Eclipse alapú fejlesztés és integráció

Házi feladat dokumentáció

Hellner Gábor (XPAW4D)

Dokumentáció

Android komponens fejlesztéshez szükséges plug-in:

Az *Eclipse IDE*-ben az *ADT plug-in* segítségével lehetőség van *Android*-os alkalmazások fejlesztésére. Az *Android*-os komponensek fejlesztésének megsegítésére egy *Android* komponensmodellező *plug-int* készítettem az *Eclipse* alapú fejlesztés és integráció tárgy házi feladatának keretein belül.

Motiváció

Az *Andoid*-os alkalmazások működése egy viszonylag lazán csatolt szisztéma szerint működik (sokkal lazább kötöttségekkel, mint a *Desktop Java* világban megszokott *Objektum-Orientált* világban). Ez alatt értem az egyes komponensek közti kommunikáció és navigáció *Android* operációs rendszer által megszakított és kézben tartott működését, mely a komponensek közti laza csatolást tesz lehetővé.

Az *Android* platform 4 alap komponenst kínál a fejlesztők számára: *Activity* (képernyők megjelenítésére), Service (háttérfolyamatok futtatására), *ContentProvider* (adatréteg kiajánlására más alkalmazások számára), *BroadcastReceiver* (*broadcast* események kezelésére). Ezen kívül még számos sokszor használt komponens létezik például aszinkron feladatok végrehajtására (*AsyncTask*) vagy az újabb verziókban megjelent szintén képernyők megjelenítésére (*Fragment*-ek).

Ezen komponensek közt a kommunikáció lazán csatolt. A komponensek közti kommunikáció több esetben az *Android* beépített *Intent* szolgáltatásán keresztül működik, amin keresztül kéréseket tudunk küldeni a rendszernek, hogy milyen komponensre szeretnénk váltani vagy elindítani (Például ha egy másik képernyőre kívánunk váltani).

Házi feladat

A főbb komponensek (*Activity, Fragment, AsyncTask, BroadcastReceiver*) közti navigáció és kommunikáció létrehoztam egy szakterület-specifikus nyelvet, amely képes modellezni az *Android*-os alkalmazások komponenseinek struktúráját és az ezek közti kommunikációt. Az elkészítet modellből pedig egy beépített kódgenerátor segítségével futtatható és a fejlesztő által kiegészíthető kódot képes generálni.

A fejlesztőeszköz célja, hogy az *Android*-os alkalmazásfejlesztés komponens kezelését egyszerűbbé és könnyen karbantarthatóvá tegye. Egyaránt könnyebbséget jelenthet a fejlesztés kezdetekor egy gyorsan létrehozható modell segítségével megalkotni az alkalmazás struktúráját és az ebből a generált kód felhasználásával dolgozni, mind a fejlesztés folyamán a modellt karban tartva, a modell segítségével módosítani az alkalmazás felépítését.

A fejlesztőeszköz megvalósításában létrehoztam egy szöveges szakterület-specifikus nyelvet a komponensek modellezésére és emellett létrehoztam egy grafikus szintaxissal rendelkező nyelvet, amely integráltan képes működni a szöveges modellel, így a felhasználó tetszés szerinti szintaxissal is megadhatja a kívánt modellt. Ezen kívül pedig megoldottam a modellek sorosíthatóságát, azok hordozhatósága érdekében.

Fejlesztőeszköz:

Az elkészített fejlesztőeszköz három *Eclipse plug-inból* épül fel. Az egyik a BSc szakdolgozatom alatt készített *Xtext*-es adatmodellező *plug-in*, a másik egy szintén *Xtext*-es komponensmodellező *plug-in*, ami képes az adatmodellező eszköz által generált kódra is építeni, a harmadik pedig egy *Sirius*-os grafikus modellező *plug-in*, amely a komponensmodellező eszköz grafikus szintaxisáért felelős.

A 3 *plug-in* úgy kapcsolódik egymáshoz, hogy az adatmodellező eszköz segítségével egy szöveges szakterület-specifikus nyelvet használva modellezhető *Android*-os alkalmazások adatrétege, melyből *Android specifikus* kód generálható. A komponensmodellező eszköz segítségével *Android*-os alkalmazások komponensei modellezhetők (*Activity, Fragment, AsyncTask, BroadcastReceiver*), melyből tudunk hivatkozni az adatmodellező fejlesztőeszköz által generált adatelemekre és a megalkotott modellből képes *Android* specifikus kódot generálni. A *Sirius*-os *plug-in* pedig a komponensmodellező eszköz grafikus szintaxisáért felelős, mellyel a komponensmodelleket tudjuk grafikusan szerkeszteni.

A szakdolgozatmohoz készült adatmodellező eszköz pár kisebb módosítástól eltekintve kiegészült 2 új funkcióval: az egyik, hogy a szöveges modelleket képes **.xmi* formátumba sorosítani a hordozhatóság érdekében, a másik pedig, hogy ezekből az **.xmi* fájlokból is képes Android specifikus kódot generálni. Ugyanilyen megfontolásokból a komponensmodellező eszközbe is beépítettem ezeket a funkciókat.

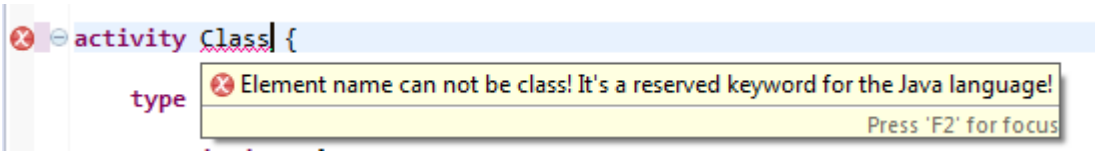
Részletek:

A komponensmodellező eszköz az *Xtext* beépített támogatása segítségével képes *Java* típusokra is hivatkozni, melyek rajta vannak a *classpath*-on, így ennek segítségével megoldható volt, hogy a komponensmodellből hivatkozzunk a generált adatréteg osztályaira. Ehhez csak annyi kellett, hogy a modellező projekt *classpath*-án rajta legyen a célprojekt, ahova a kódot generáltuk. Ezt a *plug-in* automatikusan megteszi a konfigurációs *(*.conf)* fájl alapján, annak minden módosításakor.

```
associations {
    One hu.bme.aut.mercury.mercuryclientandroid.entities.Student "student"
}
```

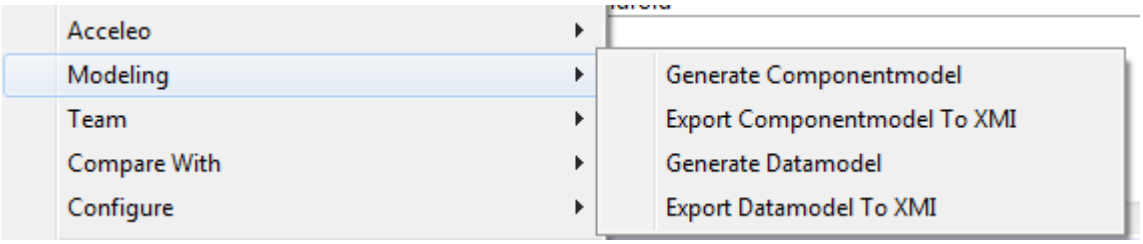
1. ábra - Komponensmodellező eszköz - hivatkozás Java típusra

A megadott szöveges szintaxist használva az elkészített modellből *Android* specifikus kód generálható. Ehhez szükséges egy konfigurációs fájl *(*.conf)*, amiben meg kell adni a célprojektet és a cél *packgae*-et. A szöveges fájl szerkesztése során az eszközfolyamatosan ellenőrzi a modell helyességét és ennek megfelelően visszajelzéseket ad a felhasználónak a modell helyességét illetően. Ez lehet súlyos hiba (*Error*), mely esetén a kódgenerálás nem is indul el és kisebb figyelmeztetés (*Warning*), mely kisebb esetleges problémákat jelezhet.



2. ábra - Komponensmodellező eszköz - hiba jelzések

A kódgenerálás elindításához a projekt kontextus menüjéből a *Modeling* fül alatt ki kell választani a megfelelő opciót (*Generate Componentmodel*), szintén ebből a menüből indítható el az **.xmi* fájlba történő exportálás. A kiexportált fájlra kattintva szintén a kontextus menü *Modeling* füle alatt érhető el a kódgenerálás. A kódgenerálás és az exportálás lépéseiről és az esetleges hibákról egy konzol ad értesítést a generálás során. A szöveges forrásfájlok rendezetten tartásához a szöveges fájlok formázhatók is a könnyebb átláthatóság érdekében.

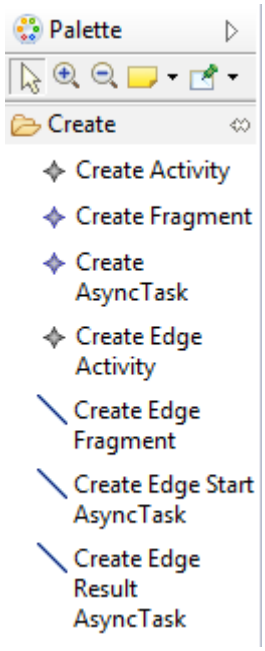


3. ábra - Kódgenerálás és exportálás indítása

```
------------------------ STARTING GENERATION ------------------------

INFO: Generating MercuryClientAndroidApplication.java
INFO: Generating FragmentViewActivityG.java
INFO: Generating LoginActivityG.java
INFO: Generating MainMenuActivityG.java
INFO: Generating PersonalDataFragmentG.java
INFO: Generating TrainingFragmentG.java
INFO: Generating HttpAsyncTask.java

2015.01.26 - 21.27.44 - GENERATION PROCESS ENDED SUCCESSFULLY
```

4. ábra – Komponensmodell konzol

Az elkészített szöveges modell fájlok és a kiexportált modellek mind felhasználhatók a grafikus megjelenítés és szerkesztés eléréséhez a *Sirius* segítségével. A modellek megnyithatók egy diagram reprezentációba, melyen keresztül grafikusan szerkeszthetők és egy paletta segítségével adhatók hozzá újabb modellelemek. A változtatások mentése, automatikusan érvényre jut a szöveges és **.xmi* fájlokban. Ilyen grafikus összekötetés esetén ez fordítva is igaz, a szöveges és **.xmi* fájlok módosítása a grafikus modellben is automatikusan megjelenik.



5. ábra - Grafikus szerkesztő paletta

Használat:

Android-os alkalmazások készítésekor igen bonyolult és időigényes lehet a komponensek lefejlesztése és a köztük lévő navigáció, kommunikáció és adatáramlás megadása, majd pedig annak karbantartása, mely egyszerűen modellezhető, mégis sok kódolást igényel. A modellben például megadhatók az alkalmazásban használatos *Activity*-k és *Fragment*-ek és ezek között a logika, hogy melyikből melyiket tudjuk indítani, továbbá pedig megadható, hogy az egyes komponensek adattagjai közül melyek kerüljenek átadásra az egyes komponensek között.

Ez jelentősen növelheti a fejlesztés gyorsaságát, hiszen például egy *Activity*-k közti adategység átadásánál nem kell kézzel implementálni az adategységet, azon belül *Android* specifikusan a *Parcelable interface* metódusait és nem kell törődni az *Activity*-k közti adatok átadásával, hogy *Intent*-be csomagoljuk ezeket, mert ezt a generált kód elvégzi és a fejlesztő számára ez észrevétlenül megtörténik.

Egy másik terület, ahol sokat segít a modellező eszköz az *AsyncTask*-ok indítása és feliratkozás ezek eredményére. Az *Activity*-knél megadható, hogy indítson egy *AsyncTask*-ot, illetve, hogy iratkozzon fel az eredményére. Ebből pedig a kódgenerátor legenerálja a teljes *AsyncTask*-ot, az *Activity*-ben az *AsyncTask*-ot indító függvényt és az *Activity*-ben létrehoz egy *BroadcastReceiver*-t, amivel feliratkozik az *AsyncTask* eredményére.

Az *Activity*-khez továbbá rendelhetők *Fragment*-ek is, ezeket meg is tudjuk jeleníteni az *Activity* indításához felvett annotációval. Ezzel az *Android* komponensei közti navigáció és kommunikáció gyakran használt *use-case*-eit lefedi a modellező eszköz, így használatával mind a fejlesztés elején, mind közben gyors prototípusfejlesztést tesz lehetővé.

A modell és a generált kód jól karbantartható, hiszen a generált kód elkülönül a kézzel szerkesztettől. A generált kód testre szabása az osztályból történő leszármaztatással lehetséges a *Generation Gap* minta alapján. Így a modell változtatásával és újragenerálásával teljesen biztonságosan lehet az alkalmazást fejleszteni olyan szempontból, hogy a kézzel írt kód sose íródik felül.

Példák:

A szöveges modellező nyelv által leírt *Activity*:

```
URI login_activity

activity LoginActivityG {

    type Simple

    associations {
        One hu.bme.aut.mercury.mercuryclientandroid.entities.Student "student"
    }

    start_activities {

        @SendEntity { student }
        main_menu_activity.MainMenuActivityG

    }

    start_async_tasks {
        http.http
    }

    result_async_tasks {
        http.http
    }

}
```

6. ábra - Login Activity szöveges modell

A fent bemutatott példában egy *Activity* leírása található, melyben látható, hogy megadjuk az *Activity* típusát, egy referenciát a hallgató adattagot reprezentáló *Java* típusra a cél projektből, megadjuk, hogy mely másik *Activity*-t szeretnénk indítani és hozzá, hogy a hallgatót át akarjuk adni a másik *Activity*-nek. Továbbá pedig megadjuk, hogy szeretnénk innen egy *HTTP AsyncTask*-ot indítani, aminek az eredményére ebben az *Activity*-ben várunk. Ez azért szükséges, mert ez aszinkron hajtódik végre és egy *callback* függvény segítségével tér vissza.

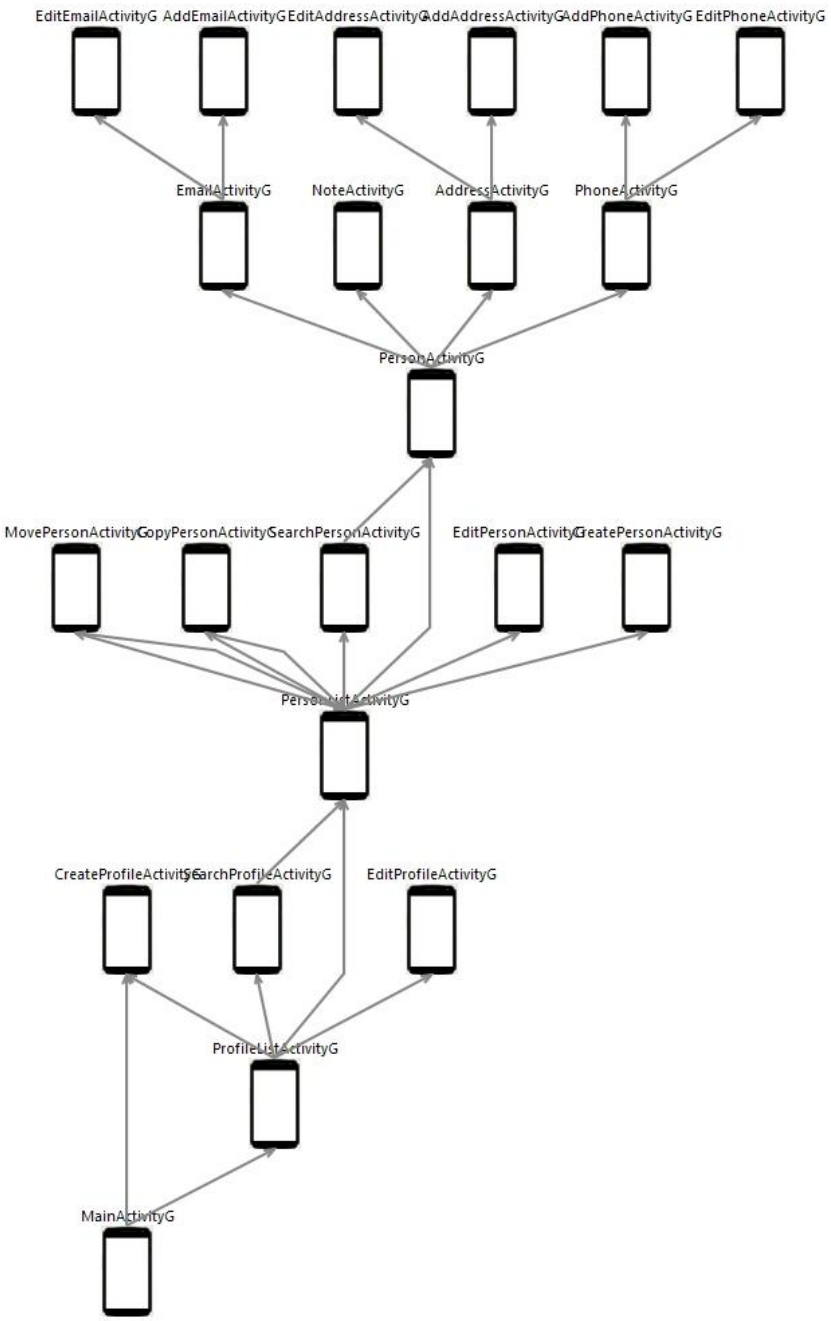
A szöveges modellező nyelv által leírt *Activity*:

```
URI main_menu_activity

import student.*
import fragment_view_activity.*
import personal_data_fragment.*
import training_fragment.*

activity MainMenuActivityG {

    type Simple

    associations {
        One hu.bme.aut.mercury.mercuryclientandroid.entities.Student "student"
    }

    start_activities {

        @SendEntity { student }
        @ShowFragment ( PersonalDataFragmentG )
        FragmentViewActivityG

        @SendEntity { student }
        @ShowFragment ( TrainingFragmentG )
        FragmentViewActivityG

    }

}
```

7. ábra – MainMenu Activity szöveges modell

A fent bemutatott példában szintén egy *Activity*-t adunk meg, ahol az indított *Activity*-knek egy-egy *Fragment*-jét akarjuk megjeleníteni és továbbítani akarjuk mindkét eseten a hallgatót az *Activity*-nek. Ezt a fent bemutatott módon lehet megadni.

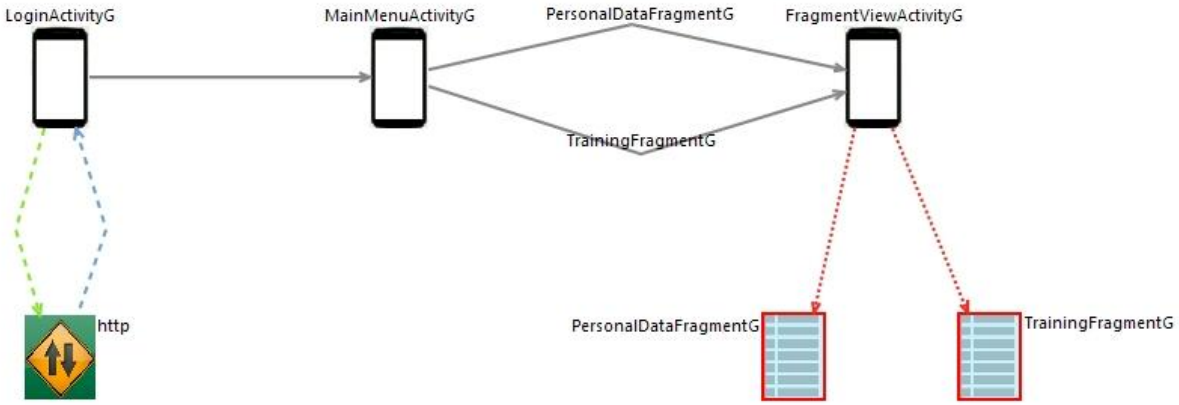
A grafikus modellező nyelv által megadott modell:



8. ábra - A Contact Book alkalmazás komponenseinek grafikus modellje

A fent bemutatott példában látható a *Contact Book* alkalmazás komponenseinek és a köztük történő navigáció grafikus modellje. Az alkalmazás viszonylag sok *Activity*-ből áll és a köztük lévő navigáció is bonyolultabb a megszokottnál. A modellt karbantartva viszont ez jól kezelhető.

A grafikus modellező nyelv által megadott modell:



9. ábra – A Mercury Client Android alkalmazás komponenseinek grafikus modellje

A fent bemutatott példában látható a *Mercury Client Android* alkalmazás komponenseinek grafikus modellje. Az alkalmazás három *Activity*-ből áll. Az egyik ezek közül egy *HTTP*-n keresztüli authentikációt végez egy *AsyncTask* segítségével, melynek az eredményére fel is iratkozik. A *MainMenuActivity* pedig a harmadik *Activity*-t indítja két különböző *Fragment*-re navigálással.