Real-Time Java implementációk összehasonlítása

Önálló laboratórium feladat összefoglalója (1. félév)

Monostori Dénes (NPZMLU)
Konzulens: Horváth Ákos
BME Méréstechnika és Információs Rendszerek Tanszék
Informatikai infrastruktúra tervezése szakirány, 2007/2008. I. félév

Az elmúlt évtizedben valós idejű rendszerek fejlesztésére túlnyomórészt az Ada, C, C++ nyelveket használták. Ennek legfőbb oka, hogy ezek biztosítani tudják az ilyen rendszerekkel szemben támasztott szigorú időzítésbeli követelmények teljesítését. Ugyanakkor a szoftverek komplexitásának növekedésével egyre bonyolultabbá, nehézkesebbé vált a használatuk. Felmerült az igény egy magasabb szintű, egyszerűbben kezelhető, nagyobb produktivitást biztosító programozási nyelv használatára.

A „hétköznapi" alkalmazásoknál bizonyított standard Java megfelel ezen igényeknek, viszont a valós idejű rendszerek követelményeit nemdeterminisztikussága miatt nem képes teljesíteni. Éppen ezért elkezdték kutatni, milyen módon érhető el, hogy azoknak is megfeleljen. Több mint 50 nagyobb cég összefogásával kidolgoztak egy nyílt specifikációt RTSJ (Real-Time Specification for Java) néven, elősegítendő valós idejű rendszerek építését. Erre alapozva többen elkészítették saját Real-Time Java implementációjukat, más-más hangsúlyt helyezve az egyes kulcsfontosságú elemekre. Erőfeszítéseiknek köszönhetően a Real-Time Java-ban írt programok sok területen utolérték, illetve bizonyos alkalmazásokban meg is előzték a fent említett nyelveken megvalósított rendszerek teljesítményét, valamint jósolhatóvá tették a programok futását. Az egyik leggyorsabban fejlődő terület, ahol az RTSJ igen jól teljesít, az a nagyméretű elosztott üzleti alkalmazások területe, ahol a határidőn kívüli teljesítések közvetlen bevételvesztést okozhatnak.

A félév elején az IBM implementációját bemutató cikksorozat első négy részét elolvasva ismerkedtem meg a Real-Time Java alapjaival, illetve a legfontosabb kihívásokra (szemétgyűjtés, szálak ütemezése, szinkronizálása, fordítás stb.) a WebSphere Real-Time 1.0 által adott megoldásokkal. Ezt követte előbb a Sun JRTS 2.0 telepítése Solaris 10-re, majd a nyílt forrású jRate 3.7.2 lefordítása és installálása Nuke Linuxra. Tapasztalataim feljegyzése után a Sun által kifejlesztett Garbage Collector működésével és finomhangolásával ismerkedtem meg. Utóbbit a JVM-nek átadott paramétereken kívül akár az MBeans nevű API-n keresztül is elvégezhetjük, bár ez még tesztelés alatt áll. A általam fellelt többi implementáció mindegyike kereskedelmi termék, így beszerzésük kétséges.

A továbbiakban a Sun JRTS 2.0 és a jRate (esetleg más implementációk) teljesítményének összehasonlítása érdekében méréseket fogok végezni. Ennek alapjául egy általam elkészített egyszerű modelltranszformációs program szolgál. A mérések célja első körben annak megvizsgálása, hogy mennyire válik jósolhatóvá a teljes modell transzformációjának időigénye a program puha valós idejű változatában, azaz képesek leszünk-e valamiféle korlátot felállítani a futási időre a bemenő modell függvényében. Az alkalmazás standard Java-ról történő átírása során érdekes kérdésnek ígérkezik, hogy az RTSJ-ben leírt két új memóriamodell használata milyen előnyökkel járhat esetünkben. Egy ígéretes eszköz lehet az implementációk összehasonlítására a Garbage Collector paramétereinek finomhangolása, mivel ebben eléggé eltérő megoldásokat kínálnak az egyes megvalósítások.

Célom hosszútávon a különféle modelltranszformáció közeli nyelvi elemek vizsgálatával annak kiderítése, hogy milyen módon lehetne a modelltranszformációk egy szűkebb részhalmazát esetlegesen valós idejű környezetekbe adoptálni, lévén rengeteg üzleti alkalmazás már így is sok területen használja egyes elemeit (pl. minta illesztés, XML adatok transzformálása stb.). Hogy ezt érdemes-e megtenni, arra többek között a mérések eredményei adhatják meg a választ.