

Tartalomjegyzék

BEVEZETŐ	2
FELHASZNÁLÓI DOKUMENTÁCIÓ	5
RÖVID ISMERTETŐ	5
FELHASZNÁLT MÓDSZEREK	5
TELEPÍTÉS, KÖRNYEZET, SZÜKSÉGES BEÁLLÍTÁSOK	6
<i>Környezet</i>	6
<i>Támogatott platformok</i>	6
DEFINÍCIÓK	6
FELHASZNÁLÁS	8
<i>A program futtatása</i>	8
<i>A repositoryból megszerezendő artifactok megadása, betöltés</i>	8
<i>Limitált működés</i>	10
<i>Hibaeset (hálózatmentes működés)</i>	10
<i>Az első POM (baloldali) és második POM (jobboldali) megadása</i>	10
<i>A POM megjelenítésének, függőségi vizsgálatának be és kikapcsolása</i>	13
<i>A függőségi gráf megjelenítése</i>	14
<i>A függőségi gráf vizsgálata, felhasználási ötletek</i>	14
FEJLESZTŐI DOKUMENTÁCIÓ	18
ÁLTALÁNOS INFORMÁCIÓK A PROJEKTRŐL	18
RÉSZLETES SPECIFIKÁCIÓ	18
<i>Az adatgyűjtésről</i>	19
<i>Adatbáziskezelés</i>	19
<i>A felhasználói felület</i>	19
FELHASZNÁLT MÓDSZEREK	21
<i>Felhasznált technológiák</i>	21
<i>A technológiák és absztrakt osztályok elhelyezkedése a program rétegeiben, struktúrájában (15. ábra)</i>	25
<i>Megoldási tervek, próbálkozások az adatgyűjtéshez</i>	25
<i>Maven Central Repository REST API</i>	27
<i>Az adatbázis megvalósítása</i>	28
<i>A program szerkezete</i>	30
<i>Alapvetően fontos osztályok</i>	33
TESZTELÉSI TERV ÉS EREDMÉNYEK	35
<i>Funkcionális tesztelés</i>	35
<i>Unit tesztelés</i>	38
IRODALOMJEGYZÉK	39
MELLÉKLETEK	40

Bevezető

Egy szoftverprojekt fejlesztésében nagy szerepet kapnak a különböző eszközök,

amik segítik a projekt gyorsabb és hatékonyabb lefolyását, menedzselését. Fontosak hisz rendszert visznek a fejlesztésbe, a projekt átláthatóbb lesz, kezelhetőbb.

Különösen hasznos ez mikor az adott projekten sokan dolgoznak. Nem szabad megengedni, hogy egy fejlesztő a többiek számára kevésbé érthető specifikációt, dokumentációt készítsen, mint ahogy azt sem hogy más, a csapattagok számára szokatlan módon fordítsa le a programját, hisz más fejlesztőknek is értenie kell mi történt az adott fejlesztési lépésben. Szerencsére az évek alatt több módszerrel, eszközzel ismerkedett meg a szoftvervilág, amely segít a projekt életciklusát kézben tartani, automatizálja a projektmenedzsment egyes lépéseit. Ezek a módszerek kombinálhatóak is attól függően mi a legmegfelelőbb a csapat számára, illetve mi alkalmazható a kiválasztott technológiára, fejlesztői környezetre, programnyelvre.

A szoftvervilág egyik legmeghatározóbb programnyelve a Java. Természetes, hogy a Java technológiákat használók körében is szükség van olyan eszközökre

amelyek képesek a projekt menedzselésére. Egy ilyen eszköz az Apache Maven szoftver.

A Mavent pontosan azért hozták létre, hogy egy Java alapú projektet könnyebben lehessen menedzselni illetve a buildfolyamatot automatizálni. Főképp képes összetett build folyamat irányítására, dokumentáció készítésére, kész termék csomagolására, de az egyik legnagyobb erénye a projekt függőségeinek kezelése.

Mikor egy szoftvert fejlesztünk kivétel nélkül támaszkodunk más fejlesztők által írt szoftverekre, könyvtárakra, csomagokra, hiszen az újrahasznosítás alapvető dolog

a programozásban. Ehhez környezetünkben elérhetőek kell hogy legyenek ezek a függőségek.

A Maven erre nyújt megoldást. Megadhatjuk, hogy a projektünk milyen külső

könyvtáraktól függ és a Maven megszerzi azokat nekünk. Ezeket a beállításokat a projekthez tartozó pom.xml nevű fájlban tudjuk elvégezni. A POM (Project Object Modell) fájl tartalmaz minden beállítást ami a projekt menedzseléséhez szükséges. Itt állíthatjuk be a kívánt build, csomagolási, tesztelési szabályokat. Ezen szabályokhoz szükséges függőségeket is itt adhatjuk meg, valamint azt is hogy a Maven melyik könyvtár tárolókban (Maven repository) keresse őket. Itt fontos megemlíteni, hogy a repositoryban található szoftverek verziója időről időre megváltozik nyilván, hisz ezek a szoftverek is élő fejlesztés vagy karbantartás álló projektek. Ezeket a verziókat szintén megtaláljuk a repositoryban, így egy adott függőségnek nem csak a legújabb verzióját használhatjuk, hanem egy régebbit is amennyiben számunkra az a megfelelő. A beállítás után a Maven kliens tudja értelmezni a POM fájlt, így a kívánt szabály lefutása már automatizált.

Tegyük fel, hogy be szeretnénk állítani a saját POM fájlunkat, megadni a könyvtárakat amikre szükségünk van. Több probléma merülhet ilyenkor fel. Az egyik nyilvánvaló probléma, hogy nem tudjuk a felhasznált könyvtár benne van-e a repositoryban. Amennyiben benne van, milyen verziók léteznek? Ez a könyvtár milyen más könyvtáraktól függ? Milyen problémák merülhetnek fel ha másik verzióját használnánk a könyvtárnak? Egy több komponenssel, komplex buildfolyamattal rendelkező project esetén nincs-e két hasonló funkcionalitással rendelkező könyvtár függőségként megjelölve? Egy adott függőség definiálása összeegyeztethető-e a már használt könyvtárak licencfeltételeivel? Tegyük fel, hogy nekünk is van több verziónk már a saját szoftverünkben. Könnyedén megeshet, hogy két verzió más-más függőségekkel rendelkezik. Nagyon hasznos lenne, ha létezne egy olyan szoftver, amely megmutatná a különbségeket, akár vizuálisan is, hiszen így könnyebben észrevehetőek lennének a különböző függőség verziókból adódható hibák. Ez a szoftver segítséget próbál nyújtani a fejlesztőnek, a gyorsabb és hatékonyabb szoftver fejlesztéshez vagy karbantartáshoz.

Működése két részre osztható. Az egyik és legfontosabb része, hogy képes egy Maven repository végigpásztázására. Az projekt adatok (Artifacts) összegyűjtésére, relációs adatbázisban tárolására. Így lehetőség nyílik további elemzések készítésére a későbbiekben. Ezt a modult fontos különválasztani,

mert így ez a szoftver a későbbiekben könnyedén továbbfejleszthető.

A másik része a megadott POM elemzése függőségi szempontból vagy egy másik POM-mal való összehasonlítása. Két eltérő projekt függőségeinek összehasonlítása is érdekes, de a legérdekesebb ugyanazon projekt más-más verziójának való összehasonlítása. Itt derülhetnek ki olyan fordítási vagy futási hibák, amik a különböző felhasznált könyvtárakból vagy azok különböző verziójából adódhatnak. Gondoljunk arra, hogy az általunk felhasznált könyvtárak is más könyvtáraktól függenek, így szükségszerű, hogy teljes mélységében végignézzük a függőségi gráfot. Ezért a szoftver képes az elemzésen túl a függőségi gráf vizuális megjelenítésére is.

A téma választásában az motivált, hogy alapvetően fontosnak tartom a fejlesztést támogató különböző eszközök használatát és továbbfejlesztését. Hiszen ez nagyban lerövidítheti a fejlesztési időt illetve több időnk maradhat olyan gyakran elhanyagolt fejlesztési lépésekre mint a tesztelés.

Felhasználói dokumentáció

Rövid ismertető

Egy szoftverfejlesztő napi szinten használ mások által írt könyvtárakat, modulokat, ez a Java fejlesztők körében sincs másképp, ebben (is) nyújt segítséget az Apache Maven.

A saját projektünkhöz egy POM fájlt létrehozva vagyunk képesek új szabályokat létrehozni illetve beállítani, milyen külső könyvtárakra lenne szükségünk. A Maven ezt a fájlt (xml struktúra) értelmezni tudja és a külső könyvtárakat a repositoryból letölteni számunkra.

A külső könyvtárak szintén ilyen módszerrel készültek, vagyis függenek más könyvtáraktól illetve más könyvtárak más verzióitól. Ilyen mélységbe mi már nem látunk bele a projektünk fejlesztése közben, így könnyedén megeshet, hogy egy felhasznált könyvtár új verziója már nem kompatibilis számunkra. Ezeket az apró különbségeket észrevenni nagyon nehéz. Ebben nyújt segítséget ez a szoftver.

A saját POM fájlunkat megadva képes azt elemezni és a függőségi gráfját kirajzolni a központi repository (Maven Central Repository) aktuális állapotában.

„Maven Central contains **over 260,000 artifacts and serves over 70 million downloads every week.**” Brian Fox - Sonatype [1]

Egy másik POM fájlt is megadhatunk aminek a függőségi gráfja szintén kirajzolódik. Nyilván azokon a pontokon ahol a két POM megegyezik a két részgráf összekapcsolódik. Minket azok a csúcsok érdekelnek amelyektől kezdve a két POM függőségi gráfja különbözik már, ugyanis itt akadhatunk hibára.

Felhasznált módszerek

Mivel a függőségek keresztbe hivatkoznak egymásra, magától adódik a könyvtárbeli függőségek gráffal való ábrázolása, ahol a csúcsok a könyvtárak amiktől függünk, az élek pedig maguk a függőségek. A két megadott POM fájlt együtt ábrázolva (csúcsként) könnyen kiderülnek a különbségek. Fontos

megemlíteni, hogy egy adott repository adatait kibányászva nézzük meg ezeket a különbségeket.

Telepítés, környezet, szükséges beállítások

A programot külön telepíteni nem kell. Használatához elég futtatni a bináris állományt. Erről bővebben a Felhasználásban olvashatunk.

Környezet

A program futtatásához szükségünk van a Java futtató környezetére. A Java futtatókörnyezete a JRE (Java Runtime Environment) és a benne foglalt JVM (Java Virtual Machine) virtuális gép segítségével tudjuk a programot futtatni. Amennyiben nincs telepítve JRE a számítógépünkre az ORACLE weboldaláról le tudjuk azt tölteni:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

A program adatbázis szervernek az Apache Derby megoldását használja. Az Apache Derby bele van integrálva a szoftverbe, beágyazott alkalmazásként, ezért külön telepítést, beállítást nem igényel.

Támogatott platformok

A program kvázi platformfüggetlen a Java virtuális gépnek köszönhetően, amely a legtöbb platformra elérhető, így a program futtatható Windows, Linux, iOS stb. operációs rendszereken is.

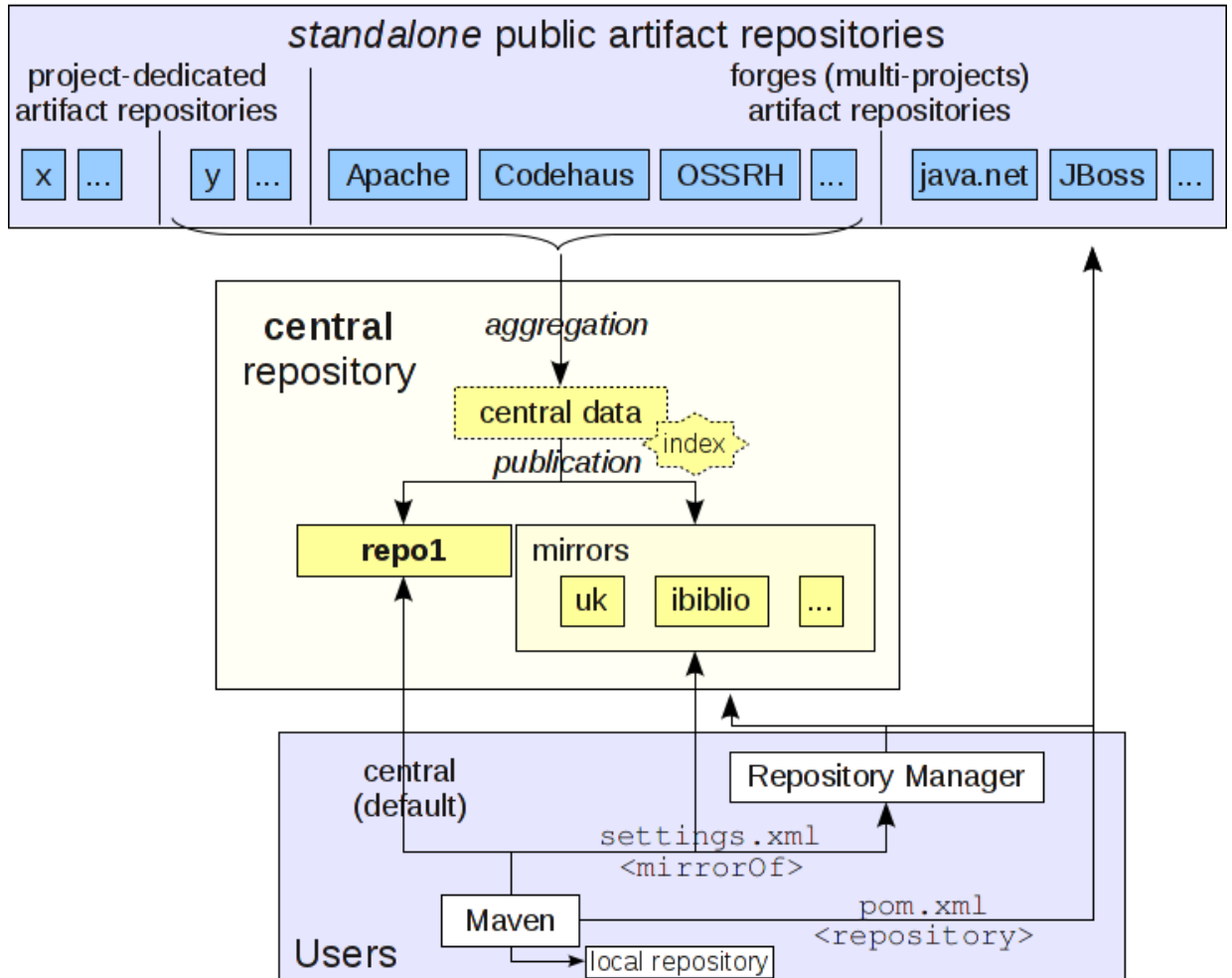
Definíciók

Sorban végigmegyünk a fontosabb kifejezéseken. A tudásuk alapvető egy Maven felhasználó számára.

POM: Project Object Model, azaz a **Projekt Objektummodell**. Egy POM egy buildelendő projektet ír le és annak függőségeit.

Repository: A Maven hálózatképes, tehát szükség esetén dinamikusan is le tud tölteni komponenseket. Repository névvel illetik a különböző hosztok fájlrendszereinek azon mappáit, ahol a letölthető komponensek találhatóak. A repository lehet lokális illetve távoli. (1. ábra) A lokális (local) repository szerepe az, hogy a Maven kliens ide tölti le a beállított függőségeket tranzitívan. A távoli

(remote) repository az amivel a Maven kliensünk kommunikál. Ez lehet a Central Apache Maven által karbantartott Repository, illetve lehet mások által üzemeltetett publikus repository.



1. ábra [2]: A repository „hálózat” struktúrája

Dependency: Függőség, vagyis egy könyvtár amitől a projektünk függ. Ezt a megadott függőséget a Maven automatikusan letölti nekünk a repositoryból.

GroupId: A konvenció alapján ez az azonosító a szervezet saját azonosítója úgymond kívülről befelé haladva (pl: org.apache.maven).

ArtifactId: A konvenció alapján ezzel azonosítjuk a projektet, ez a neve.

Version: A projektünk egy adott verzióját jelöli.

GAV – hármás – triple: A groupId, artifactId, version együtt azonosítanak egyértelműen egy POM fájlt, egy repositoryban.

Felhasználás

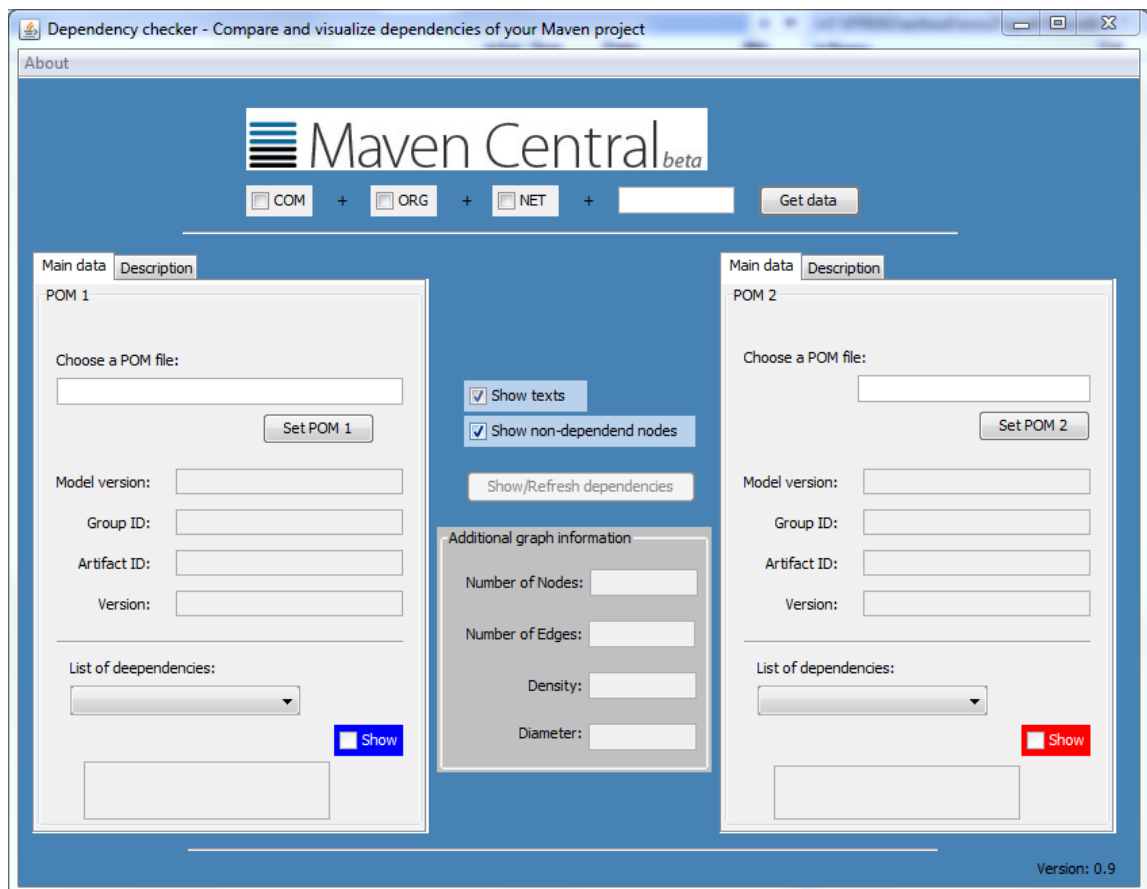
A program felhasználói szempontból nem bonyolult, egy fejlesztő számára nem jelenthet problémát a használata. Nézzük sorban.

A program futtatása

A programot a következő paranccsal indíthatjuk:

```
java -jar depchecker
```

Az elindítást követően a program betöltődik, használatra kész.



2. ábra

A repositoryból megszerezendő artifactok megadása, betöltés

A program felső területén tudjuk beállítani mely artifactokra van szükségünk.

Itt három jelölőnégyzetet látunk, illetve egy üres szövegmezőt. A három jelölőnégyzet (*com*, *org*, *net*) a leggyakrabban használt három groupId prefixet jelöli. A szövegmezőben megadhatunk számunkra más érdekes groupId prefixet is.

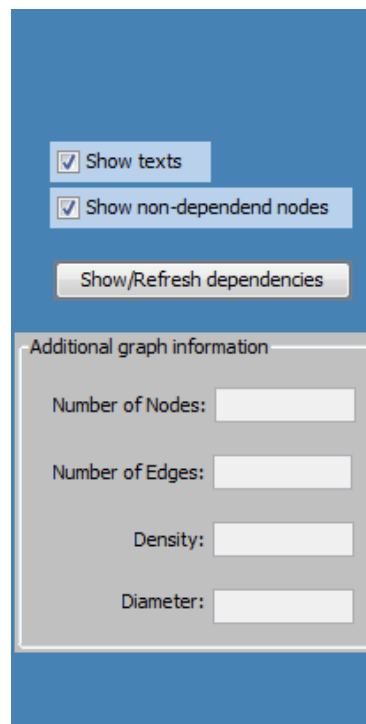


3. ábra

A "Get data" gomb megnyomására a program elkezd letölteni a megadott artifactokat a Maven központi tárolójából. Ezek az artifactok a program beágyazott adatbázisában tárolódnak el. Ezzel megszerezzük a Maven központi tárolóján található artifactok egy részhalmazát. A teljes halmaz letöltése nem lehetséges, ez óriási adatbázist eredményezne, az ebből adódó adatforgalom mennyisége miatt automatikusan tiltásra kerülhetünk.

„The **central repository is over 180 GB** and growing. To save us bandwidth and you time, **mirroring the entire central repository is not allowed**. (Doing so will get you **automatically banned**) Instead, we suggest you setup a repository manager as a proxy.”

A letöltés befejeztével aktív lesz a program középső területén található *Show/Refresh dependencies* gomb.



4. ábra

Limitált működés

A program nem töltheti le a teljes Maven központi adatbázist, de csak egy-egy groupId prefixre korlátozott betöltés is hatalmas adatbázist eredményezhet. Gondoljuk el, hogy csak a “com” prefixű projektekből 3884 db van és ez még mindig csak az adott projekt egy verziója. Általában egy projektnek több verziója megtalálható a tárolóban, így több tízezer fájlletöltésről beszélhetünk csak erre az egy prefixre.

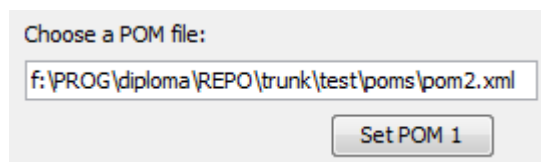
A betöltés tovább korlátozható a program könyvtárában létrehozott jelző fájl (flagfile) segítségével. Ezt a fájlt “limit.flg”-nek kell elnevezni. Létezésével a program korlátozott mennyiségben tölti le az adatokat. A szoftver jelenleg (demonstrációs célból) 10 projekt/prefix korlátozza a betöltést, azaz 10 projektet tölt be az összes verziójával együtt prefixenként. Ez a funkció a program esetleges kipróbálására is alkalmas lehet.

Hibaeset (hálózatmentes működés)

Amennyiben a Maven Central Repository valamilyen oknál fogva nem érhető el, a program hibaüzenetet jelenít meg. A program ilyenkor megkísérli betölteni az adatokat az adatbázis legutoljára sikeresen elmentett állapotában. Sikeres betöltés esetén a program hálózat nélkül is használható a továbbiakban.

Az első POM (baloldali) és második POM (jobboldali) megadása

Ahhoz, hogy egy projektünk függőségeit megvizsgálhassuk meg kell adnunk ezeket a projekteket. Egy ilyen projektet a POM XML formátumú fájlja ír le, ezért a baloldali és jobboldali panelben található “Choose a POM file” mezőbe (5. ábra) a kiválasztott két projekt POM fájljának az elérési útját kell megadnunk. A betöltéséhez a Set POM 1 illetve 2 gombok megnyomása szükséges, attól függően, hogy az 1-es vagy a 2-es panelben beállított XML fájlt akarjuk-e betölteni.



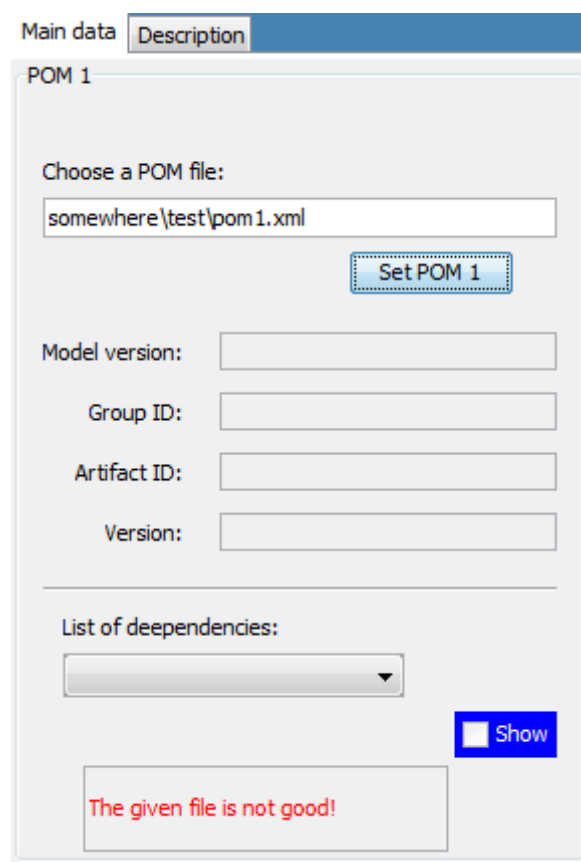
5. ábra

Fontos, hogy nem kötelező megadni POM fájlokat. Amennyiben egyik POM fájlt sem jelenítjük meg, a program csak a Maven Central Repository-ból letöltött

projektek függőségi gráfját mutatja be és elemzi. Csak egy POM fájl megadása esetén a program a projekt függőségi fájlát mutatja be a már említett letöltött repository részhalmazon belül. Két POM megadása esetén mind a kettő függőségi fa megjeleníthető.

Hibás fájl megadásának esete

Nem létező illetve nem érvényes POM fájl megadása esetén a POM panel alján található státusz mezőben jelenik meg a hibaüzenet. (6. ábra)



The screenshot shows a software interface with a tabbed window. The active tab is 'Description'. Below the tab, the title is 'POM 1'. There is a section 'Choose a POM file:' with a text input field containing 'somewhere\test\pom1.xml' and a 'Set POM 1' button. Below this are four input fields labeled 'Model version:', 'Group ID:', 'Artifact ID:', and 'Version:'. A 'List of dependencies:' section has a dropdown menu and a 'Show' button. At the bottom, a red error message reads 'The given file is not good!'.

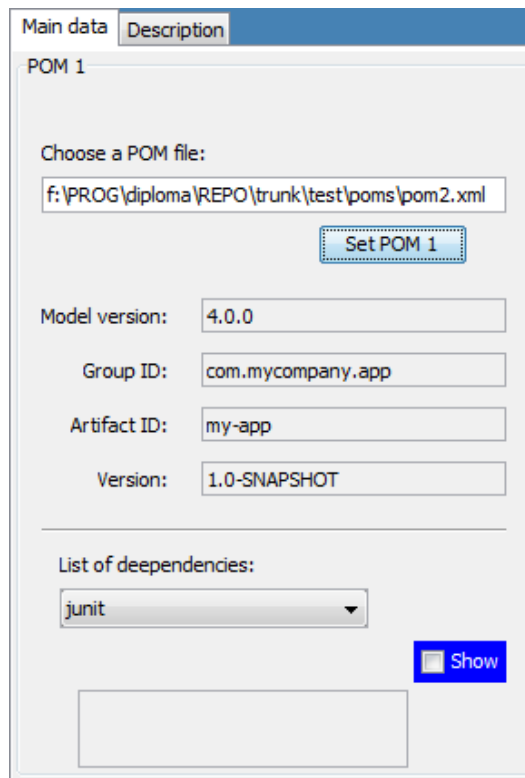
6. ábra

A szoftver számára érvényes a POM fájl, ha kielégíti az XML formátum követelményeit, illetve ha követi a Maven által definiált modellt, vagyis legalább tartalmazza a `modelVersion`, `groupId`, `artifactId`, `version` mezőket. A Maven számára egy minimális POM XML fájl struktúrája a következőképpen néz ki:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

Egy projektet igazából csak a groupId, artifactId, version hármast (GAV) azonosít pontosan.

A projekt panelen található *Set POM* gomb megnyomása után a program (amennyiben lehetséges) betölti a POM fájlt és a főbb, legfontosabb adatait megjeleníti. (7. ábra)



The screenshot shows a software interface with two tabs: 'Main data' and 'Description'. The 'Main data' tab is active, displaying the following fields and controls:

- POM 1** (Section header)
- Choose a POM file:** A text input field containing the path `f:\PROG\diploma\REPO\trunk\test\poms\pom2.xml`.
- Set POM 1** (A button with a dotted border)
- Model version:** `4.0.0`
- Group ID:** `com.mycompany.app`
- Artifact ID:** `my-app`
- Version:** `1.0-SNAPSHOT`
- List of dependencies:** A dropdown menu showing `junit`.
- Show** (A blue button)
- An empty rectangular box at the bottom of the panel.

7. ábra

Az első fülre (*Main data*) kattintva láthatjuk a POM-ot azonosító hármast, illetve a függőségeinek listáját.

A második fülre (*Description*) kattintva további információk jelennek meg (név, projekt leírása). (8. ábra)

The image shows a screenshot of the Maven IDE interface. At the top, there are two tabs: 'Main data' and 'Description'. The 'Description' tab is active. Below the tabs, there is a 'Name:' label followed by a text input field containing 'my-app'. Below that is a 'Description:' label followed by a larger text area containing the text 'This is a good project for something!'.

8. ábra

Mivel a Maven nem köti ki, a name és description mezők létezését a POM fájlban, ezért megtörténhet, hogy üresen találjuk ezen mezőket. A name mező a projekt nevét tartalmazhatja, a description mező a leírását. A projekt leírása hasznos lehet számunkra, hisz nem biztos hogy minden projektünket azonosítani tudjuk a POM fájl neve alapján, sőt az sem biztos, hogy az artifactId-ből kiderül számunkra mit csinál az adott projekt, hisz nem biztos hogy mi készítettük. Gondoljunk csak arra az esetre, amikor egy már meglévő fejlesztésbe kapcsolódunk be.

A POM megjelenítésének, függőségi vizsgálatának be és kikapcsolása

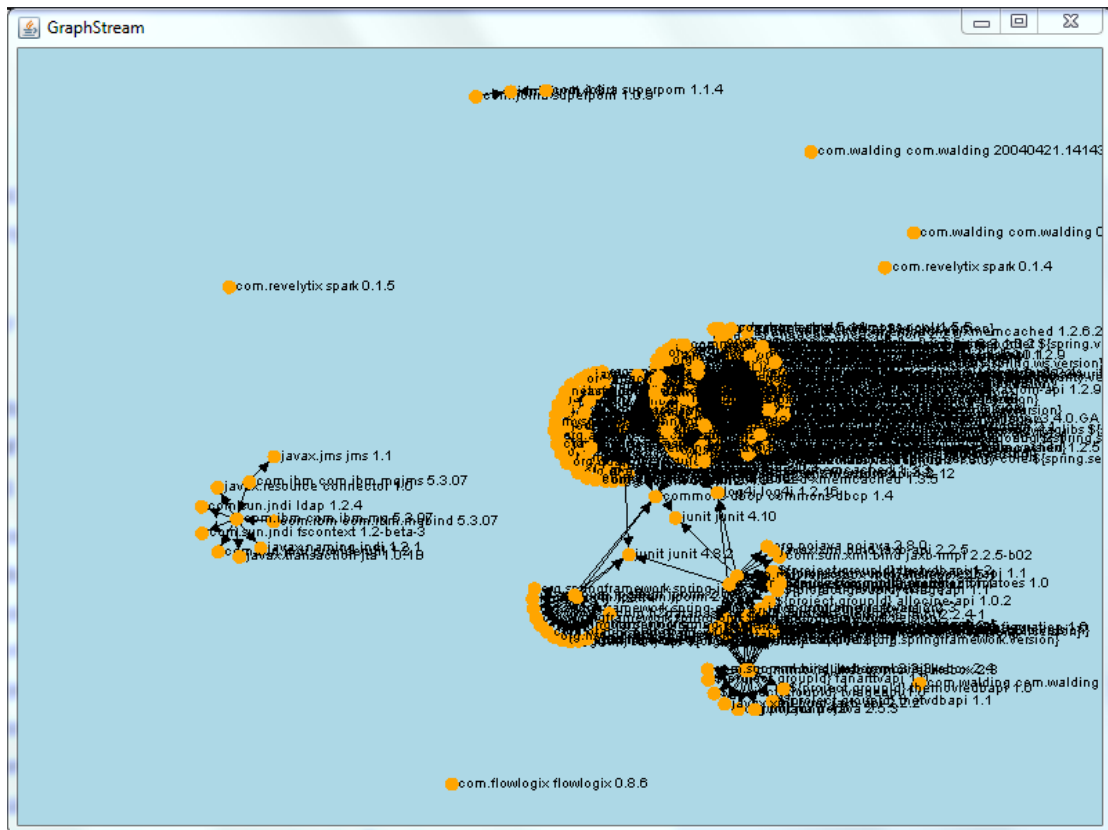
A POM fájlunk vizsgálatát, megjelenítését ki és bekapcsolhatjuk a POM paneljén található "Show" jelölőnégyzet segítségével. (9. ábra)

The image shows a screenshot of the Maven IDE POM panel. At the top, there is a label 'Dependencies:' followed by a dropdown menu. Below the dropdown menu, there is a blue button with a white checkmark and the text 'Show'.

9. ábra

A függőségi gráf megjelenítése

A program felületének közepén található Show dependencies/Refresh gombot megnyomva az aktuális adatok és beállítások figyelembevételével egy felugró ablakban kirajzolódik a függőségi gráf. (10. ábra)



10. ábra

A függőségi gráf kinézete függ:

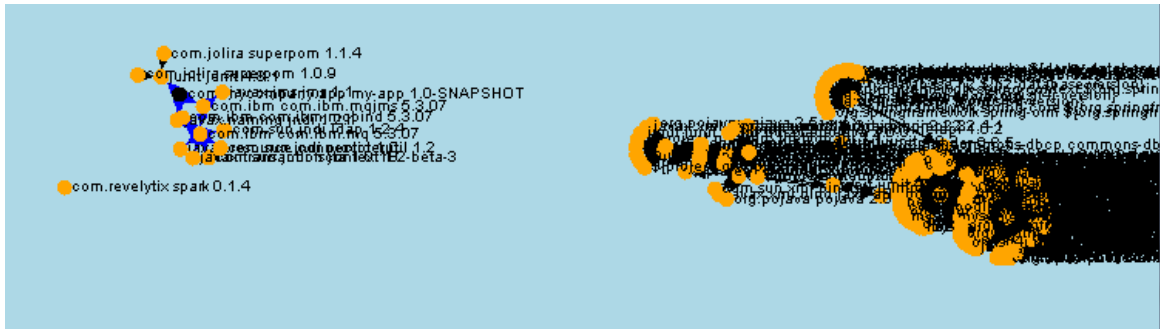
1. a Maven Central Repository-ből betöltött adatoktól
 2. az adatok által alkotott struktúrától
 3. a megadott lokális POM fájlaktól, illetve azok megjelenítésének engedélyezésétől
- a megadott projektekből gyökerező függőségi fákon kívüli függőségek megjelenítésének engedélyezésétől

A függőségi gráf vizsgálata, felhasználási ötletek

A függőségi gráf megmutatja az aktuálisan betöltött projektek közötti függőségeket, függőségi útvonalakat. Saját projekt megadása nélkül is hasznos következtetéseket vonhatunk le.

Egy projektet egy csúcs jelöl. Egy A projekt B-től való függőségét egy A-ból B-be mutató súlytalan irányított él jelöl.

Természetesen minél több adatot töltünk be, annál sűrűbb gráfot kapunk. Érdekes megfigyelni, hogy sokszor a gráf egy-egy része nagyon összesűrűsödik, illetve, hogy különálló részgráfok jelennek meg. (11. ábra)



11. ábra

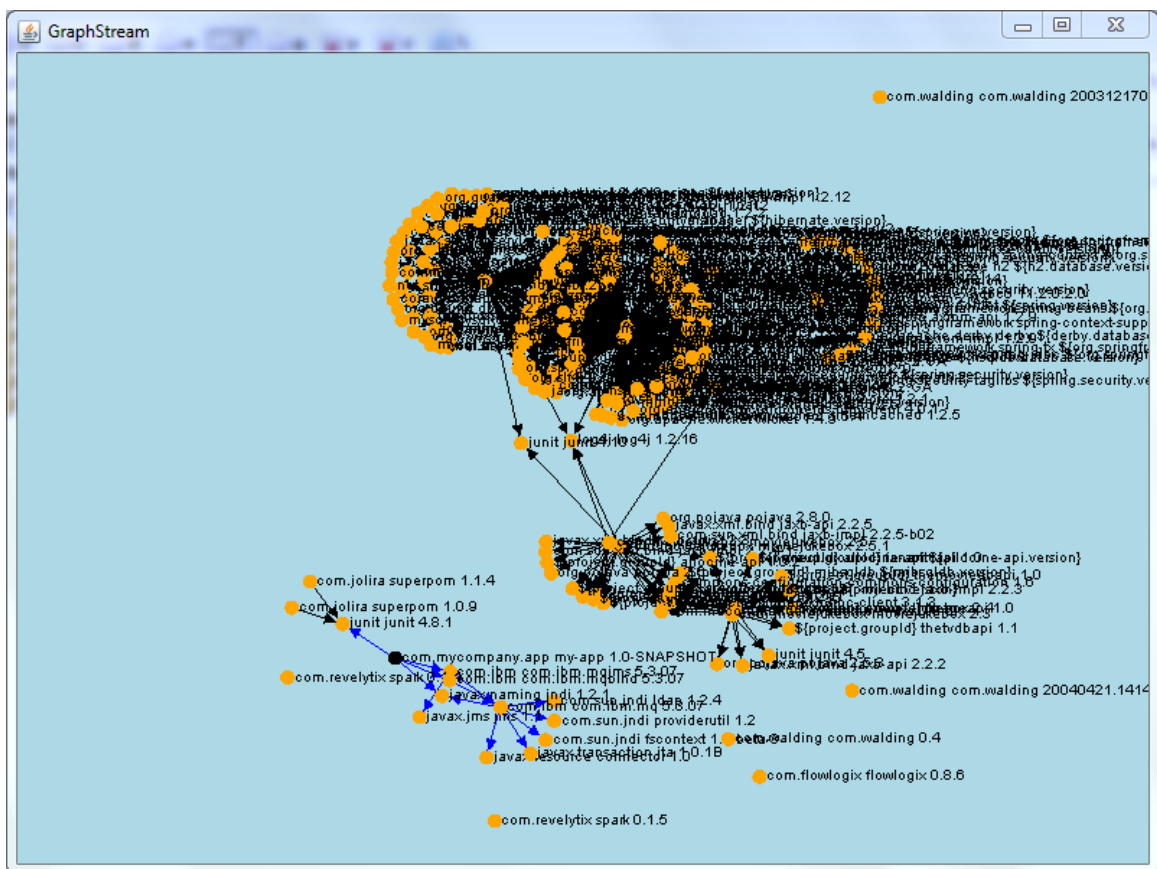
Az összesűrűsödésnek két oka van általában. Az egyik ok, hogy ezen részgráfot alkotó projektek valamilyen módon összetartoznak. Ez azért van, mert egy probléma megoldására írt hasonló programok általában hasonló módszerrel oldják meg a feladatot, hasonló harmadik fél által írt könyvtárakhoz nyúlnak. Az összetartozás másik oka lehet az, hogy egy fejlesztőcsapat saját projektjei, könyvtárai függnek egymástól. Ezek a könyvtárak ugyanúgy megjelenhetnek a központi tárolóban.

A másik ok, ami ennek valamilyenfajta következménye is, hogy vannak olyan projektek, amiknek használata sokszor nélkülözhetetlen, így rájuk sok él mutat. Érdekes megnézni, melyek azok a projektek, amelyekre a legtöbb könyvtár hivatkozik. Egy ilyen projektben bekövetkezett változás óriási mennyiségű szoftverre lehet hatással. Ezt nagyban fokozza az, hogy a legtöbb fejlesztő nem feltétlen törekszik arra, hogy visszafele kompatibilis szoftvert készítsen. Erre kiváló példa a sokak által használt *log4j* könyvtár. Folyamatos fejlesztés alatt áll, a módosítások miatt gyakran romlik el sok tőle függő program.

Most nézzük meg azt az esetet, amikor az egyik projektünket is belevesszük az elemzésbe, vagyis megjelenítjük a függőségi fában. Ilyenkor a program külön színnel jelöli a függőségi fában lévő éleket. A függőségi fa a teljes gráf egy részgráfja. A helyzet akkor válik érdekessé, amikor a projektünk egy másik

verzióját vizsgáljuk meg.

Nézzünk meg egy példát. A szoftverünk függ egy könyvtártól aminek használja a fgv() függvényét. A függvény szignatúrája a könyvtár új verziójában megváltozik, így gondba vagyunk ha mindig a legújabb verziót használjuk. (Egy POM fájlban nem kötelező megadni a használt függőség verzióját.) Az ebből adódó gondok kivédhetők részben úgy, hogy a projektünkben pontosan megadjuk a felhasznált könyvtár verzióját is és nem mindig a legújabbat használjuk.



12. ábra

A probléma akkor merül fel igazán, ha a könyvtár amitől függünk a saját függőségeiben nem definiálja pontosan melyik projekt verzióktól függ, így a legrosszabb esetben hirtelen a programunk máshogyan is működhet. Ezt gyakorlatilag akkor védhetjük ki, ha észrevesszük időközben (a projektünk új verzióiban) merrefelé ágazott el a függőségi fánk.

Egy másik éppen aktuális példa a szoftverek Strutsről Springre való átmigrálásának kérdése. Ezt ha a projektünk egy következő verziójában tervezzük, a függőségi gráf egy részgráfja teljesen megváltozik. Ezt érdemes megvizsgálni, de továbbmegyünk és gondoljunk bele, milyen függőségi gráfot láthat az aki a mi szoftverünket használja?

Ebben kíván segíteni ez a szoftver.

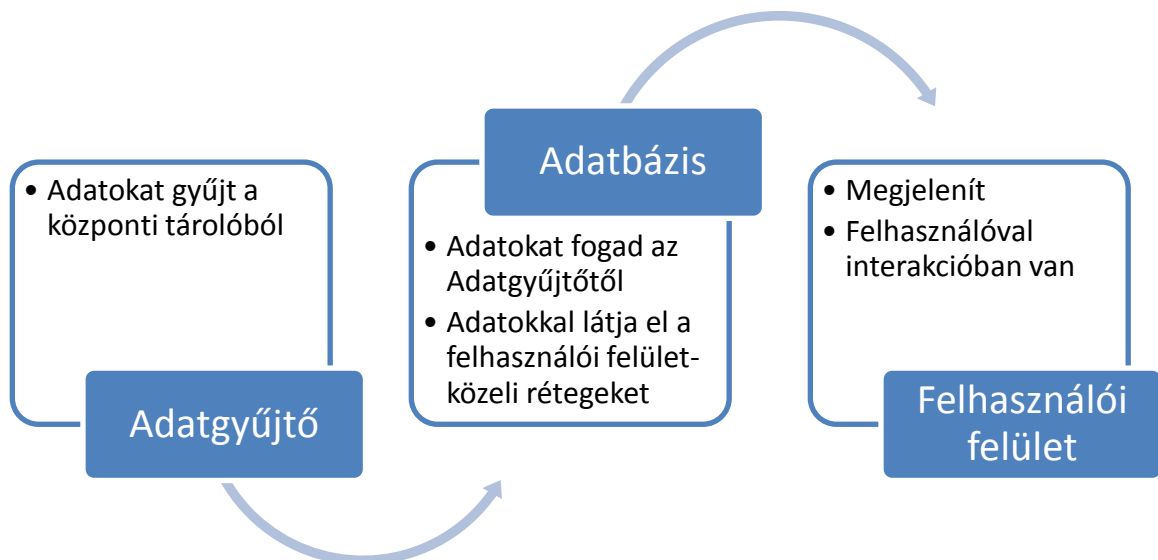
Fejlesztői dokumentáció

Általános információk a projektről

- Project oldal: <http://code.google.com/p/dependency-checker-for-java-applications/>
- Felhasználói levelező lista: dependency-checker-for-java-applications-user-group@googlegroups.com
- Issue tracking: <http://code.google.com/p/dependency-checker-for-java-applications/issues/list>
- Licenc: GNU GPL v3 (<http://www.gnu.org/licenses/gpl.html>)
- SVN: <http://code.google.com/p/dependency-checker-for-java-applications/source/>
- API: Javadoc mellékelve

Részletes specifikáció

A feladatot és megoldását három fő részre oszthatom. Az adatgyűjtés, adatbázis-kezelés illetve a felhasználói felület alkotják a programot.



Az adatgyűjtésről

Az adatgyűjtésben a feladat az, hogy elérje a program a Central repositoryt, végigböngéssze a benne található artifactokat és az adatbázis interfészen keresztül eltárolja őket. Itt kivételesen az artifactok alatt a POM fájlokat értjük, hiszen ahhoz hogy egy projekt tulajdonságait vizsgálhassuk csak és kizárólag a POM fájlra van szükségünk. A benne található groupId, artifactId és version pontosan azonosítja a projektet, illetve a függőségeinek listáját is a POM fájlból nyerhetjük ki. A POM fájlban található további adatok begyűjtése a szoftver egy továbbfejlesztésében illetve további elemzésekhez lehet hasznos.

Az adatgyűjtés nehezebb feladat mint hinnénk. Egy Maven repository saját protokollal rendelkezik, nem tudunk csak úgy körülnézni benne. Az adatgyűjtés nehézségeiről és a szoftver által alkalmazott megoldásról részletesen foglalkozom a Felhasznált módszerek fejezetben.

Adatbáziskezelés

Az adatgyűjtés folyamán a program folyamatosan elmenti az aktuálisan elolvasott POM fájlból kinyert adatokat, illetve a felhasználói felületen megjelenő adatok is az adatbázisból kerülnek elő. Az adatbázis egy a háttérben futó relációs adatbázis, aminek a kezelését egy adatbázis-kezelő osztály segítségével valósulhat meg. A szoftver saját adatbázis-kezelőjét használva biztonságosan lehet az adatbázisba új POM-ot, illetve annak függőségeit eltárolni és természetesen fordítva, POM objektumot ki is olvashatunk belőle. Fontos szempont, hogy az adatbáziskezelés során felhasznált konkrét relációs adatbázis kicserélhető legyen egy másikra, amennyiben erre szükség van, ezért a szoftver egy megvalósítandó interfészen keresztül kommunikáljon a relációs adatbázissal. Illetve amennyiben a későbbiekben még egy köztes réteget építünk be a szoftverbe, ez könnyen kivitelezhető legyen. Köztes réteg lehet például egy ORM, ahol a Java osztályainkat könnyen le tudjuk képezni relációs táblákká. Egy ilyen réteget megvalósító szoftver biztonságosabbá, stabilabbá teheti alkalmazásunkat.

A felhasználói felület

A felületen lévő panelekről, textmezők, vizsgálat, POM parseolás, gráfkirajolás, firstPOM, secondPOM.

A felhasználói felület az amit a felhasználó lát a programból. Ennek kezelhetősége fontos feladat.

Négy részre (panelre) osztható a felhasználói felület. Az elvárt működés a következő:

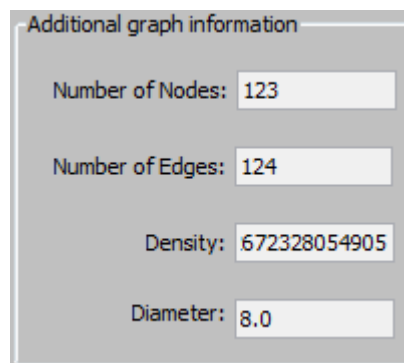
1. A felső részben található panelben állítható mit akar a felhasználó betölteni a központi repositoryból. Ennek beállítására három jelölőnégyzet (*Checkbox*) és egy szövegmező szolgál. A három jelölőnégyzet a *com*, *org* és *net* a három leggyakoribb *groupId* prefixek jelölése. A jelölőnégyzetek mellett a szövegmezőben a felhasználó megadhat alternatív prefixet is (pl. *hu*). A *Get data* gomb (*Button*) megnyomására hívódik meg az adatgyűjtő objektum letöltő függvénye amely az adatbázis objektum megfelelő függvényeit használja.

2. A baloldali és jobboldali panelben állíthat be a felhasználó egy-egy POM fájlt.

A mezőben megadott fájlt a program a *Set POM* gomb megnyomására megnyitja és XML formátumként tördelve beolvassa egy POM objektumba. Itt hibaesetként kell megvizsgálni a fájl létezését illetve érvényességét. Érvényes, ha XML formátumú és ha megtalálhatóak benne a *modelVersion*, *groupId*, *artifactId* és *version* mezők. A hibát jelzi a program a felhasználónak. Sikeres betöltés után a program megjeleníti a POM fájl főbb adatait, amiket a felhasználó nem tud szerkeszteni. A panelen található *Show* jelölőnégyzet segítségével állíthatja be a felhasználó, hogy a gráf kirajzolásánál ez a projekt jelenjen-e meg, a függőségi gráfja rajzolódjon-e ki.

3. A középső panelen található vezérlők kapcsolódnak magához a gráf megjelenítéséhez. A középső panel tetején lévő két jelölőnégyzet segítségével állíthatjuk be a gráfot milyen módon akarjuk megjeleníteni. A *Show texts* jelölőnégyzet alapértelmezetten ki van pipálva. Ilyenkor a gráf csúcsainál megjelenik az adott csúcs által jelölt POM fájl GAV adata. A *Show non-dependent POMs* jelölőnégyzet is ki van pipálva alapértelmezetten. Ilyenkor a gráf nemcsak a két megadott POM függőségi fáját mutatja meg, de a teljes függőségi gráfot megjeleníti az adatbázisban található összes POM-ot felhasználva. A két jelölőnégyzet alatt található gomb megnyomásakor kezdi el a program olvasni az

adatokat az adatbázisból. Soronként megy végig a POM-okat tartalmazó táblán és megjeleníti azokat a hozzá tartozó függőségeivel együtt, amelyeket a függőségi táblában talál meg. Szerencsére nem kell gráfbejárást alkalmazni, ahhoz hogy megtaláljuk hova kell beszúrunk. Erről bővebben írok a Felhasznált módszerekben. A középső panelen hasznos információk jelennek meg a kirajzolt gráfról. (14. ábra) Hasznos tudni hány csúcsa van a gráfnak, hány éle, mi a sűrűsége, illetve milyen hosszú a leghosszabb "egyenes út" két csúcs között (átmérő).



13. ábra

Felhasznált módszerek

Felhasznált technológiák

SVN (v1.7.1)

A **Subversion (SVN)** egy verziókezelő rendszer. Fájlok jelenlegi verzióinak és történeteinek kezelésére használják, mint például forráskódok, weboldalak és dokumentációk. Ennek a szoftvernek a verziókezelése is SVN-nel van megoldva. SVN repositoryt a **Google Code** szolgáltat.

Java (1.6 JDK)

A szoftver teljes mértékben Java programnyelven íródott. Elkészítéséhez kézenfekvő volt Java technológiákat, könyvtárakat felhasználni. Ezeket meggyünk most végig. Fontos megemlíteni, hogy minden felhasznált technológia nyílt forráskódú (open source), mint ahogyan ez a szoftver is nyílt forráskódú

licenc alatt íródik.

Maven (v3.0.3)

Bármilyen furcsa is a Maven könyvtáira gyakorlatilag nincs szükség a program használatához. Természetesen a szoftvernek csak akkor van értelme ha Maven felhasználók vagyunk és szükségünk van arra a segítségre amit ez a fejlesztés nyújt. A Mavenről már olvashattunk a bevezetőben, ezért külön nem mutatom be.

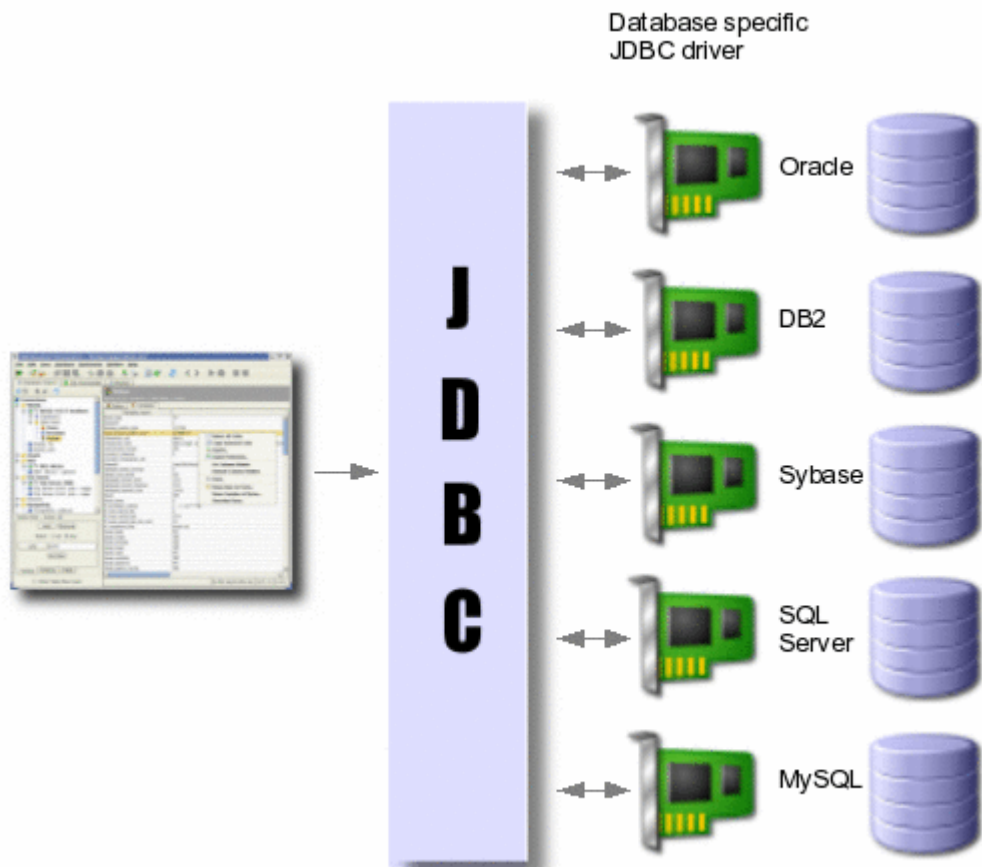
Apache Derby (v10.8.2.2)

Az Apache Derby egy beágyazott relációs adatbázis-kezelő rendszer. (angol rövidítéséből: (O)RDBMS). Licencét tekintve szabad szoftver. Sok más szabad szoftverhez hasonlóan a fejlesztést önkéntesek is végzik közösségi alapon. Ez az adatbázis-kezelő rendszer tökéletesen megfelel nekünk. Legfőbb tulajdonsága a beágyazhatósága. Képes a Java virtuális gépében futni, és a memóriát használja adattárolásra, így működése alatt gyorsabb, biztonságosabb elérést biztosít, valamint kényelmesebb használni, hisz nem kell külön telepíteni, beállítani mint más külön futó adatbázis-kezelő rendszereket. Kisméretű adatbázisok tárolásához kiváló. Működése közben a memóriát használja adattárolásra, így leállítása után az adatbázist el kell menteni külön egy állományba, illetve elindításnál az adatbázist betölti. Ez a művelet sajnos időigényes.

Ahhoz, hogy szoftverünk kommunikálhasson az adatbázis-kezelő rendszerrel szükség van egy köztes rétegre. Fontos megemlíteni, hogy ez a köztes réteg lehetővé teszi számunkra, hogy bármilyen adatbázis-kezelő rendszert használhassunk mint például a MySQL vagy a JavaDB. Ez a megoldás tesztelve is volt egy MySQL és egy PostgreSQL adatbázis-rendszerrel is, tökéletes eredménnyel.

JDBC

A szoftver és az adatbázis-kezelő rendszer közötti kommunikációt a JDBC biztosítja. A **Java Database Connectivity**, röviden **JDBC** egy API a Java programozási nyelvhez, amely az adatbázishozzáférést támogatja. A JDBC definiálja az adatbázisok lekérdezéséhez és módosításához szükséges osztályokat és metódusokat. A relációs adatmodellhez igazodik. A JDBC lehetővé teszi több implementáció létezését és használatát egy alkalmazáson belül. Az API biztosít egy mechanizmust a megfelelő java csomagok betöltésére és regisztrálására az úgynevezett *Driver Manager*-en keresztül. A *Driver Manager* az objektumorientált programozás tervezési mintái szerint egy *factory* amely adatbáziskapcsolatokat gyárt. Az adatbáziskapcsolatot a java.sql csomag



14. ábra [18]

Connection osztálya reprezentálja. Ezekkel SQL kifejezéseket lehet készíteni és futtatni.

Eclipse IDE (Indigo)

Az **Eclipse** nyílt forráskódú, platformfüggetlen szoftverkeretrendszer, amellyel úgynevezett vastag kliens (rich client) alkalmazásokat lehet készíteni. Ezt a keretrendszert eddig jellemzően integrált fejlesztőkörnyezetek (Integrated Development Environment, IDE) készítésére használták fel, mint például a Java IDE, a *Java Development Toolkit* (JDT) és compiler (ECJ) amelyet az Eclipse részeként terjesztenek (illetőleg az Eclipse fejlesztéséhez is használnak).

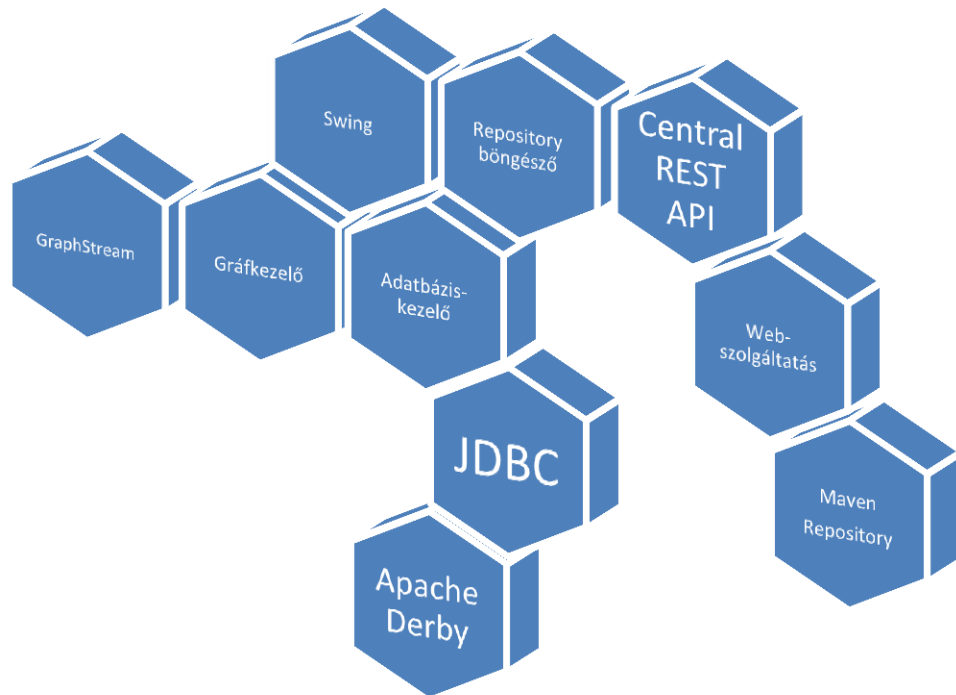
WindowBuilder

A Google által fejlesztett Window Builder Eclipse plugin segítségével készítettem el a felhasználói felületet. A Window Builder egyik nagy előnye, hogy használható generált kódot készít. Tehát a grafikus felületen összerakott felhasználói felület forráskódja nem csak megtekinthető, érthető, de módosítható is. A forráskódban lévő módosítások pedig megjelennek automatikusan a Window Builder prezentációs felületén. A másik nagy előnye, hogy egy meglévő forráskódot is értelmezni tud, meg tudja jeleníteni, illetve a fejlesztés tovább folytatható a saját grafikus felületén.

GraphStream (v1.1)

A GraphStream nyújt segítséget abban, hogy a gráf megjelenjen a felhasználói felületen. A Le Havre egyetemen fejlesztik, gyakran a magyarországi ELTE Informatika karának közreműködésével. A GraphStream egy viszonylag fiatal fejlesztés. Egyik nagy előnye, hogy nagyon gyors szoftver. Jól optimalizált, így nagyobb méretű gráfok megjelenítése és kezelése sem okoz neki problémát. Több fajta gráfmodellt támogat illetve több gráfalgoritmus megvalósítással is rendelkezik. CSS stílusok segítségével is manipulálható a megjelenése. Ez nagyon hasznos például nagyobb gráfoknál, ahol egy-egy átszínezéssel könnyebben meg tudjuk különböztetni a csúcsokat más csúcsoktól, illetve a éleket, élsorozatokat más élektől. Az alkalmazásprogramozási felülete könnyen használható, felhasználóbarát. A GraphStream SingleGraph osztályát származtatva hoztam létre saját gráf osztályomat.

A technológiák és absztrakt osztályok elhelyezkedése a program rétegeiben, struktúrájában (15. ábra)



15. ábra

Megoldási tervek, próbálkozások az adatgyűjtéshez

A tervezés során első próbálkozásnak érdemes volt megnézni, milyen meglévő szoftverek jöhetnek szóba egy Maven repository átböngészéséhez. Sajnos mint később kiderült, a legtöbb meglévő szoftver valójában külön weboldal volt saját zárt megvalósítással. Ezeken keresztül tehát nem lehetett megoldani a problémát. A megvalósítás mögött valószínűleg egy saját repository karbantartása lehet, ami egy tükrözése más repository-nak (pl. Maven Central Repository). Mi is csinálhatunk saját repositoryt, ugyanakkor letükrözni a Central Repositoryt "engedély nélkül" nem tudjuk.

Egy következő próbálkozás lehet megnézni, milyen interfészt kínálnak egyes repository megoldások. Három úgynevezett Repository Managert érdemes megemlíteni.

1. *Sonatype Nexus Maven Repository Manager*

Széleskörben alkalmazott Repository Manager. Korábban Proximity néven futott, illetve abból alakult ki. Érdekesség, hogy a Proximity projektet Cservenák Tamás magyar fejlesztő indította el.

2. *Artifactory*

Könnyen telepíthető Repository Manager, gyakorlatilag egy tömörített állományt kicsomagolva futtatható. A Maven 2-es verziójához fejlesztették.

3. *Apache Ariva*

Az Apache saját megoldása.

Ezek a repositoryk indexelhetőek. Kézenfekvő volt megnézni hogy például a Nexus Indexer hasznunkra válhat-e. Első pillantásra úgy tűnt ez lehet a megoldás, de sajnos itt is az a probléma, hogy egy Repository indexelhető ugyan, de általában lokálisan érhető el csak az adat.

A harmadik egyértelmű lehetőség a Maven klienst felhasználni a megoldáshoz. Hiszen a Maven kliens pontosan azt teszi amire nekünk szükségünk van. Kommunikál a Repositoryval, és a megfelelő adatokat letölti a lokális repo-ba, így valamilyen módon biztosan megszerezhetjük rajta keresztül az értékes információt. Egyértelműen a Maven saját API-ját érdemes először átnézni, hiszen a Javadoc segítségével dokumentált Maven Java projekt különböző osztályait így tudjuk a legkönnyebben megismerni és felhasználni. Bonyolultságát nézve így talán a legnehezebb megoldani az eredeti problémánkat. A Maven API-ja viszonylag szegényesen van dokumentálva, egy ekkora projektnél pedig nagyon sok idő lenne mire a forráskódot megértenénk.

Egy részleges megoldás lehet még a Sonatype Aether fejlesztése, amely egy gyökér POM-tól kiindulva képes felépíteni a függőségi fát. Sajnos a repository további adatainak betöltésére nem alkalmas.

Szerencsére az adatgyűjtésben segítségünkre van a Central repository saját REST API-ja.

REST API

REST API (**Representational state transfer**) egy szoftver-architektúra stílus, amely olyan hipermédia rendszerekre lett kitalálva mint amilyen a világháló. A

REST- stílusú architektúra kliensekből és szerverekből áll. A kliensek http kéréseket küldenek a szervereknek, a szerverek feldolgozzák a kérést és válaszolnak rá. Ennek az az előnye, hogy a legtöbb esetben a technológia amit használunk képes http kéréseket küldeni és válaszokat fogadni, ezért ez a köztes réteg megoldja a kommunikációt a szoftverünk és egy olyan szolgáltatás között amivel nem feltétlen tudnánk kommunikálni közvetlenül a kompatibilitás hiánya miatt.

Ezen az interfészen keresztül kérhetjük le a számunkra az értékes adatokat, így nem feltétlen szükség megismerni a Maven saját protokolljait, illetve az elszeparálódás miatt nem kell követnünk a Maven új verzióinak változásait. Ezt a gyakorlatilag webes szolgáltatást felhasználva tudtam kinyerni a számomra fontos információt.

Maven Central Repository REST API

A Maven központi tárolóját a REST API-n keresztül böngészhetjük. Gyakorlatilag a HTTP kéréseinkre válaszol. Kéttípusú formátumban képes válaszolni. Az egyik az XML formátum, a másik JSON formátum. Én a JSON formátumot választottam.

Több HTTP kérést kell kiadnunk ahhoz hogy az adatbázisba kerüljenek az adatok. Ezeket a kéréseket itt most részletesen ismertetem.

Sajnos a teljes repository betöltésétől le kell tennünk. Óriási méretű adatbázisról beszélünk, mire letöltenénk már nem beszélhetnénk friss adatokról. Elegendő tehát egy részhalmazát feldolgoznunk. A programot ezért felkészítettem arra, hogy kiválaszthassuk körülbelül mely artifactokra van szükségünk.

1. Tegyük fel, hogy minket csak a "com" groupId-val kezdődő projektek érdekelnek. Ekkor a következő HTTP kérést kell kiadnunk:

<http://search.maven.org/solrsearch/select?q=com&rows=1&wt=json>

Észrevehető, hogy a row mező 1-re van állítva. Ennek az az oka, hogy nem tudjuk még pontosan hány rekordot fogunk visszakapni a válaszban. Erre a megoldás az hogy az első kérés **numFound** mezőjét olvassuk ki.

```
... "response": { "numFound": 3854, "start": 0, "docs": ...
```

2. Ezután az előző kérést kiadhatjuk a következő módon:

```
http://search.maven.org/solrsearch/select?q=com&rows=numFound&wt=json
```

Így már visszakapjuk az összes com-os artifactot. Már csak végig kell mennünk ezen az artifact tömbön (JSON objektumok tömbje). Fontos megjegyezni, hogy a visszakapott artifactok listáján az artifactok utolsó verziója szerepel, szükségünk van tehát arra, hogy tudjuk az artifactnak milyen más verziói léteznek.

3. Erre adhatjuk ki a következő kérést:

```
http://search.maven.org/solrsearch/select?q=g:%22org.apache.maven%22%20AND%20a:%22maven-artifact%22%20AND%20v:%22%22%20&wt=json
```

Gyakorlatilag a verzió mezőt üresen hagyjuk a lekérdezésben.

Már csak egy problémát kell megoldanunk. Hisz még nem tudjuk, hogy az adott artifact milyen más artifactoktól függ. Ezt sajnos nem tudjuk egyszerűen a JSON objektumból megtudni, gyakorlatilag le kell töltenünk az adott artifact POM-ját, belenézni, és a dependency mezőket végigolvasnunk. Egy POM fájl ugyebár egy xml fájl.

1. Egy POM fájl letöltéséhez a következő kérést adjuk ki (pl: groupId: org.apache.maven, artifactId: maven-artifact, version: 3.0.1.):

```
http://search.maven.org/remotecontent?filepath=org/apache/maven/maven-artifact/3.0.1/maven-artifact-3.0.1.pom
```

A program futása során több helyen is szükség van POM XML formátumú fájlok értelmezésére, ezért erre külön osztályt írtam. Erről részletesen a Program szerkezetében olvashatunk.

Az adatbázis megvalósítása

Az adatbázis 2 táblából épül fel gyakorlatilag.

A POM tábla tartalmazza az repository artifactjait. Felépítése nagyon egyszerű.

Öt attribútumból áll: id, groupId, artifactId, version, timestamp. Ebből az első attribútum a kulcs, de lehetett volna az első négy attribútum együtt a kulcs is, mert egy projektet egyértelműen a groupId, artifactId és version határoz meg. Az id-re más táblákkal való kapcsolódás miatt van szükség, ez a projekt saját azonosítója. A timestamp segítséget nyújt abban, hogy tudjuk mikor született a projekt.

	id [PK] serial	groupid character varying(255)	artifactid character varying(255)	version character varying(255)	timestamp character varying(255)
1	19	org	jaudiotagger	2.0.3	1289997036000
2	20	org	jaudiotagger	2.0.2	1289997036000
3	21	org	jaudiotagger	2.0.1	1289997036000
4	22	org.jibx.schema.config	schema-library-parent-w3	1.0.0	1299509103000
5	23	org.jibx.schema.config	schema-library-parent-w3	1.0.0	1299508858000
6	24	org.jibx.schema.config	schema-library-parent-hr	1.0.0	1299448186000
*					

A *dependencies* tábla tartalmazza a pom-dependency egy-sok kapcsolatokat. Szintén négy attribútumból áll: id, groupId, artifactId, version. Az id itt azt jelöli melyik POM függőségéről van szó, a további hármast pedig magát a projektet jelöli amitől a POM függ.

	id integer	groupid character varying(255)	artifactid character varying(255)	version character varying(255)
1	19	junit	junit	3.8.1
2	20	junit	junit	3.8.1
3	21	junit	junit	3.8.1

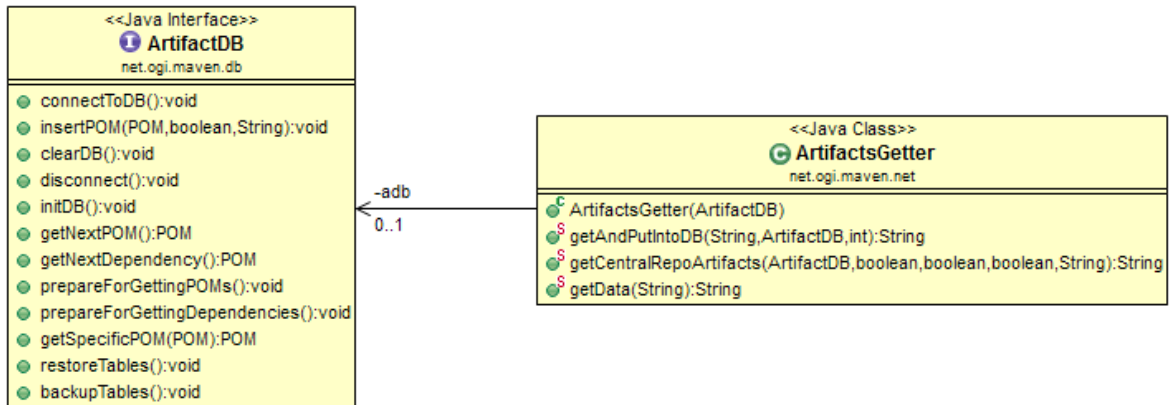
Az adatbázis-kezelő osztályról a Program szerkezeteiben olvashatunk részletesen.

A program szerkezete

A program szerkezetét az a programot alkotó osztályok jellemzésével és kapcsolatával mutatom be UML osztálydiagramon keresztül.

Adatgyűjtő réteg

Az adatgyűjtő réteget valósítja meg az *ArtifactsGetter* osztály.



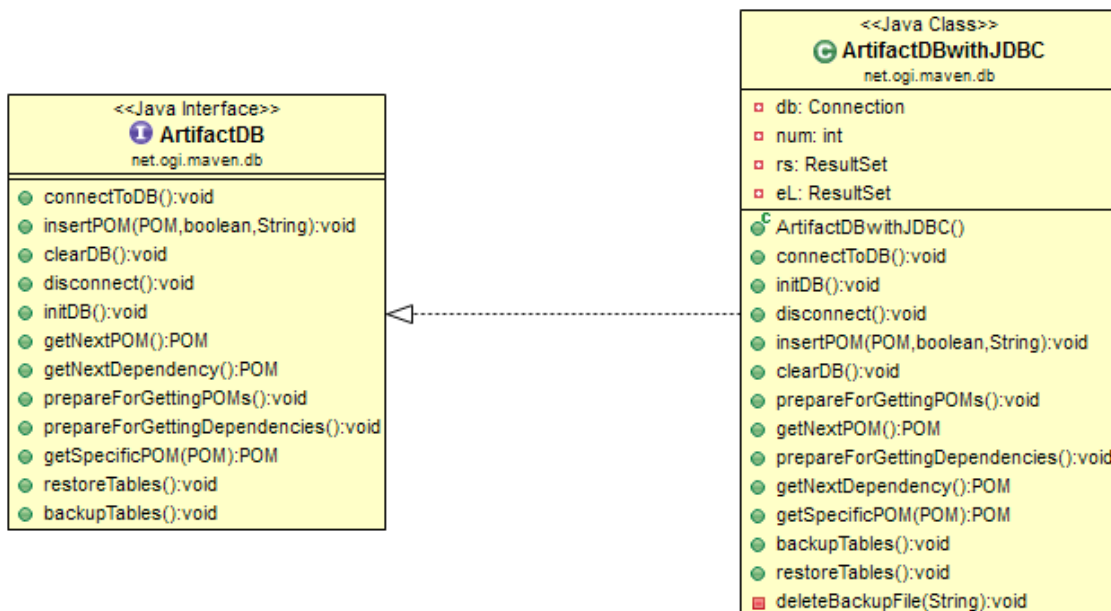
16. ábra

Működésének lényege:

- A megfelelő http kéréseket küldi ki a Maven Central Repositorynak.
- A kéréseire jött válaszok JSON formátumban érkeznek, ezeket feldolgozza és elküldi a következő kérést
- A feldolgozás alatt projekt adatokat ment le az adatbázisba. Az adatbázist az *ArtifactDB* interfészen keresztül éri el

Adatbázis-kezelés

Az adatbázis-kezeléséhez külön adatbázis osztályt hoztam létre.



17. ábra

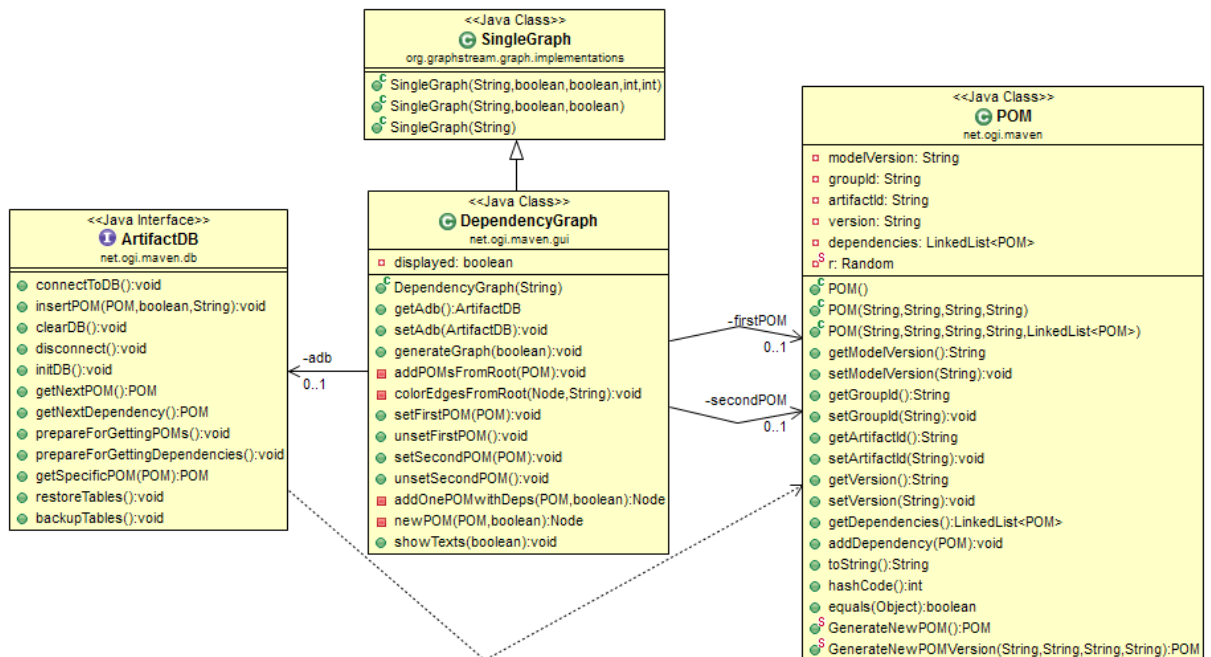
A tervezés során lényeges szempont volt, hogy az adatbázis-kezelés jól elkülöníthető legyen, mind a többi modultól mind magától az adatbázistól. Fontos, hogy az osztály ne függjön az adatbázistól, ezért a JDBC technológia segítségével köszönhetően a legtöbb adatbázissal együtt tud működni a program. Jelenleg a program az Apache Derby relációs adatbázist használja. Működésének lényege:

- Képes POM objektumokat adatbázisba menteni, az összes függőségeivel együtt.
- Képes POM objektumokat az adatbázisban keresni és azokat rendelkezésre bocsájtani.
- Képes az összes adaton végigmenni sorban és egyenként feldolgozásra bocsájtani.

Gráfkezelés

A gráf elkészítéséhez, kirajzolásának irányításához külön osztályt hoztam létre. Az osztályt a `GraphStream SingleGraph` osztályból származtattam, így az

alapvető funkciók birtokában lehettem. A gráfkezelést végző osztály neve DependencyGraph. (18. ábra)



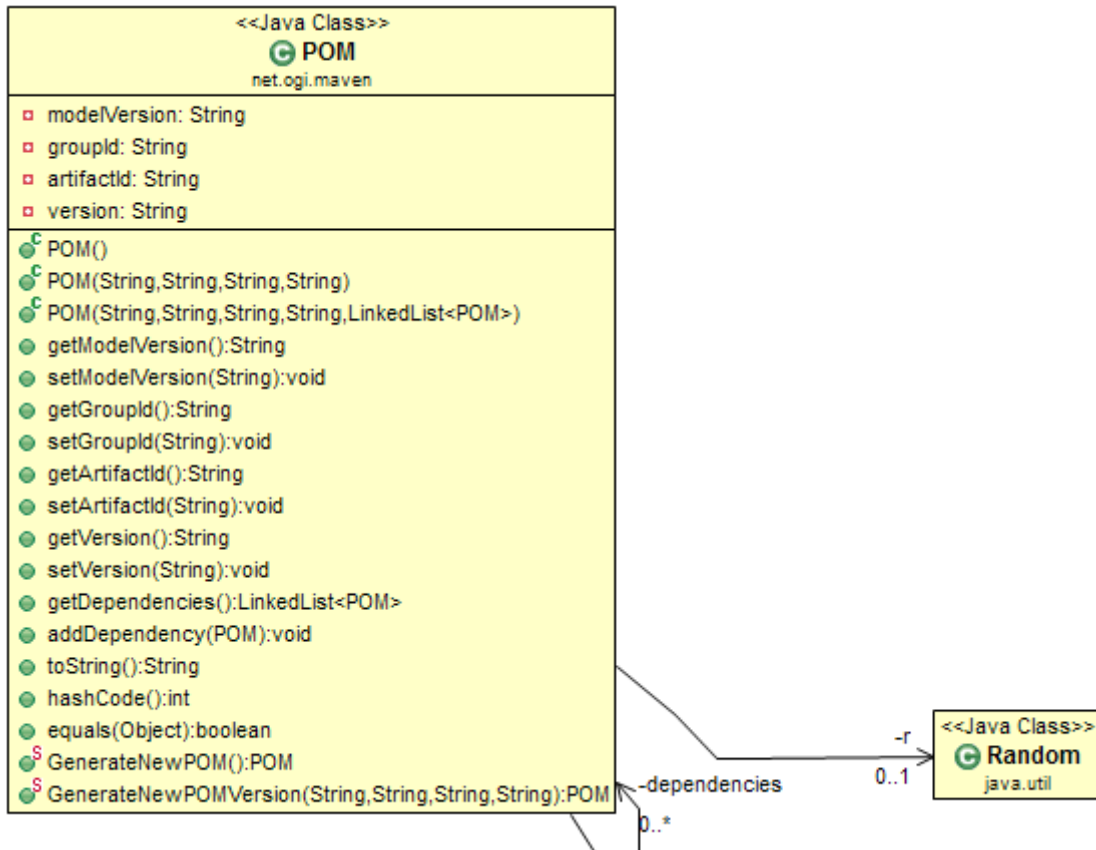
18. ábra

Működésének lényege:

- Képes a függőségi gráfot az adatbázis aktuális állapotától függően felépíteni.
- Képes a felhasználói beállítások alapján egy gyökércsúcstól induló teljes függőségi fa kirajzolására, megjelölésére.
- A feldolgozás alatt projekt adatokat olvas az adatbázisból. Az adatbázist az *ArtifactDB* interfészen keresztül éri el.

Alapvetően fontos osztályok

- Egy POM objektum tárolásához, kezeléséhez egy *POM* osztályt hoztam létre.



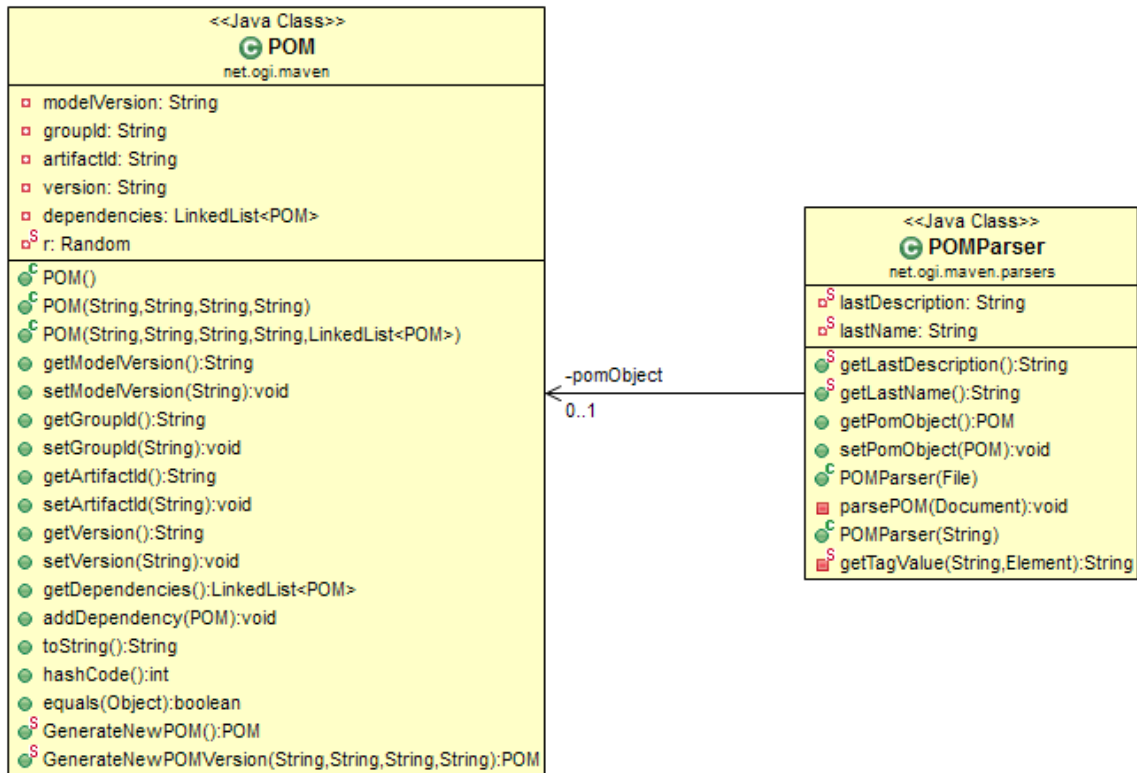
19. ábra

Működésének lényege:

- Egy POM objektum egy projekt szükségszerű adatait képes tárolni. Ezek a `groupId`, `artifactId`, `version` és a számunkra még érdekes függőségek listája.
- Képes más POM-ok véletlenszerű generálására is. Ez teszteléshez, demonstrációhoz lehet hasznos.

Mivel egy projektet leíró POM egy XML formátumú fájl, ezért szükséges volt egy olyan osztály is, amely képes egy XML formátumú POM fájlt, szöveget értelmezni és egy POM objektumban tárolni azt.

Erre való a *PomParser* osztály. (20. ábra)



20. ábra

Működésének lényege:

- Képes egy XML formátumú POM fájlt, szöveget értelmezni és egy POM objektumban tárolni azt.

Tesztelési terv és eredmények

A programot három mélységi fokban tesztelem. Funkcionális teszt illetve fontosabb függvényeknél unit tesztelést végzek.

Funkcionális tesztelés

A program funkcionális tesztelése gyakorlatilag a felhasználói felület segítségével végezhető el, hiszen a funkciókat a felhasználói felület kapcsolja össze. A teszteseteket különböző szempontok szerint csoportosítom.

Maven Repository elérhetőségéből adódó tesztesetek

Bemenet:

Kiválasztok egy prefixet (*com, org, net*) és megnyomom a *Get data* gombot.

Esetek, elvárt működés:

Maven Repo elérhető	Maven Repo nem érhető el	
A betöltés befejezte után az adatbázisra épülő modulok használhatóak.	Van biztonsági mentés Hibaüzenet amiben a program jelzi, hogy az adatokat a biztonsági mentésből tölti be. A továbbiakban az adatbázisra épülő modulok használhatóak.	Nincs biztonsági mentés Hibaüzenet amiben a program jelzi, hogy az adatokat nem tudja betölteni. A továbbiakban az adatbázisra épülő modulok nem használhatóak.

A baloldali és jobboldali panelen való POM XML fájlok megadásából adódó tesztesetek (mind a két panelre)

Bemenet:

A panel *Choose a POM file* mezőjébe szöveget írok és megnyomom a *Set POM* gombot.

Esetek, elvárt működés:

A megadott szöveg egy elérési út egy létező fájlra		A megadott szöveg nem egy elérési út egy létező fájlra	
A megadott fájl egy XML fájl		A megadott fájl nem egy XML fájl	
A megadott fájl POM fájl	A megadott fájl nem POM fájl	Hibaüzenet	
A program betölti a fájlt. A program tovább használható.	Hibaüzenet amiben a program jelzi, hogy az adatokat nem tudja betölteni a megadott fájlból.	amiben a program jelzi, hogy az adatokat nem tudja betölteni a megadott fájlból.	
		Hibaüzenet amiben a program jelzi, hogy nem tudja megnyitni a megadott fájlt.	

Gráf megjelenítése POM fájlok megadásával, megjelenítési szempontból

Bemenet:

A panelben megadott POM fájl betöltése után megnyomom a *Show/Refresh dependencies* gombot.

Esetek, elvárt működés:

A POM panelen lévő <i>Show</i> jelölőnégyzet ki van pipálva		A POM panel <i>Show</i> jelölőnégyzete nincs kipipálva	
A <i>Show non-dependent POMs</i> jelölőnégyzet ki van pipálva	A jelölőnégyzet nincs kipipálva	A <i>Show non-dependent POMs</i> jelölőnégyzet ki van pipálva	A jelölőnégyzet nincs kipipálva
Az összes letöltött adat és a POM-ból gyökerező függőségi fa is megjelenik	Csak a POM-ból gyökerező függőségi fa jelenik meg	Az összes letöltött adat és a POM-ból gyökerező függőségi fa nem jelenik meg	Egy üres gráf jelenik meg (nincs csúcs, nincs él)

Gráf megjelenítése szövegmegjelenítési szempontból

Bemenet:

Megnyomom a *Show/Refresh dependencies* gombot.

Esetek, elvárt működés:

A POM panelen lévő <i>Show texts</i> jelölőnégyzet ki van pipálva	A POM panel <i>Show texts</i> jelölőnégyzete nincs kipipálva
A megjelenő gráf összes csúcsa mellett látható sorban a csúcshoz tartozó POM GAV adata	A megjelenő gráf csúcsai mellett nincs szöveg

About felület ellenőrzésének ellenőrzése

Bemenet:

Megnyomom a menüsoron található *About* gombot.

Esetek, elvárt működés:

A megjelent <i>About</i> ablakon belül megnyomom az <i>OK</i> vagy bezáró gombot.	Az <i>About</i> ablakon belül a linkre kattintok
Bezáródik az <i>About</i> ablak	Az alapértelmezett böngészőben betöltődik a projekt oldala

Gráf megjelenítése adatmennyiségi szempontból

Bemenet:

Megnyomom a *Show/Refresh dependencies* gombot

Esetek, elvárt működés:

Kis adatbázis	Nagy adatbázis
Új ablakban megjelenik a gráf szinte azonnal.	Az adatbázis nagyságával egyenesen arányosan eltelt idő után megjelenik a gráf egy külön ablakban.

Unit tesztelés

A főbb publikus függvények unit tesztelése a projekt előrehaladtával folyamatosan történik. A unit tesztelést junit-tal végzem.

Egyelőre a *POMParser* osztályra létezik unit teszt.

- Bemenet: Egy érvényes XML formátumú POM String.
- Elvárt működés: Kimenatként az ehhez tartozó POM objektum.
- Bemenet: Egy érvényes XML formátumú POM fájl elérési útja.
- Elvárt működés: Kimenatként az ehhez tartozó POM objektum.
- Bemenet: Egy érvénytelen XML formátumú POM fájl elérési útja.
- Elvárt működés: Hiba.

Irodalomjegyzék

- [1] Sonatype, „Sonatype,” [Online]. Available:
<http://www.sonatype.com/people/2010/12/now-available-central-download-statistics-for-oss-projects/>. [Hozzáférés dátuma: 10 2011].
- [2] Apache, „Apache Maven,” [Online]. Available: <http://maven.apache.org/>.
[Hozzáférés dátuma: 09 2011].
- [3] ORACLE, „Trail: JDBC(TM) Database Access,” [Online]. Available:
<http://docs.oracle.com/javase/tutorial/jdbc/index.html>. [Hozzáférés dátuma: 08 2011].
- [4] The PostgreSQL Global Development Group, „PostgreSQL JDBC Driver,” [Online]. Available: <http://jdbc.postgresql.org/>. [Hozzáférés dátuma: 08 2011].
- [5] GraphStream Team, „GraphStream A Dynamic Graph Library,” [Online]. Available: <http://graphstream-project.org/>. [Hozzáférés dátuma: 07 2011].
- [6] Eclipse Foundation, „Eclipse,” [Online]. Available: <http://www.eclipse.org/>.
[Hozzáférés dátuma: 08 2011].
- [7] Apache, „Apache Maven,” [Online]. Available:
<http://maven.apache.org/guides/mini/guide-mirror-settings.html>. [Hozzáférés dátuma: 06 2011].
- [8] Apache, „Apache Maven,” [Online]. Available: <http://maven.apache.org/>.
[Hozzáférés dátuma: 06 2011].
- [9] SwingLabs, „SwingLabs SwingX,” [Online]. Available: <http://swingx.java.net/>.
[Hozzáférés dátuma: 08 2011].
- [10] TortoiseSVN Team, „TortoiseSVN,” [Online]. Available:
] <http://tortoisesvn.net/downloads.html>. [Hozzáférés dátuma: 06 2011].
- [11] Google, „WindowBuilder User Guide,” [Online]. Available:
] <http://code.google.com/intl/hu-HU/javadevtools/wbpro/>. [Hozzáférés dátuma: 07 2011].
- [12] jdbc-tutorial.com, „JDBC - Tutorial,” [Online]. Available: <http://www.jdbc-tutorial.com/>. [Hozzáférés dátuma: 07 2011].
- [13] Apache, „Apache Maven,” [Online]. Available:
] <http://maven.apache.org/guides/introduction/introduction-to-repositories.html>.
[Hozzáférés dátuma: 06 2011].
- [14] Apache, „Apache Maven,” [Online]. Available:
] <http://maven.apache.org/ref/3.0.3/maven-model/apidocs/>.
- [15] Wikipedia, „Wikipedia,” [Online]. Available:
] http://hu.wikipedia.org/wiki/Apache_Maven. [Hozzáférés dátuma: 05 2011].
- [16] Sonatype, „Aether Sonatype,” [Online]. Available: <http://aether.sonatype.org/>.
] [Hozzáférés dátuma: 10 2011].
- [17] Minq Software AB, „DbVisualizer,” [Online]. Available:
] <http://www.dbvis.com/products/dbvis/doc/4.2.1/doc/ug/getConnected/getConnected.html>.
- [18] L. R. Olivér, „Richard Oliver Legendi,” [Online]. Available:
] <http://people.inf.elte.hu/legendi/>. [Hozzáférés dátuma: 01 2011].

Mellékletek

- 1) 1 db olvasható CD
 - a) bin könyvtár: a futtatható állományt tartalmazza
 - b) src könyvtár: a forrásállományt és az API-t tartalmazza
 - c) libs könyvtár: a felhasznált könyvtárakat tartalmazza
 - d) test könyvtár: tartalmazza a unit teszteket és a teszteléshez használható pom fájlokat
 - e) README szöveges állomány