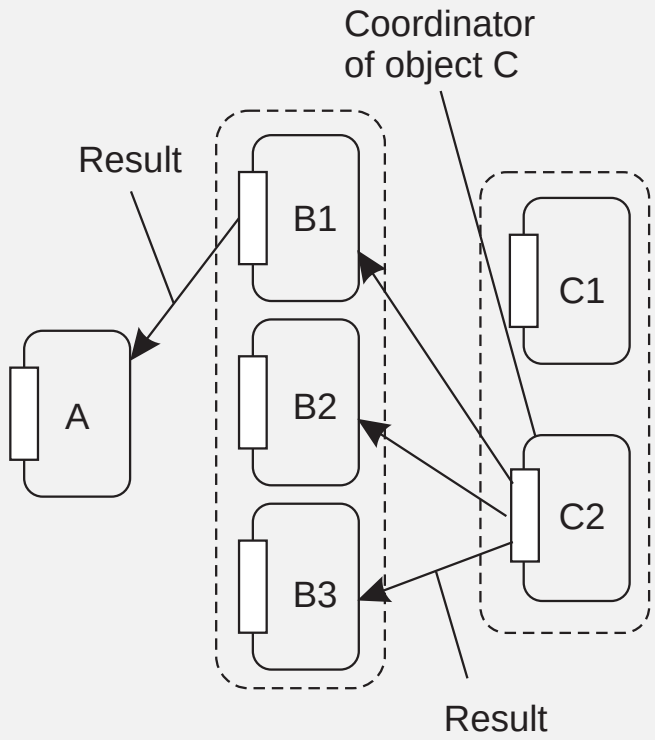


Replikált hívások

Megoldás: mind a szerver-, mind a kliensobjektumon válasszunk koordinátort, és csak a koordinátorok küldhessenek kéréseket és válaszokat.



(a)



(b)