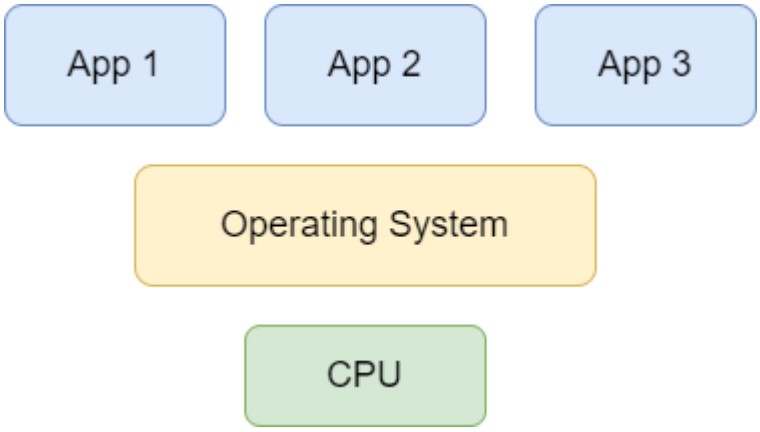
Informatikai Rendszerek

Az alábbiakban bemutatjuk, hogy alapvetően milyen módszerekkel lehet informatikai rendszer-komponenseket fejleszteni. Az informatikai rendszerek és egyes komponenseik abban különböznek a hagyományos alkalmazásoktól, hogy elvárjuk tőlük, hogy **szolgáltatásként közel állandó rendelkezésre állással működjenek**. Viszont ahhoz hogy, egy alkalmazás/szoftver szolgáltatásként tudjon működni, rögtön felveti a következő kérdéseket:

- hogyan lehet a komponens életciklusát vezérelni?
- hogyan tud gazdálkodni a környezete erőforrásaival?
- honnan/hogyan kaphatja meg a futásához szükséges konfigurációs információt?
- hogyan tud kommunikálni a környezetével?

Natív fejlesztési módszer



Bár ez a legrégebbi módszer, mégis sok tekintetben ma is alkalmazzák. Például beágyazazott rendszerekben. A forráskódot egy adott CPU és operációs rendszer kombinációra fordítjuk le: pl. (amd64/Ubuntu). Lehetőség nyílik a futtatandó kód finomhangolására, sebességének vagy méretének optimalizálásra. A komponensek folyamatos futtatásának követelménye komoly fejlesztői felkészülést kíván.

- c/c++/d fordítók: msvc, gcc, clang, dlang
- pointerek, referenciák, heap/stack memória kezelés
- a lefoglalt memória felszabadítása a fejlesztő feladata
- nagy kihívást lehet: API hívásokat operációs rendszer szinten ismerni kell
- a különböző komponensek integrációja körülményes: a szerilaizációt egyedileg kell implementálni

Nincs beépített erőforrás kezelés

- az erőforrás gazdálkodás a fejlesztő feladata

Nincs széles körben használt függőségkezelés

- a felhasznált függőségek (mások által fejlesztett komponensek) szabványos kezelése nem

egységes

RELEASE/DEBUG módú fordítási módszer lassítja a fejlesztést

- előfordulhat, hogy másik fordítóval a program működése eltér (hibás, lassabb)

Az alkalmazás életciklusát (indítását, leállítását, monitorozását) az operációs rendszer kezeli

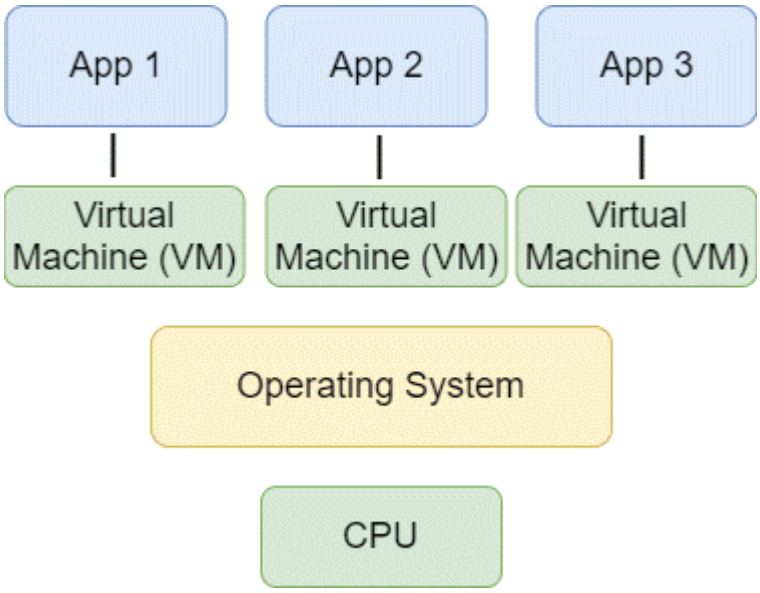
Speciális alkalmazási területei

- maximális tranzakciósebességre van szükség
- IOT eszközök: nem áll rendelkezésre elegendő memória a futtatáshoz vagy FPGA megoldások szükségesek

Hibák

- Nincs szabványos kivételkezelés
- A fejlesztőnek kell kezelni a hibákat, a lekezeletlen problémák rendszerösszeomláshoz vezetnek
- Nagyon körülményes a hibák felkutatása (memória dump, speciális log-ok)
- Nagyon könnyű hibázni – nem inicializált adat struktúrák

Virtuális gépes fejlesztési módszer



A Java VM bevezetése (1997-) óta terjedt el. Egy virtuális processzort és a hozzá tartozó úgynevezett Byte Kódot, saját gépi kódú utasításkészletet definiál. A forráskódot nem közvetlenül a CPU-ra fordítja le, hanem a virtuális gép saját byte kódjára, amit futtatáskor a gazda rendszer gépi kódjára alakít át. Az virtuális gép általában c/c++ nyelven írt natív alkalmazás, ami általában több plattformon is működik.

Fontosabb virtuális gép implementációk

- Java Virtual Machine
- NodeJS, chromium
- Common Language Runtime (CLR): .net rendszer
- Zend Engine: php
- Adobe Flash Player: swf futtatás
- HHVM: php alapú VM a facebook fejlesztésében
- ABAP: SAP virtuális gépe
- Python: VM
- LLVM: ez nem a klasszikus VM, hanem a forrást egy u.n. llvm byte kódra fordítja, majd ez fordul le natív kóddá. “LLVM is designed around a language-independent intermediate representation that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.”

Just in Time (JIT) fordítás

A virtuális gép képes az alkalmazások kódját folyamatosan optimalizálni, a byte kód átalakítás dinamikus.

Memóriakezelés

- Pointerek használata tiltott (általában)
- Szemétgyüjtési algoritmus felel a nem használt memória felszabadításáért

Beépített erőforrás kezelés

- az erőforrás gazdálkodás a fejlesztő feladata
- a VM rendelkezik erőforráskezelő lehetőségekkel

Beépített, széleskörben használt függőségkezelés

- a felhasznált függőségek (mások által fejlesztett komponensek) szabványos kezelése egységes (pl. Maven)

RELEASE/DEBUG módú fordítási módszer nem értelmezett

- a fejlesztés a korábbi DEBUG módhoz hasonló, a nyomkövetésnél az optimalizáció rejtve marad

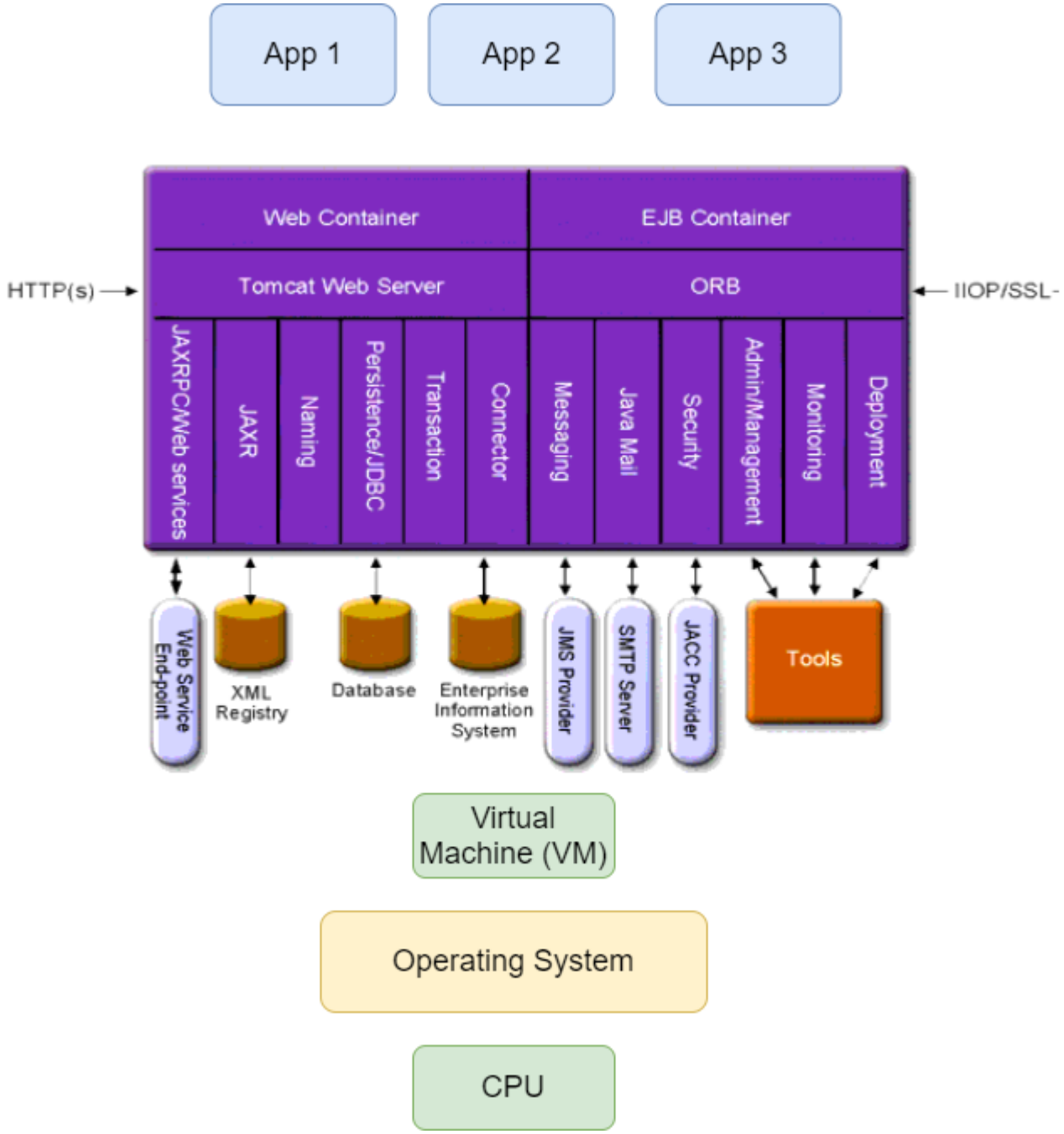
Az alkalmazás életciklusát (indítását, leállítását, monitorozását) a virtuális gép kezeli

Komponensek fejlesztése

- együttműködő komponensek fejlesztéséhez ideális, mivel a VM-en futó alkalmazások TCP/IP

segítségével könnyen kommunikálhatnak egymással
- a hálózati objektumokat önelemzés segítségével könnyen lehet használni, illetve módosítani. (Java reflection)

Middleware fejlesztési módszer



Alkalmazás kiszolgálós fejlesztési módszer. Eredetileg a Sun Microsystems adta ki 1999-ben akkor még J2EE néven. A szabványos specifikáció jelenleg a 8-as verziónál tart (2017): https://javaee.github.io/javaee-spec/

- Jellemzően Java nyelven implementált Middleware-t alkalmaz. Ismertebb (26) Java middleware
- Fontosabb Glashfish, Websphere, Weblogic, JBoss, Wildfly
- Az alkalmazások teljes életciklusát a middleware kezeli.

Web Container: webes komponensek életciklusának kezelése.

Servlet: Olyan Java osztály ami Http kérések szabványos feldolgozásáért és válaszaiért felel. Eredetileg a dinamikus Web tartalmak létrehozásáért felel. A generált tartalom HTML, de újabban JSON. Tartalmaz URL mapping-et is. 1996-ban mutatták be először, mint koncepciót!

- Automatikusan generálható servleteket is létre lehet hozni a a JSP techniológia segítségével, ahol a HTML kód tartalmazhat Java kódokat is.
- **HTTP kérések**: GET, POST, PUT, DELETE, OPTIONS

Java servlet API:

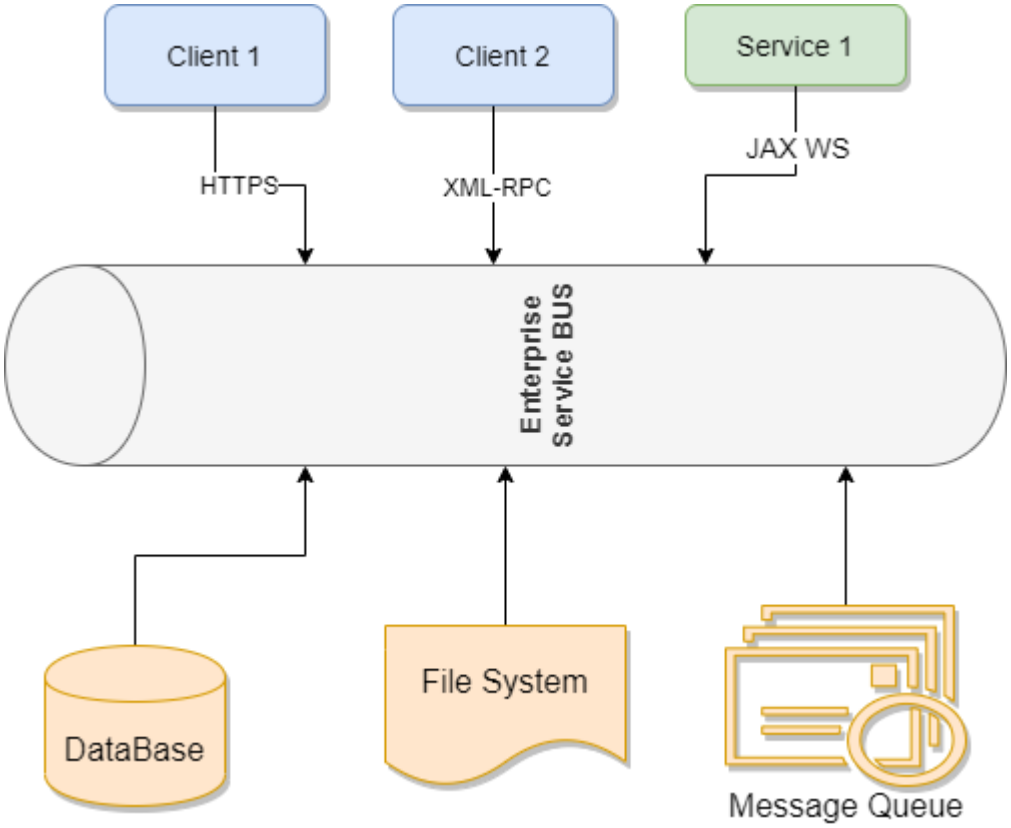
- Java API for RESTful Web Services (JAX-RS 2.0)
- Java API for XML Web Services (JAX-WS)

Mintapélda:

```
  public class MyServlet extends HttpServlet{

    public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
        {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();

        pw.println("<html><body>");
        pw.println("Hello from servlet");
        pw.println("</body></html>");

        pw.close();
        }
    }
```

A metaadatok kezelése az első változatokban XML leírókkal történt. Ezekben lehetett megadni, hogy egy osztály hogyan viselkedjen: URL mapping, futó példányok száma, stb.

ESB - Enterprise Service Bus



Szolgáltatás BUSZ: Szolgáltatás Orientált Architektúra (SOA). Lazán összekapcsolt komponenseken (szolgáltatások) alapul. A hálózatoknál ismert BUSZ fogalom analógiája.

Legfontosabb funkciók:

- Üzenet továbbítás - Message Routing a szolgáltatások között
- Szolgáltatás felderítés
- Különböző protokollok konverziójának támogatása
- Validáció - séma validáció
- Szolgáltatások verziókezelése
- Monitoring szolgáltatások
- Üzleti folyamatok menedzselése

Előnyök:

- könnyen skálázható használat - lokális szolgáltatástól a teljes vállalti elérésig
- az integráció implementálása (kódolás) helyett, konfigurációk kialakítása
- lazán kapcsoltság miatt könnyen lehet szolgáltatásokat indítani, leállítani

Hátrányok:

- lassú kommunikáció
- központosítás miatt, hiba esetén teljes leállás lehet
- nagy komplexitás a konfigurációban

Ismertebb implementációk:

- Azure Service Bus, Microsoft Biztalk Server, Mule ESB, Oracle ESB, IBM Websphere ESB, JBOSS ESB