A Jini és a P-Grade rendszerek integrálása klaszteren

PVM program futtatás, mint Jini szolgáltatás

Az alapprobléma

Egy P-Grade -ben megírt, majd lokálisan lefordított, nem interaktív PVM program futtatásához távoli erőforrásokat szeretnénk igénybe venni.

Alapkérdések

- Honnan lehetne megtudni, hogy jelen pillanatban hol áll rendelkezésre futtatásra alkalmas, szabad környezet?
- Hogyan lehetne eljuttatni oda a programot, majd az ottani erőforrásokat használva lefuttatni?
- A konzol kimenet, és az eredmény-fájlok hogyan fognak visszakerülni?

Jini, mint a megoldás eszköze

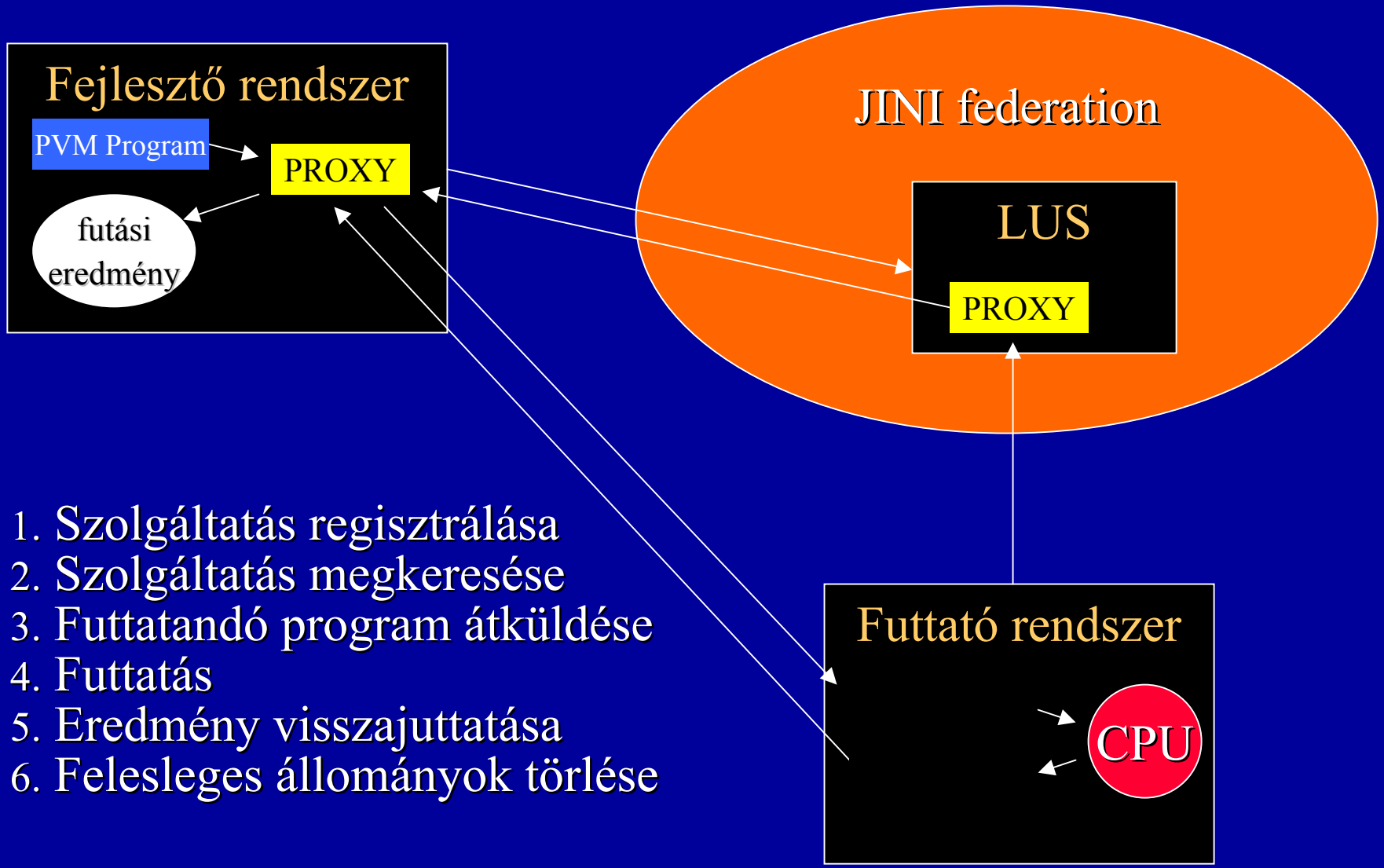
- A Jini federation Lookup Service-einél (LUS) a szabad, és PVM program-futtatásra képes gépek bejegyezhetik magukat, mint futtató szolgáltatást
- A P-Grade programot tartalmazó gép a LUS-eknél keres futtatásra alkalmas gépeket.

  (valójában PVM program - futtató szolgáltatást)
- A szolgáltatáshoz tartozó proxy segítségével:
  - átviszi a futtatandó PVM programot és a futáshoz szükséges adatokat
  - kezdeményezi a futtatást
  - visszahozatja a futási eredményeket
  - kezdeményezi a feleslegessé vált állományok törlését

A Jini legnagyobb előnye

Segítségével a futtatásra alkalmas rendszerek dinamikusan változó halmaza könnyen nyilvántartható.

A teljes folyamat forgatókönyve



1. Szolgáltatás regisztrálása
2. Szolgáltatás megkeresése
3. Futtatandó program átküldése
4. Futtatás
5. Eredmény visszajuttatása
6. Felesleges állományok törlése

A megvalósítás menete 1.

1. A szabad gépek megkeresik a Jini Federation-ben található LUS-eket.
2. Regisztráltatják szolgáltatásukat egy vagy több LUS-nél. A regisztrálás során a proxy objektumuk letárolásra kerül.
3. A P-Grade-et tartalmazó gép megkeresi az elérhető LUS-eket és lekérdezi a náluk elérhető szolgáltatásokat.
4. Kiválasztja a számára szükséges szolgáltatást (pl. „Szabad LINUX PVM klaszter") és letölti a hozzá tartozó proxy-t.

A Jini infrastruktúra funkciói biztosítják

A megvalósítás menete 2.

5. Az előzőleg JAR-fájllá csomagolt, a P-Grade programhoz tartozó program és adatfájlokat a proxy segítségével elküldi a szabad gépeknek.
6. A szolgáltató kicsomagolja őket, és átadja futtatásra a helyi PVM démonnak.
7. A szolgáltató a fájlba mentett konzol-kimenetet, hibafolyamot, illetve a kliens által igényelt további futás közben keletkező állományt JAR fájllá csomagolja.
8. A proxy segítségével az „eredmény JAR fájlt” az eredeti gépre visszajuttatja.
9. Az eredmény sikeres visszajuttatása után a proxy kezdeményezi a futás után feleslegessé vált állományok törölését.

A szolgáltatás funkciói biztosítják

Felmerülő kérdések 1.

- Mi legyen a felhasználó dolga?

  PVM programot átadni a proxy-nak. Semmi több!

  (PVM program helyett egy Java program futtatás.)

- Hogy történjen ez az átadás?

  1. Kliens HTTP szervert indít, melyen keresztül a JAR-fájl elérhető.
  2. A proxy-nak a JAR fájl URL-jét adja.
  3. A proxy továbbítja a címet a service-programnak.

  (Az eredmény visszatöltése ehhez hasonló.)

- Hogy kommunikál a proxy a szolgáltatóval?

  RMI hívásokkal.

  Az RMI nem titkosított, titkosítással nem foglalkozunk.

Felmerülő kérdések 2.

- Kliens azonosítás legyen?
  - Ha igen:
    - A kulcs eljuttatása a szolgáltatóhoz hogy történjen?
    - Megoldható az azonosítás „aláírt” JAR fájlokkal?

Válasz: Ne.

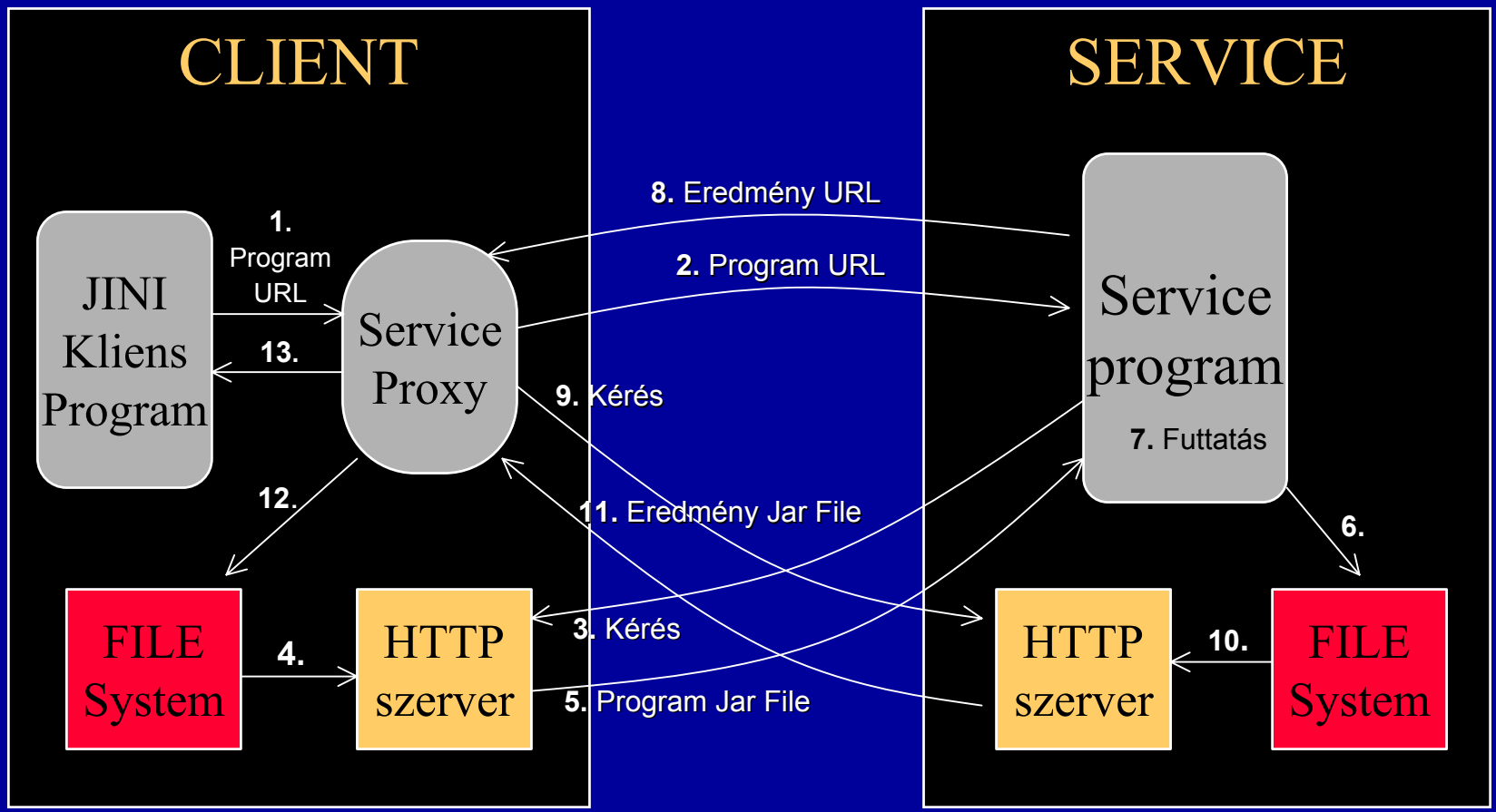
A szolgáltató a klienst nem azonosítja – bárki igénybe veheti a szolgáltatást aki a Jini szövetség tagja.

Felmerülő kérdések 3.

- A szolgáltatás igénybevétele kizárólagosan történjen?
  - Ha igen:
    - a LUS-ból kliens kapcsolódása után el kell távolítani a proxy-t? vagy
    - Proxy marad a LUS-nél de a service nem szolgál ki több klienst?
  - Ha nem:
    - Hogyan különböztetjük meg hogy az egyes kliensek melyik programot töltötték fel?
- Válasz: Nem.

  Emiatt nem a klienseket, hanem a futtatandó programokat kell megkülönböztetni egymástól: minden feltöltött JAR fájl más-más könyvtárba kerül.

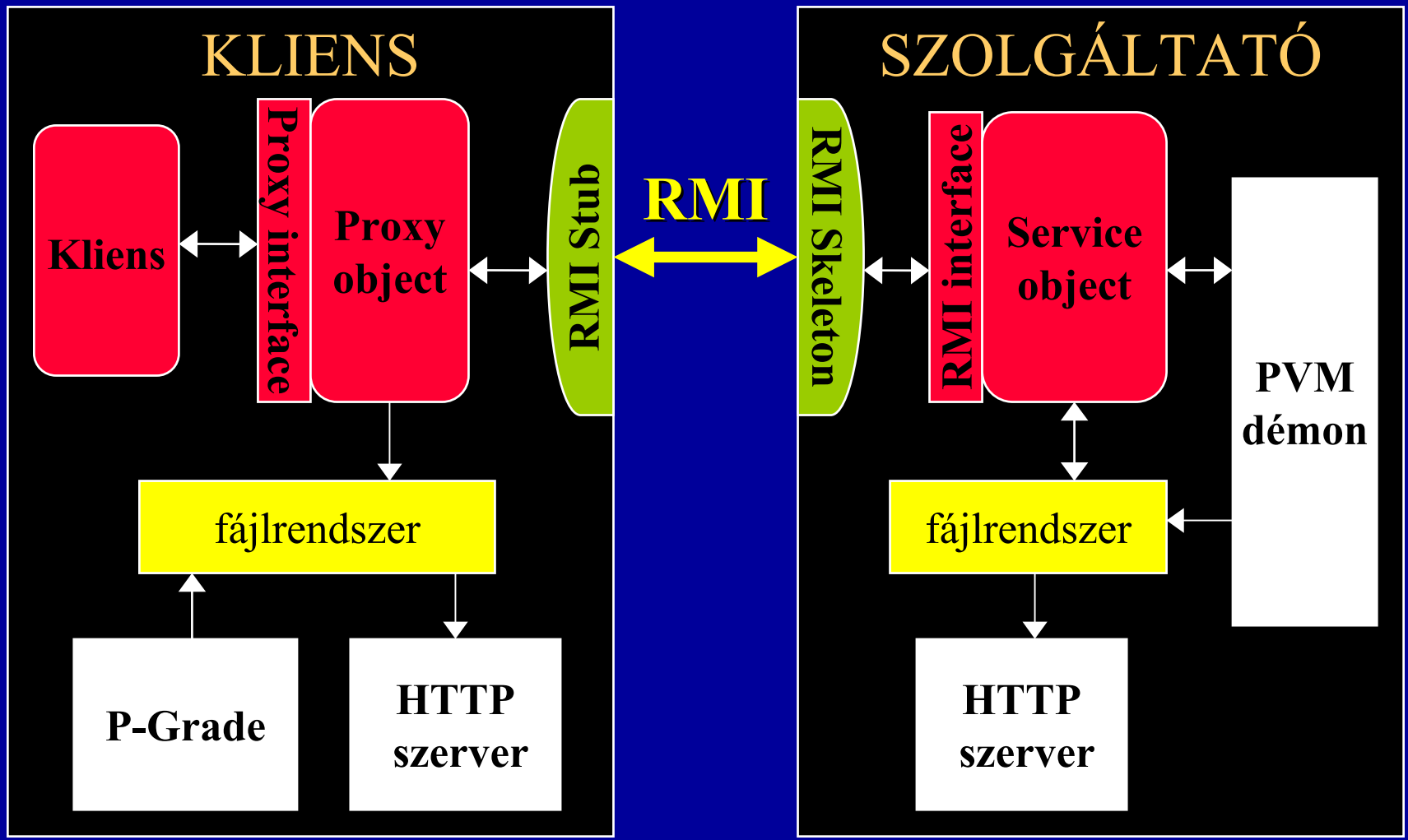
Részletezett forgatókönyv a proxy letöltése után



Lépések

1. A kliens átadja az URL-t a proxinak.
2. A proxi ezt eljuttatja a szervernek.
3-6. A szerver erről az URL-ről letölti a JAR fájlt.
7. A szerver kicsomagolja a JAR fájlt, feldolgozza az információs fájlokat és elvégzi a futtatást. Az eredményekre vonatkozó információk alapján JAR fájlt készít az eredményekből.
8. Átadja a proxinak az URL-t, ahonnan letölthető az elkészített JAR fájl.
9-12. A proxi letölti az eredményeket.
13. Visszajelez a kliensnek, hogy a futtatás el van végezve.

Az igénylő és a futtató rendszerek komponensei



Proxy Interface

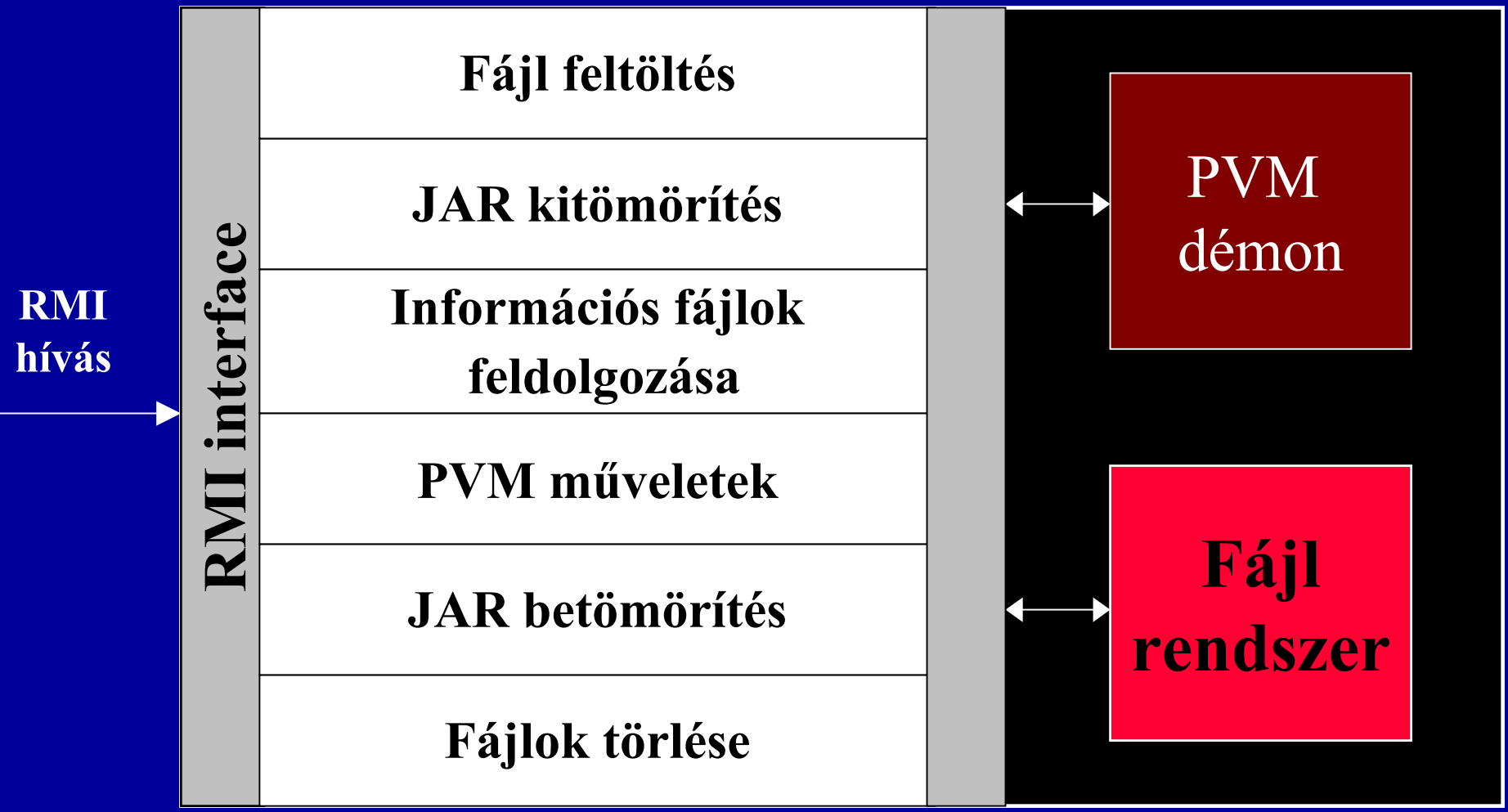
- A kliens a LUS-ban „ezt” az interfészt implementáló szolgáltatást keresi
- Tartalma jelen esetben egyetlen függvény, mely a PVM program „submit”-tálását végzi
- Paramétere az URL, ahonnan a PVM program és a szükséges fájlok JAR fájlba csomagolva letölthetőek

```
public void executePVMProgramInJarFile(URL location)
   throws RemoteException, IOException
```

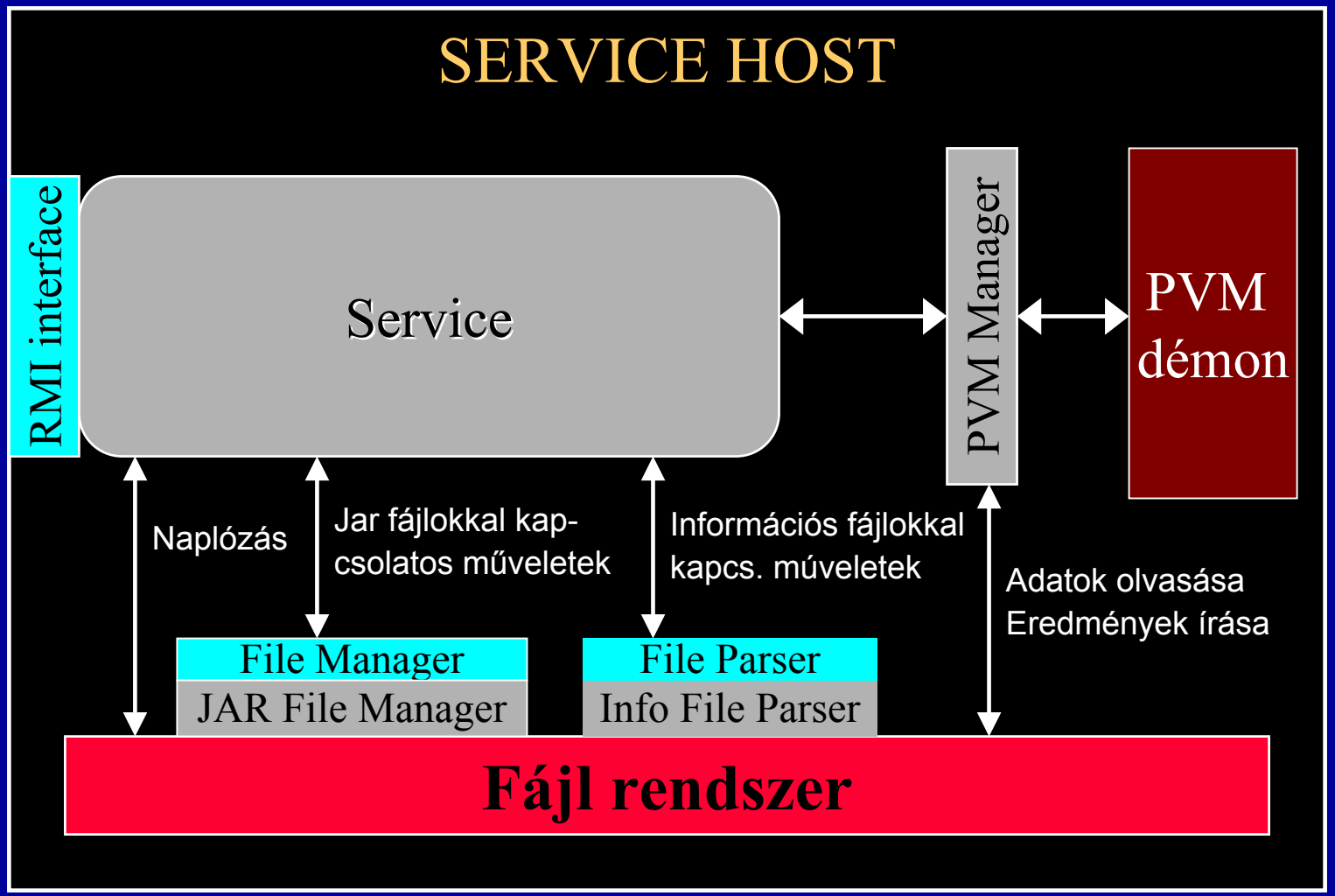
RMI Inteface

- A szerver oldal és a proxy közti kommunikációt megvalósító függvényeket deklarálja
- A proxy a klienstől kapott kérés hatására sorban meghívja függvényeit:
  - Feltöltés kezdeményező, melyben paraméter a klienstől kapott JAR file URL, és sikeres feltöltés esetén visszakap egy azonosítót, mely a további függvények hívási paramétere
  - Kitörömítést kezdeményező
  - Végrehajtást kezdeményező
  - Eredményfájlok betömörítését kezdeményező, mely visszatérési értékként szintén egy URL-t kap, ahonnan letölthető az eredmény JAR fájl
  - A felesleges fájlok törlését kezdeményező függvény

A szolgáltató program funkciói



A szolgáltató program komponensei



File Parser

- Az információs fájlokból a szükséges adatok kinyerését végző függvények deklarációja:
  - `getExecutionCommand()`: A futtatandó fájl nevét adja a futtatási paraméterekkel együtt
  - `getResultFileCommand()`: A visszavárt eredményfájlok nevét adja vissza
- Ennek egy lehetséges implementációja az Info File Parser, amely az alábbi formátumú fájlok feldolgozását végzi:

Információs fájlok

A futtató rendszer részére a futtatási paraméterek átadása ezekkel a fájlokkal történik

- Futtatási információk: „execute.inf”:

  egy sorból áll, ami futtatandó fájl neve a JAR fájlon belül a futtatási paraméterekkel

  (valójában a parancssor)

  pl: LINUX/Buffer

- Eredményfájlok felsorolása: „result.inf”
  - Visszaküldendő kimeneti fájlok

    (minden fájl neve külön sorban)

File Manager

- Ez az interfész a szolgáltatóhoz történő feltöltést és a fájlok be- és kicsomagolását végző függvényeket deklarálja:
  - `upload()`
  - `extract()`
  - `archive()`
- Ennek egy lehetséges implementációja a JAR File Manager
- Az interface implementálásával lehetőség nyílik a későbbiekben más fájltömörítési módszert is támogatni.

PVM Manager

A PVM program futtatását végzi el:

- Összeállítja a parancssort
- Ezzel rendszerhívást generál
- A standard kimenetet és a hibafolyamot kiírja fájlba :
  - `consol.out`
  - `error.out`

A szolgáltatás indítása

- A fejlesztéskor használt szoftverek:

  Jini 1.2.1

  Java 2 SDK 1.4

- Kicsomagolás után a `RemoteRunService` jegyzékbe kerül a program. Ennek tartalma:

  ```
  Classes/
  Policy/
  Sources/
  Start/
  ```

- A forrásfájlok a `source/` jegyzékben találhatók

  Lefordíthtóak az alábbi módon:

  ```
  cd start
  ./compile
  ```

A szolgáltatás indítása 2.

- Ezzel a class fájlok bekerülnek a nekik megfelelő könyvtárba:
  - `client/`: kliens oldal
  - `service/`: szerver osztályok
  - `service-dl/`: szolgáltatás igénybevételekor letöltődő osztályok
- A szolgáltatás elindítható a mellékelt scriptekkel a `start/` jegyzékből. Ezekben az alábbi környezeti változók értékeit kell beállítani:
  - `$JINI_HOME`: Jini telepítési jegyzék (pl. ~/jini_1_2_1)
  - `$JDK_HOME`: JDK telepítési jegyzék (pl. /usr/local/j2sdk1.4.0)

A szolgáltatás elindítása

- A LUS elindítása valamely host-on
- PVM démon elindítása (`PVM_ROOT/pvm`)
- WEB szerverek elindítása

  1.Proxinak

  ```
  start/startServiceHTTPserver port1
  ```

  (Home könyvtára a `classes/service-dl` lesz)

  2. Eredményfájloknak

  ```
  start/startResultHTTPserver port2
  ```

  (Home könyvtára a `classes/service` lesz)

  A portok különbözőek, és 2000-10000 –ig terjedhetnek.

- Szerverprogram elindítása

  ```
  start/startService Ipcím port1 port2
  ```

  Az IP cím az adott gépé, a portszámok az előzőleg megadottak. (sorrendjük is az előzőleg felsorolt)

Kliens indítása

1. HTTP szerver indítás a JAR fájl feltöltéséhez:

   `start/startClientHTTPserver port`

   IP : a kliens gépének címe

   port : 2000-10000 közötti

2. Kliens program indítás:

   `start/startClient IP port filename`

   IP : a kliens gép IP-je

   port : az előzőleg elindított HTTP server portja

   filename : a classes/client könyvtárban lévő jar fájl, amit végre szeretnénk hajtatni a szolgáltatóval.

Egy PVM program távoli végrehajtása

- Az „info” fájlok elkészítése szövegszerkesztővel
- A futtatandó PVM program JAR fájlba csomagolása az adat és „info” fájlokkal együtt:

```
$JDK_HOME/bin/jar -cf jarfile *.*
```

- A jarfile classes/client könyvtárba másolása
- A kliens oldali gépen a WEB szerver és a kliens program indítása megfelelő paraméterezéssel az előzőeknek megfelelően.
- A futási eredményül kapott result.jar fájlt a proxy a classes/client/result könyvtárba menti.
- Futtatás után a WEB szerver leállítása a

  start/stopHTTPserver port

  paranccsal. (a port az indításkor megadott)

Policy fájlok

Jogok adományozása szükséges a kliens és a szerver részére is, ezek az alábbi fájlokban találhatóak:

`policy.MyClient:`

```
permission net.jini.discovery.DiscoveryPermission „*”;
permission java.net.SocketPermission „*”, „connect, accept, resolve”;
    // a proxy ide menti az eredmenyt:
permission java.io.FilePermission „result”, „read”;
permission java.io.FilePermission „result/-”, „read”;
```

`policy.RemoteRunService:`

```
permission net.jini.discovery.DiscoveryPermission „*”;
permission java.net.SocketPermission „*”, „connect, accept, resolve”;
//logolashoz:
permission java.io.FilePermission „service.log”, „read, write, delete”;
// a LUS –től kapott Service paramétereket menti bele
permission java.io.FilePermission „serveicestate”, „read, write, delete”;
// munkakönyvtár a kliens programokhoz:
permission java.io.FilePermission „workdir”, „read, write, delete”;
permission java.io.FilePermission „workdir/-”, „read, write, delete”;
// a PVM rendszerhívás miatt:
permission java.io.FilePermission „<<ALL FILES>>”, „execute”;
```