Absztrakt

A Modell Vezérelt Fejlesztés(MDE) jelentős szereppel bír a kritikus rendszerek fejlesztése során, melyeknél fontos az érdekelt felek közötti közreműködés biztosítása, mialatt a megfelelő szellemi tulajdon védelme minden esetben megvalósul. Több hasznos fejlesztő eszköz áll rendelkezésre a különböző ipari fejlesztési területeken, melyek általában biztosítják a modell alapú verziókezelést, azonban a legtöbb technológia csak fájl szintű hozzáférést, verziókezelést kínál, így a modell különböző részekre bontásához és különböző jogkörök kiadására egyáltalán nincs mód vagy csak körülményes, drága, komplikált lehetőségek adódnak.

Ezen fő problémák, kihívások teljesítésének megsegítésére jött létre a Mondo Collaboration Framework, egy szabály alapú hozzáférés szabályozási rendszer, mely modell szintű biztonságos hozzáférést, tulajdon alapú lockolást és automatikus modell összefésülést biztosít minden felhasználója számára. A Mondo Framework a fájlok tárolásához és verziókezeléséhez a Subversion(SVN) már létező verziókezelő technológiát veszi alapul, kihasználva a technológiában rejlő lehetőségeket. Ezen keretrendszer segítségével a modellt használó, szerkesztő vagy megtekintő felek csak azon részeit látják a modellnek, melyeket a rendszergazda előre beállított, így biztosítva a modell(termék) integritását, megóvva akár az ipari versenytársaktól, csak a megmutatni kívánt részeit publikálva.

A megalkotott keretrendszer architektúrája folyamatos szinkronizációt biztosít az elemei között. A rendszer használatához minden esetben szükség van pontosan egy adminisztrátori repository-ra és egy vagy több végfelhasználó repository-ra. Az adminisztrátori repository neve Gold repository a végfelhasználó repository neve Front repository. A Gold repository-ban, megtalálható a teljes modell, amiből publikálhatjuk a modell egy részét vagy akár az egészet bármelyik front felhasználó repository-jának. Természetesen különböző felhasználók különböző modell részletekkel rendelkezhetek, így jöhet létre az az eset például, hogy az első felhasználó látja a modell egyik felét, a második felhasználó a másik felét, semmit nem látva az első felhasználó modelljéből, miközben a valóságban egy és ugyanazon modellen dolgoznak. Ezen tulajdonságra építve egyazon cég különböző dolgozóinak munkái is megfelelő módon elszeparálhatóvá tehetők, elkerülve a nem kívánt jogköri vitákat, félreértéseket, hibákat.

A célom a keretrendszer szinkronizációjának fejlesztése volt. A fejlesztés során a létező szinkronizáció belső logikája nagyrészben megmaradt, azonban a kódbázis továbbfejlesztve, Java nyelven került újraírásra, ezáltal operációs rendszer-függetlenné téve a szinkronizációs elemeket. A továbbfejlesztések között szerepel a jobb válaszidő, átláthatóbb kód.

A működés egy alappillére, miszerint a felhasználó által létrehozott változtatások egyből felkerülnek a saját front repository-jából a Gold repository-ba és onnan tovább az összes többi front repository-ba, természetesen csak akkor, ha az aktualizálandó front repository-nak van joga látni a modell megváltoztatott részét. A szinkronizáció biztosítása a keretrendszer működéséhez alapvető, ezért a különböző futás idejű hibák, hibalehetőségek kiküszöbölése elengedhetetlen a megfelelő működés biztosításához. A törekvések ellenére minden felmerülő hibát nem lehet javítani a futási idő alatt. Ilyenkor a rendszergazda kezelheti a felmerülő problémát, a keretrendszer által biztosított eszközökkel.

Bevezetés

A modellekre alapuló rendszerek egyre népszerűbbek lettek a kritikus rendszerek terén, például az autógyártás és repülőgéptervezés iparágakban, ahol a minőség és a produktivitás folyamatos fejlődése fontos tényező a gyártási technológiák megválasztásakor. A rendszer integrációs eszközökre vetett igény mára már meghaladja az fejlesztőeszközök kitermelésének ütemét. Egyre gyakrabban van szükség az együttműködésre elosztott, különböző szakterületeken dolgozó (rendszer integráló, szoftver fejlesztő, hardver fejlesztő), különböző elhelyezkedéssel rendelkező csapatoknak.

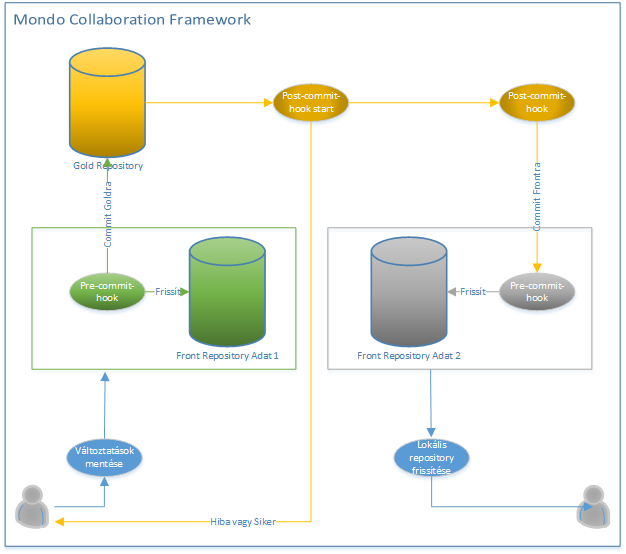
A keretrendszer kétféle együttműködést biztosít, offline és online együttműködés formájában. Offline együttműködés során, aszinkron módon valósulnak meg a modellen történt változtatások, a frissítés egy hosszabb tranzakció keretein belül történik. Online módban a felhasználók egy webes frontend felületet kapnak, ahol a létrehozott változtatások, egyből szinkron megtörténnek a többi felhasználó modelljén. Több modellező keretrendszer létezik a piacon, viszont a biztonság kezelése még elég kezdetleges fázisban van. A biztonság megfelelő kezelése azonban sok felhasználási területen fontos követelmény lehet. Könnyen előfordulhat egy vállalatnál, hogy egy belső modell egy részét vagy komponensét meg kell mutatnia egy külsős szervezetnek, például egy tanúsítvány megszerzéséhez, de a modell többi részét szükség lenne elrejteni az esetleges vetélytársaktól, megvédve ezzel a cég szellemi tulajdonát.

Egy ilyen együttműködés komoly biztonságtechnikai kihívásokat rejt magában. A legtöbb jelenlegi modellező keretrendszer, a modell fájlokat a már létező verziókezelő szolgáltatásokkal használja, mint például a Git vagy a Subversion, melyek csak fájl, projekt vagy repository szintű biztonsági szabályzást engednek. Megoldás lehet, ha a modell fájlt feldaraboljuk több, kisebb modell részletre, melyeket már lehet egyesével szabályozni, azonban ilyenkor a fájlok száma megsokszorozódik, 100-1000 modellrészt már nehéz kezelni, esetleg újra felhasználni.

A Mondo Collaboration Framework céljai között voltak a hagyományos verziókezelő rendszerek kibővítése biztonságos modellezési együttműködési funkciókkal, mint például a model-részlet szintű hozzáférés, tulajdon alapú zár, automatikus modell összefésülés, és online offline együttműködés biztosítása. Ezen kitűzött célokat az eredeti rendszer teljesítette.

A bemutatott keretrendszer jövőre néző céljai között szerepel a rendszer folyamatos továbbfejlesztése; a minél gyorsabb, hatékonyabb munka és jobb felhasználói élmény elérése érdekében. Én a szinkronizációs eljárások fejlesztését tűztem ki célul, a keretrendszer offline együttműködéses módjában.

A rendszer hiányosságai között szerepelt többek között az operációs rendszer függőség. Mindkét szinkronizációs eljárás, melyek névszerint egy ’Front pre-commit hook’ és egy ’Gold pre-commit hook’ , linux bash szkriptként voltak megírva, ezáltal erősen operációs rendszer függővé téve a szinkronizációs eljárásokat. Általános probléma továbbá egy szkript írásakor a szkript hossza. Minél hosszabb lesz a megvalósított szkript, annál valószínűbb, hogy a megírt kód gyorsan átláthatatlanná, egy idő után kezelhetetlenné válik. Egy a Mondo Collaboration Framework-höz hasonló keretrendszer szinkronizációjának megvalósításához jobb lehetőséget kínálnak az objektum orientált nyelvek, melyek nem, vagy sokkal kevésbé veszítenek az olvashatóságból a komplexitás növelésével. Az én választásom a Java nyelvre esett, mert ez a nyelv nem csak a modern objektum orintált felfogást képviseli, egyben operációs rendszer független is, tehát az újra megvalósított szinkronizációval a keretrendszer könnyebben fejleszthető más operációs rendszerekre is.

 A rendszer második fontosabb fejlesztési célkitűzése a felhasználói élmény javítása volt. A szinkronizáció során a felhasználó által elindított változtatások felvezetése a Gold repositoryba és onnan a változtatások publikálása a többi felhasználó felé viszonylag lassú folyamat volt. A változtatások időbeli létrejötte erősen függ a fő modell méretétől, így nagyobb modellek során előfordult a több perces felhasználói várakozás is, amíg a felhasználó visszajelzést kapott a beküldött változtatások sikeréről vagy meghiúsulásáról. A probléma megoldását egy új, szkriptben eddig nem létező program megírása jelentette, mely elindítja ugyan a szinkronizációs folyamatot, azonban egyből visszatér a felhasználóhoz, így javítva a rendszer válaszidejét a felhasználó felé. Természetesen a megfelelő ellenőrzések így is megtörténnek, ezért nem jöhet létre az az eset, hogy a program helyes működéssel tér vissza, a változtatások mégsem hajtódnak végre. Ha a program hiba nélkül fut le, akkor minden esetben megtalálható a Gold repositoryban az új, már módosított modell.

1 Szinkronizáció lépései

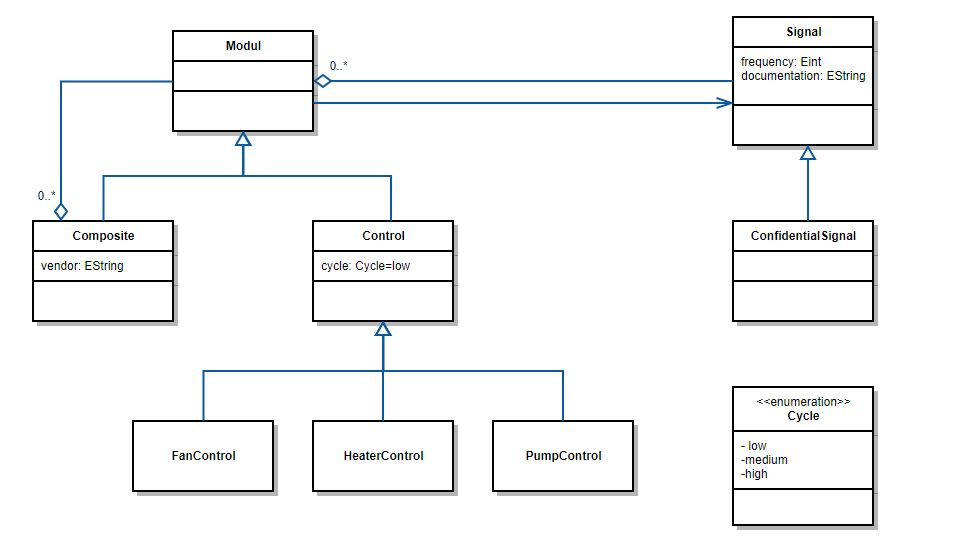
Összesen három programmal javítottam a keretrendszer működését. Az első a Front pre commit hook program, mely egyből a felhasználó commitja után, de még a front repositoryban történt változtatások mentése előtt hajtódik végre, a második a gold post commit start hook, ami a gold repositoryban történt változtatások létrejötte után hajtódik végre és meghívja a harmadik programot, a Gold post commit-ot. A második program egyből visszatér a felhasználóhoz, amíg a harmadik program a háttérben szinkronizálja a Gold repositoryban történt változtatásokat a többi felhasználó felé. A szinkronizáció alapelveit és a repositoryk közötti kapcsolatot az 1. ábra szemlélteti.

A működés a felhasználó által indított ’svn commit’ utasítás (változtatások frissítése a reposítoryba) kiadása után kezdődik. A kiadott parancs után a felhasználó Front repository-ján meghívódik az első program, a pre-commit hook. A pre-commit hook egy olyan utasítássorozat, mely még azelőtt fut le, hogy megtörténne a commit, ezáltal akár vissza is lehet utasítani azt. Általában az SVN-be beépített hookokat formátum ellenőrzésre, kódellenőrzésre szokás használni, hogy ne lehessen olyan változtatásokat eszközölni a projekten, amiket nem preferál a vállalat, vagy a vezető fejlesztők. Jelen modellező keretrendszer ezen hookok lehetőségeit használja ki, hogy megfelelő logikával biztosítsa a szinkronizációt az architektúrális elemek között.

A pre-commit hook lefutása során meghív egy belső commit-ot a Gold repositoryra, ezzel szinkronizálva a felhasználó változtatásait a Gold Repositoryba. Fontos tudni, hogy a Gold repositoryban előbb jelennek meg mindig a változtatások, mint a változtatásokat létrehozó felhasználó Front repositoryjában, így elérve, hogy mindig a fő repository frissül a leghamarabb. A Gold repositoryn történt változtatás után (már létrejött commit után) meghívódik a második program, mely visszajelzi az első felhasználónak a sikeres vagy sikertelen commit ereményt. A harmadik post-commit program végigiterál az összes további Front repositoryn, és a Goldon történt változtatásokat egyesével meghívja rajtuk. Mivel minden Front repositorynak van pre-commit hook-ja ezért ezek fognak először lefutni, de mivel Goldról jön már a commit, ezért egyből beengedik a commitot, ezzel megakadályozva a végtelen ciklus lehetőségét.

A Java-ban újraírt, és újonnan megírt programok belső logikája a régi, szkriptek logikájára épít, megtartva a régi keretrendszer verzió fő funkcionalitását, kiegészítve azokat új elemekkel, átláthatóbb kóddal, ezzel is segítve a jövőbeli fejlesztéseket. Hosszútávú cél lehet akár az összes operációs rendszer-függő elem lecserélése független elemekre.

Esettanulmány

A keretrendszert egy rendszerintegrációs esettanulmányon keresztül mutatom be, egy egyszerűsített modellező nyelv felhasználásával. A rendszert nyílt tengeri szélturbinák szoftveres vezérlésének beállítására használták fel, ahol a különböző modell jogkörökkel rendelkező mérnökök együttműködése volt a kitűzött cél. Az esettanulmány a MONDO EU FP7 projekt esettanulmánya volt.

2.Egyszerűsített Metamodell

A metamodell a 2. ábrán látható, és a rendszer különféle moduljait, és moduljaiból felépülő rendszert mutatja be. A modulok tudnak jelezni egymásnak, és el tudják nyelni egymás jelzéseit. Egy modul küldhet több modulnak több különböző jelzést és fogadhat is szintúgy azokat. A jelzések üzenetet tartalmaznak, és a gyakoriság ’frequency’ attribútum által beállított érték gyakoriságával hívódnak meg. Ezen felül minden jelzéshez tartozik egy dokumentáció attribútum(documentation), ami leírja a jelzés felelősségeit, ezzel szabályozva, hogy melyik jelzés használható különböző szituációkban, események következtében. A jelzések egy fajtája a bizalmas jelzés (Confidential Signal), mely jelzi a jelzés bizalmas mivoltát. A bizalmas jelzéseket nem tekintheti meg az összes felhasználó erre külön jogkörök tartoznak. A modulok két különböző csoportra lettek osztva eredetük szerint. Az első csoport a vegyes (Composite) modulokat tartalmazza, amiket a külső eladók szállítmányoznak, az eladó személye az eladó(vendor) attríbútumban tárolódik el. A második csoport a fizikai vezérlő eszközöket szimbolizáló csoport, mint például a szélturbina forgólapátjainak a vezérlése, vagy a hőmérséklet szabályzó vezérlése. Ezek a vezérlő fizikai egységek a vezérlő (Control) modulok. A vezérlő csoport moduljainak ciklus prioritását, a modul ciklus (cycle) attribútumja határozza meg.

A szélturbina különböző vezérlő egységeinek tervezéséhez megfelelő szaktudásra van szükség. Három fajta vezérlő egység található a metamodellben, bármelyik egység módosításához megfelelő szaktudású mérnök szükséges, így három különböző szaktudású mérnököt igényelve a rendszer, a fan(forgó lapát??) fűtés és szivattyú vezérlő mérnököt. Fontos cél a három mérnökcsoport elhatárolása egymástól, hiszen a más csoportban lévő mérnökök nem értenek egymás munkájához. A véletlen vagy akaratlagos módosítások egymás munkájába, akár súlyos károkat is okozhatnak. Ezért a különböző szaktudással rendelkező mérnököknek nem megengedett a többi mérnök munkáinak módosítása, különleges esetben akár olvasni sem olvashatja a nem rá tartozó modell részletet. Ezeknek a kritériumoknak a teljesítésére lett kiépítve a projektre egy alkalmazott szabályrendszer, mely garantálja az elszeparálását a különböző mérnöki ágazatoknak.

Minden dolgozó mérnökhöz hozzárendelünk egy hozzá tartozó vezérlő elemet, ezzel birtokába kerül a vezérlő elem. A mérnök mindig csak a saját vezérlő elemét módosíthatja, és csak ennek az elemnek a kimenő és bemenő jelzéseit tekintheti meg. A mérnök látja az összes olyan vegyes modult, ami tartalmazza a mérnök vezérlő modulját, vagy ha létezik egy modul, ami almodulja a mérnök birtokolt moduljának. A mérnök által birtokolt moduljainak jelzéseit módosíthatja, és megfigyelheti, mely más modulok nyelik el az ő modulja által kibocsátott jelzéseket. Az utolsó szabály, hogy egyetlen mérnök sem láthatja a bizalmas jelzéseket. Ezen szabályok betartásával egyértelművé válik, ki melyik modellrészlethez fér hozzá, így létrehozva a kívánt szeparációt.

A rendszer integrációs vállalat a szélturbina vezérlő modelljét egy a keretrendszert futtató szerveren tartja, ahol tárolja a modell fájlokat és verziókezeli azokat. A felhasználók kétféleképpen léphetnek interakcióba a szerverrel. Az első mód az online együttműködéses üzemmód, a második az offline együttműködéses üzemmód.

Online együttműködéses üzemmódban a felhasználók egy online együttműködésben vesznek részt, ahol folyamatosan kapcsolatban vannak a központi repositoryval, egy erre a célra kifejlesztett frontend kliensen keresztül, például egy felhasználó által választott böngésző által. Minden felhasználó a legfrissebb modell verziója képét látja, az általa látható, módosítható modellrészletről. A változtatások felvezetése a fő modellbe, folyamatosan, menet közben történik, rövid tranzakciók keretein belül. Minden tranzakció egy saját műveletet tartalmaz, úgymint létrehoz, frissít, töröl vagy mozgat. A keretrendszer fontos feladata, hogy el kell utasítania minden olyan műveletet, mely sértené a biztonsági előírásokat. A felhasználók modellt módosítani a felhasználói kliensen keresztül tudják, ami az elvégzett módosításokat egyből felküldi a fő modellt tartalmazó szervernek. Miután megkapta a változtatásokat, a szervernek döntenie kell, hogy a felhasználó által bevezetett módosításokhoz van-e a felhasználónak joga, a szerveren található írási korlátozások alapján. Ha megengedett a változtatás, az összes felhasználó nézete frissül egyből a tranzakció végrehajtása után, feltéve, hogy a felhasználónak van olvasási joga a frissített modellnézetre.

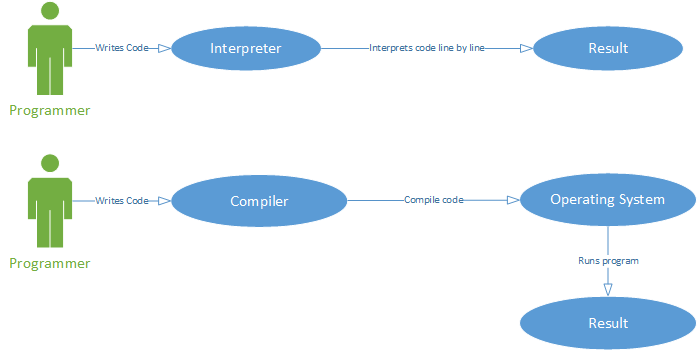
Offline együttműködéses módban minden felhasználó letöltheti a szerverről a jogkörének megfelelő modell részletet. A modell letöltése után a felhasználó lokálisan megnézheti, feldolgozhatja, módosíthatja a saját modelljét. A modellt olyan eszközzel fejlesztheti, módosíthatja, ami a legkényelmesebb számára, mert az eszköznek nem kell tudatában lennie a keretrendszer sajátosságaival. Ha a felhasználó elvégezte a módosításait, egy hosszabb tranzakció keretein belül tölti fel a szerverre a módosításait.

A Mondo Collaboration Framework az elvárt feltételeket teljesítette. A szélturbina vezérlőn dolgozó mérnökök sikeresen tudtak együtt dolgozni, egymás munkájának zavarása, szaktudási területének megsértése nélkül. A modell hozzáférés szabályozási rendszer sikeresnek bizonyult.

Háttértechnológiák

1. Java

A szinkronizációs programok tervezésekor a legfontosabb technológiai elvárások a programmokkal szemben, a gyors futási idő, az operációs rendszer függetlenség elérése és a jól strukturált könnyen átlátható, értelmezhető kód írása volt. Ezeknek a feltételeknek az általam választatott Java nyelv eleget tesz, habár a gyors futási idő mindig relatív, főleg a többi szempont figyelembe vételével. Valószínűsíthető, hogy a megírt programok gyorsabban futnának C-ben, azonban a többi elvárás már nehezebben teljesíthető lett volna, ha erre a nyelvre esik a választás.



Az eredeti szinkronizációs eljárások linux bash szkriptként voltak megvalósítva, melyek ugyan könnyen és gyorsan implementálhatók voltak, viszont semmilyen formában nem használhatóak másik operációs rendszeren. A szkript nyelven megírt szkripteket a linux héj interpretere soronként értelmezte a megírt kódot. Ez a megoldás egy elég egyenes megoldás a megoldandó feladatra, viszont sajnos a szkript nyelvek sebessége soha nem fogja meghaladni az előre lefordított nyelvek sebességét, hiszen a kód mindig futás időben van értelmezve. Másik hátránya a szkript nyelvnek, a komplexitás növelésével, nagyban nő a kód komplexitása, egy idő után nehezen értelmezhetővé téve azt. Az utolsó szempont egy programozási nyelv választása mellett a megfelelő hibakezelés volt. Szkript írásakor a hibák kezelése nehézkes, a programozónak sokszor nincs elég eszköze a hibajelenségek lekezeléséhez. Java programozási nyelvre váltás után jóval kedvezőbbek az esélyek a stabil szinkronizációs háttér előállítása a keretrendszerhez.

A programozási nyelvek kódja a szkript nyelvek kódjával ellentétben fordító program által vannak lefordítva, így egy futtatható állományt létrehozva előre, a programozó által írt kódból. A futtatható állomány kiterjesztése operációs rendszer és nyelv függő, tehát ez még önmagában nem garantálja az operációs rendszer függetlenséget, viszont már így is nagy előnyt jelent a hagyományos szkriptekkel szemben, mert nem szükséges az utasításokat futás időben értelmezni. Könnyen belátható a gyorsabb működés egy egyszerű példán keresztül. Az általunk használt számítógép csak magyarul beszél és nekünk olaszul vannak utasításaink, amiket szeretnénk, hogy a számítógépünk elvégezzen. Szkript nyelvek esetén odaadjuk a számítógépnek az olasz utasítássorozatot, majd az soronként lefordítja és értelmezi azokat. Nagy segítség a számítógépnek, ha már az utasításokat egyből magyarul adjuk oda neki (programozási nyelvek fordítás után) és neki csak végre kell hajtania a megfelelő utasításokat. A fordítást olaszról magyarra egy külső fordító program végzi, ami nevében is utal az általa betöltött szerepre.

A fordító a legtöbb esetben operációs rendszer függő, azonban ez alól a Java nyelv kivétel, mert a Java kód egy bytekóddá fordul először, melyet a Java Virtual Machine (JVM) értelmez és futtat, így fenntartva az operációs rendszer függetlenség látszatát. A kód egy bytekóddá fordul, melynek kiterjesztése általában a ’.jar’, melyet bármelyik rendszer futtatni képes, ahol telepítve van a JVM. Mivel a JVM operációs rendszer függő, ezt minden rendszeren külön telepíteni szükséges. Szerencsrére azonban elég népszerű a Java nyelv, így a gyakorlatban használt rendszerek nagyrészén már megtalálható, vagy utólag telepíthető a JVM, így megoldva az operációs rendszer függetlenség problémáját.

A Java programozási nyelv a fentebb említett előnyein kívül, egy másik pozitív változást hoz az új programok írásához. Objektum orientált, osztály alapú nyelvként komplex működést lehet megvalósítani jól értelmezhető, könnyen dokumentálható, könnyen újra felhasználható módon. Nagy hangsúlyt kap az objektum orientált felfogásban az egységbezárás gondolata, miszerint egy osztályban található az összes osztályhoz tartozó attribútum, és a rajtuk értelmezett műveletek összessége. Egy osztálynak lehetnek attribútumai és metódusai. Egy osztály egy példányát nevezzük objektumnak.

Az utolsó, de nem elhanyagolható pozitív tulajdonsága a választott nyelvnek, az aktív programozói bázis mérete. Jelenleg az egyik legtöbbet használt nyelv a világon, és ezt mutatja a Java külső könyvtárak mennyisége és minősége. Ha problémába ütközne egy programozó, könnyen utánakereshet, mert szinte biztos, hogy valaki más is beleütközött már abba a problémába. A programozást nagyban segítik a Java nyelvbe beépített könyvtárakon kívül, a mások által megírt külső könyvtárak, amiket nehézkes, vagy hosszú idő lenne megírni magunknak. Programozás során sokszor törekedünk arra, hogy ne találjuk fel újra a kereket, ezért próbálunk már létező, mások vagy magunk által megírt elemeket használni.

Összeségében a Java nyelv választása a programok megírásához megoldás volt a felmerülő problémákra, tervezett fejlesztésekre.

1. Verziókezelés

A verziókezelés fontos része a szoftverrel kapcsolatos fejlesztéseknek. A verziókezelő rendszerek biztosítják a lehetőséget a kódbázisban történt változtatások követésére, szoftverfejlesztő csapatok együttműködésének biztosítására. Általában egy projekt nem csak lokálisan (csak speciális esetekben) érhető el, hanem legtöbb esetben egy szerveren található. Egy másik eltérő módja a verziókezelésnek, ha nincs dedikált szerver, és a verziókezelés egy peer-to-peer modellre alapul, ahol csak a felhasználóknál tároljuk a projektet és azokat szinkronizáljuk egymással. Jelen projekt esetében ezek a verziókezelő rendszerek kizártnak tekinthetőek voltak, hiszen a keretrendszer alapstruktúrája nem engedné meg ezt a megoldást, és az általam írt programoknál is haszontalan lett volna, mert egyedül dolgoztam a projekten.

A verziókezelés alapja, ha projekten módosításokat végzünk lokálisan, azokat fel kell töltenünk a szerveren található projektbe, ahonnan a többi fejlesztő megkapja a projekten végzett módosításokat. A szerveren levő projektet bármelyik, hozzáféréssel rendelkező fejlesztő is módosíthatja, így biztosítva a fejlesztők közötti együttműködést. A verziókezelő rendszerek különbözőségeit figyelembe véve, nem mondható ki általános folyamat a projekt verziókezelésére. A munkám során két különböző verziókezelő szoftvert használtam, az egyik a manapság legfelkapottabb Git, a másik régebbi, viszont szintén jól működő Apache Subversion(SVN). A saját projektemen belül Git-et használtam, a keretrendszer szinkronizációs programokon belül pedig SVN-t, a megrendelő kérése alapján.

A két verziókezelő rendszer több aspektusban is hasonlít egymáshoz. A kiadott parancsok nagyban eltérnek egymástól, azonban mindkét rendszerben léteznek parancsok, amik megfeleltethetők egymásnak. Egy felhasználó munkafolyamata mindkét esetben egy lokális repository létrehozásával kezdődik, Git esetében a git clone, SVN esetében az SVN checkout parancs kiadásával. Ha létrejött a lokális projekt repository, a fejlesztő elkezdhet dolgozni a projekten. Érdemes minden napi munka elkezdése előtt, vagy akár naponta többször frissíteni a lokális projekt repositoryt a git pull és az svn update parancsok kiadásával, hátha más fejlesztők módosítottak a projekten. A módosítások megtekinthetőek a git status és az svn status parancsok kiadásával. Miután a fejlesztő elvégezte az általa végzett munkát, a módosításokat a git add ’fájl neve’ git commit –m „message” git push parancsokkal tudja feltölteni a git szerverre, és svn add ’fájl neve’ svn commit –m „message” parancs kiadásával tudja feltölteni az Apache Subversion szerverre. Ha ugyanazon a fájlon dolgozott két fejlesztő akkor létrejöhetnek összefésülési konfliktusok, melyeket a fejlesztőknek kézzel kell feloldaniuk. A projekten dolgozó fejlesztők dönthetnek úgy, hogy új projekt ágat hoznak létre, tehát az alap projektet nem módosítják, annak egy tökéletes másolatát hozzák létre és azon dolgoznak tovább. Ha befejezték a másolat módosításait, akkor összefésülhetik a fő projekttel a projekt vezető beleegyezésével általában. Szükség esetén minden verziókezelőben vissza lehet térni előző, jól működő verziókra, így a projekt egyik verziója sem veszik el.

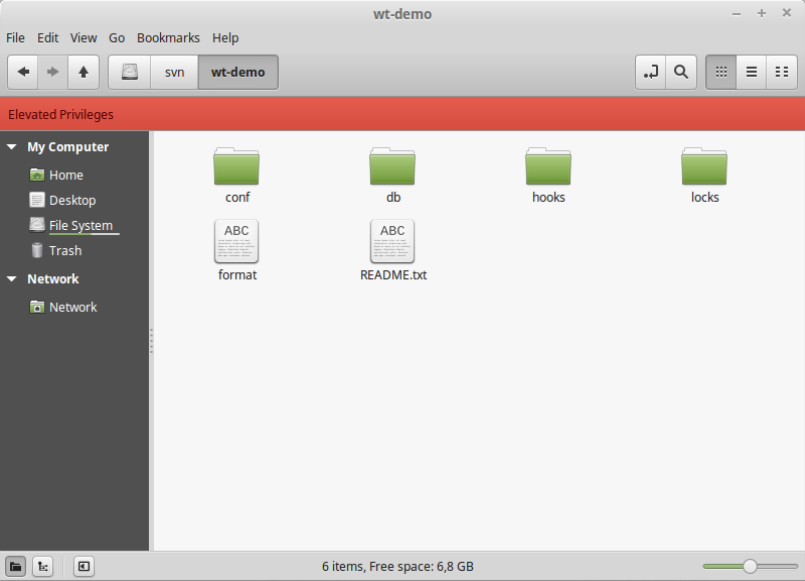
A keretrendszer szinkronizáció programjainak írásakor fontos szerepet kapott az SVN egy hasznos sajátossága a hook-ok kihasználása. Az SVN repositoryk commit operációjához hookokat rendelhetünk, melyek vagy a commit előtt, vagy a commit után lefutnak. Ezekben a hookokban szokás ellenőrizni a commit formai követelményeit, ilyenek például a commit üzenet, a kód formaiságának ellenőrzése. Az általunk írt commit előtti(pre-commit) és commit utáni(post commit) hookokban szinkronizációs logikát valósítottunk meg, melyben modell-transzformációs programokat is meghívunk.

Fontos megemlíteni a Subversion két különleges verziókezelést segítő számát, a TXN és revision számokat. Minden repositoryn végzett módosítás commit utasítás kiadásával lép érvénybe. Ha a commit utasítás sikeresen lezajlott, akkor létrejön egy új revision szám, ami pontosan eggyel nagyobb az előzőnél. Minden repository a létrehozásánál az egyes számról indul, majd minden módosításnál növelve az értéket, így átláthatóbbá téve a régebbi verziók közötti különbségeket. A TXN működése nagyban hasonlít a revision számra, azonban ez a szám a commit elfogadása előtt keletkezik, mielőtt a commit utasítás érvénybe lépett volna. A szám minden esetben a legutolsó commit száma kiegészítve egyedi betű és szám kombinációval, mellyel később a pre-commit programban egyértelműen azonosítható a tervezett commit.

Megvalósítás

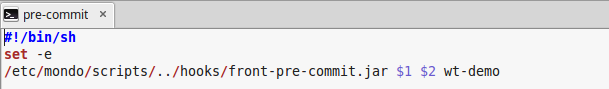
Ebben a fejezetben a három, általam megírt programot mutatom be részletesen, kitérve az azok által használt algoritmusok helyességére, különböző működési módjainak bemutatására. A három program neve a szinkronizációs folyamatban betöltött szerepükre utal. Az első program a front-pre-commit nevet kapta, ezzel jelezve működését, miszerint minden repository commit előtt lefut a program. A második program a gold-post-commit-start nevet kapta, melyet a gold repositoryba, fogunk elhelyezni, és minden commit esemény után indul a program futása. A harmadik program a post-commit, aminek futását a gold-post-commit-start program kezdeményezi.

A legelső lépés a szinkronizáció biztosításához, a programok meghívása a megfelelő repository-kból. Az szerveren tárolt összes repository felépítése azonos, amit a 4.Ábra mutat be.



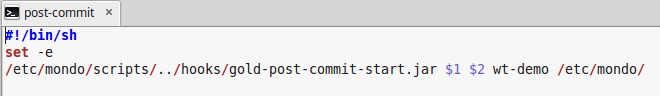
A hooks mappában az Apache Subversion által előre gyártott hook sablonok találhatóak. Sablonok közül nem használunk egyet sem, egy rövid scriptet használunk, ami meghívja a megfelelő pre-commit vagy post-commit programokat repository-tól függően.

Minden Front repository commit előtt le kell futnia a pre-commit programnak, ezért az összes szerveren található Front repositoryba beillesztjük az 5. ábrán található szkriptet.

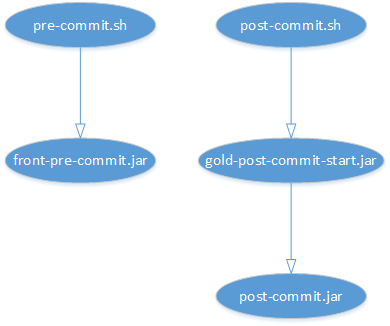


A szkript egyetlen funkciója, hogy meghívja az általam megírt front-pre-commit.jar programot. Az 5. ábrán bemutatott szkript az SVN által megadott argumentumokat kapja paraméternek ($1,$2) és ezeket egyenesen továbbítja a java programnak. Az első paraméter a front-pre-commit esetében a commit előtt álló repository neve, a második paraméter a commitra egyedien jellemző TXN, a harmadik paraméter a gold repository neve.

A front repositorykhoz hasonlóan a Gold repository-n is elhelyezünk egy egyedi szkriptet, ami biztosítani fogja, hogy meghívódik a gold-post-commit-start programom.



Az első gold-post-commit-start-nak átadott paraméter a módosított repository neve, ahol már létrejöttek a változtatások, hiszen commit után vagyunk. A második paraméter a commit után frissen létrejött revision szám, a harmadik a gold repository neve, a negyedik a szerveren egyik mappája, ahol dolgozni tud a program. A jelen konfiguráció szerint a /etc/mondo mappába dolgozik a post-commit, így például oda naplózza a működését, oda hozza létre a zár fájlokat, és maga a program is itt található. A programok egymásutáni lefutását a 6.ábra szemlélteti.



A szkriptek létrehozása automatikusan történik a keretrendszer szerverét konfiguráló szkriptben. Miután minden hook a helyére került, az általam írt java programokon van a felelősség a felhasználók repository-jainak közötti szinkronizáció biztosítására.

Zárolás