Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozáselmélet és Szoftvertechnológiai Tanszék

Ekvivalens Python forráskód-párok generálása

Témavezető:
Szalontai Balázs
doktorandusz

Szerző:
Verebics Peter
programtervező informatikus BSc

Budapest, 2024

Tartalomjegyzék

1. fejezet

Bevezetés

Szakdolgozatom témája Python forráskódok átalakítása és ezen átalakítások szemléltetése. A motiváció az átalakítások mögött egy olyan adathalmaz generálása amiben ekvivalens és nem ekvivalens forráskód-párok egyaránt szerepelnek. Egy ilyen adathalmazt felhasználhatunk egy mélytanuló neuronháló tanítására, ami forráskód-párok ekvivalenciáját dönti el.

Az ekvivalencia eldöntése fontos feladat, mivel egyre több, kódokat gépi tanulással refaktoráló, eszköz létezik. Ezek az eszközök egy kódot változtatva sokszor a kód jelentését is megváltoztatják. Egy ekivalenciát eldöntő neuronháló képes lenne kiszűrni az ilyen eszközök által generált rossz eredményeket, javítva az eszközök hatékonyságán.

Tehát ekvivalens és nem ekvivalens kódokat generálva felépíthetünk egy adathalmazt, ami ekvivalenciával felcímkézett kódpárokat tartalmaz, és alkalmas egy fent leírt neuronháló tanítására.

Az általam implementált átalakítások absztrakt szintaxisfák (AST-k) módosításával működnek. Egy forráskód fordítása alatt a szemantikus elemző előállítja a kód AST-jét, ami a kódot egy fa adatstruktúrával reprezentálja. Az AST-nek a szemantikus elemzésben van szerepe, de használhatjuk kódok átalakítására is, mivel vissza lehet alakítáni forráskóddá.

Az általam megvalósított átalakítások a Python *ast* modulját használják, ami része a Python standard könyvtárának. Az *ast* modul lehetőséget biztosít egy Python kód AST-vé és AST kóddá alakítására is.

Az átalakításokat szabályok végzik. Átalakításkor a Python kódból létrehozott AST-n végrehajthatunk egy szabályt. A szabály megváltoztatja az AST-t, amit ha visszaalakítunk kóddá egy megváltozott Python kódot kapunk.

A szakdolgozatomban ekvivalens és nem ekvivalens szabályokat is definiálok. Egy szabály akkor tekinthető ekvivalensnek, ha a kód szemantikáját nem változtatja meg. Például a Python-ban is teljesül a valós számok körében a szorzás kommutatív tulajdonsága. Tehát ha egy Python kódban két szám szorzásánál a bal és jobb operandust megcseréljük, akkor a szorzás eredménye nem változik, vagyis ez az átalakítás ekvivalens. Ez a példa természetesen nagyon egyszerű, a szakdolgozatomban összetettebb átalakításokra is adok példát.

Az adathalmazban az ekvivalens kódok generálásához saját szabályok mellett, a *ruff* Python linter és formatter szabályait is felhasználtam. A *ruff* már létező Python lintereket implementál Rust programozási nyelven, így sok más Python refaktoráló eszköz szabályait is képes elvégezni, amik tökéletesek az általam implementált szabályok kiegészítésére.

A szakdolgozatom következő fejezeteiben az adathalmaz generálására és az átalakítások szemléltetésére alkalmas szoftver használatát és működését részletezem.

2. fejezet

Felhasználói dokumentáció

A szoftver két felhasználói felülettel rendelkezik. Az egyik egy parancssoros (CLI) program az adathalmaz generálásához, a másik egy grafikus (GUI) alkalmazás az átalakítások szemléltetéséhez és az adathalmaz böngészéséhez. Mindkét alkalmazás felületének nyelve angol.

2.1. Futtatási környezet

A szoftver egy Python 3.10-es vagy újabb verziójú Python interpreterrel futtatható. Futtatás előtt a szoftver a függőségeit installálni kell a *pip* csomagkezelővel. Ezt a legegyszerűbben a szoftver forráskódjának könyvtárból tehetjük, a következő parancs kiadásával:

```
1  $ pip install --editable .
```

2.2. Adatbázis beállítása

Az adathalmaz generálásához szüksége van egy *mongodb* adatbázis kliensre [1]. Az adatbázis kiszűri a kódpárok generálása közben a duplikált kódokat és az adatok lekérdezését is megkönnyíti. Azért döntöttem a *mongodb* mellett, mert az SQL adatbázisoknál sokkal flexibilisebb.

Az adatbázis elérést a forrás könyvtárában a *config/default.ini* útvonal alatt található konfigfájlban lehet beállítani. Egy adatbázist három paraméter határoz meg: *host*, *port*, *database* (az adatbázis neve). Ha szükséges a konfigfájl alapértékeit átírhatjuk.

2.3. Adathalmazt generáló CLI

Ezzel a CLI alkalmazással van lehetőségünk az adathalmaz generálására egy adott csv fájlban található kódokból vagy egy könyvtárban taláható forrásfájlokból. Adathalmazt a következő paranccsal generálhatunk:

```
$ python -m source.persistor <mode> <path>
```

A parancs paraméterei a következők:

1. *mode* - az adatok forrásának típusa, lehetséges értékek:
   - *csv* - csv fájlból olvassa a forrásfájlok tartalmát
   - *dir* - könyvtárból rekurzívan olvassa a forrásfájlokat
2. *path* - az adatok forrásának elérési útvonala

Ha megadtuk a parancsot a program megpróbálja a forráskódok olvasását, ha az input nem megfelelő leáll.

Futás közben a program a kiírja az éppen feldolgozott forráskóddal kapcsolatos információkat, például a kódon végzett átalakítások számát és azt, hogy el tudta-e menteni az átalakítások eredményeit.

```
reading csv, be patient this might take a while...
-------------------------------------------------------------------------------------------------[0]
No changes to save.
-------------------------------------------------------------------------------------------------[1]
Fixed 13 errors.
1 file reformatted
1 file reformatted
1 file reformatted
Inserted 3 changes on hash f3842737e0fe9141df61f299c99e65d9b20a41ae3c76644ef663483aec8aeb31.
-------------------------------------------------------------------------------------------------[2]
Fixed 12 errors.
1 file reformatted
1 file reformatted
1 file reformatted
Inserted 3 changes on hash bdd72cdcf458519e9e88afa499c0e6363e427fa7ea8c9fe5484a426babed40c1.
```

2.1. ábra. A CLI alkalmazás futás közben

Ha a program végigolvasta a csv fájlt vagy a könyvtárban található forrásfájlokat leáll. Miután a program leállt a forráskód-párok már az adatbázisban vannak. A *mongoexport* eszköz segítségével az adatbázisból a forráskód-párokat egy csv fájlba exportálhatjuk.

2.4. Átalakításokat szemléltető GUI

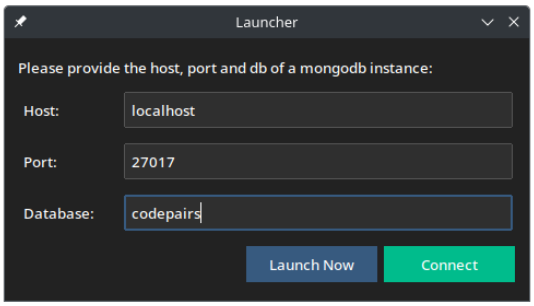
Ez a GUI alkamazás szemlélteti az átalakításokat. Kipróbálhatunk vele egy vagy több átalakító szabályt, vizualizálhatjuk kódok absztrakt szintaxis fáját, és az adatbázisba bekerült átalakítások eredményét is megnézhetjük.

2.4.1. Alkalmazás indítása

Az alkalmazás a forrás könyvtárából indítható a következő paranccsal:

```
1  $ python -m source
```

A parancs kiadása után felugró ablakban beállíhatjuk az adatbázis kapcsolathoz szükséges paramétereket: a *host*, *port* illetve *database* értékeit. A *Connect* gombra kattintva az alkalmazás adatbáziseléréssel indul, ha a megadott adatbázishoz 10 másodperc alatt kapcsolódni tud, különben adatbáziselérés nélkül. Adatbáziselérés nélkül a *Launch Now* gombbal indíthatjuk az alkalmazást.



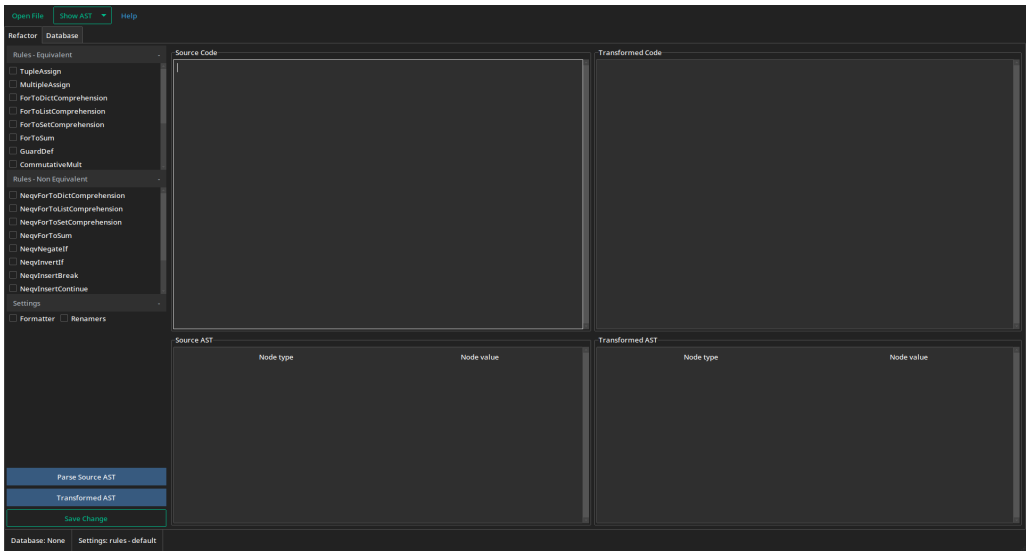
2.2. ábra. Az alkalmazást indító ablak

2.4.2. Alkalmazás felülete

Az alkalmazás felülete funkciók szerint két fő nézetre osztható, a refaktoráló és az adatbázis-böngésző nézetre. A refaktoráló nézetben (*Refactor* tab) egy kódon ekvivalensen és nem ekvivalensen átalakító szabályokat próbálhatunk ki, és elmenthetjük a szabályok által végzett átalakítások eredményeit. Az adatbázis-böngésző (*Database* tab) nézetben az adatbázisba bekerült kódpárokat tekinthetjük meg. A fájlok olvasásáért, az AST-k gráfos ábrázolásáért és a segítségért felelő gombok, illetve a beállításokat mutató állapotsor a két fő nézeten kívül helyezkednek el.

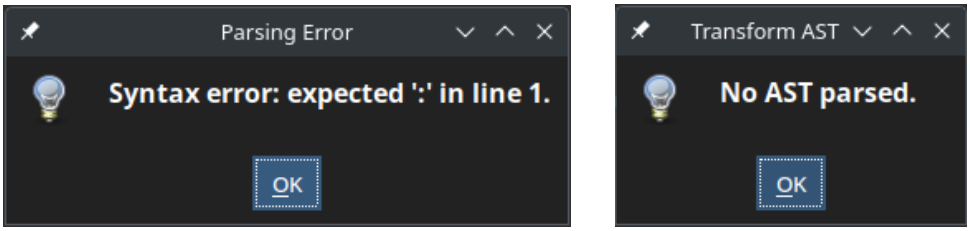
2.4.3. Refaktoráló nézet

Indítás után a felhasználót a refaktoráló nézet fogadja. Egy Python forráskód átalakításához a kódot begépelhetjük a *Source Code* szöveges input mezőbe, vagy egy *.py* fájlból is betölthetjük a menüben látható *Open File* gombra kattintva.



2.3. ábra. Refaktoráló nézet az indítás után

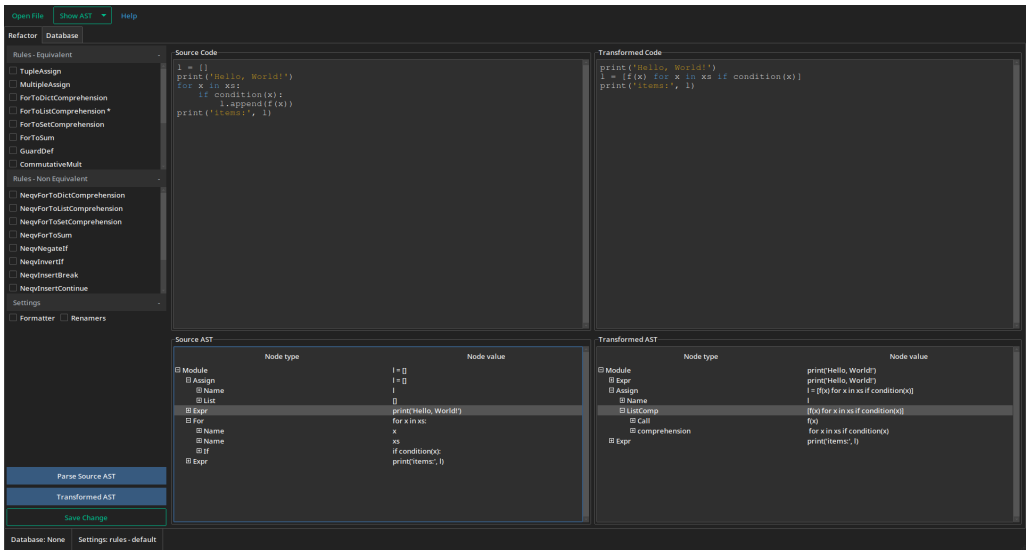
Az átalakítás előtt a begépelt vagy betöltött forráskódból az elemező (parser) futtatásával létre kell hozni egy AST-t, ezt a *Parse Source AST* gombbal tehetjük meg. Ha a megadott forráskódban szintaxis hiba található, vagy valami egyéb okból kifolyólag nem elemezhető, akkor az alkalmazás ezt jelzi egy felugró párbeszéd-ablakkal. Akkor is jelez ha parse-olás nélkül klikkelünk az átalakító gombra.



(a) elemzési hiba esetén (b) hiányzó AST esetén

2.4. ábra. Hibákat jelző párbeszéd-ablakok

Sikeres elemzés után az AST felépítését a *Source AST* fa-nézeten láthatjuk, az első oszlopban a csúcs típusa, a második oszlopban a csúcs szintaxis fájából generált kód látható. Elemzés után a fát átalakíthatjuk a *Transform AST* gombra kattintva, ekkor az átalakított fa megjelenik a bal oldali *Transformed AST* fa-nézeten, az átalakított fából generált kód pedig a *Transformed Code* readonly szövegdobozban.



2.5. ábra. Példa egy átalakítás eredményére

Az alkalmazásba összesen 28 átalakító szabály van, ezek közül 16 ekvivalens és 12 nem ekvivalens eredményt állít elő. Az alkalmazás indításakor az összes ekvivalens szabály ki van választva, ez az alkalmazás alapbeállítása amit a *'rules - default'* felirat jelez az állapotsorban.

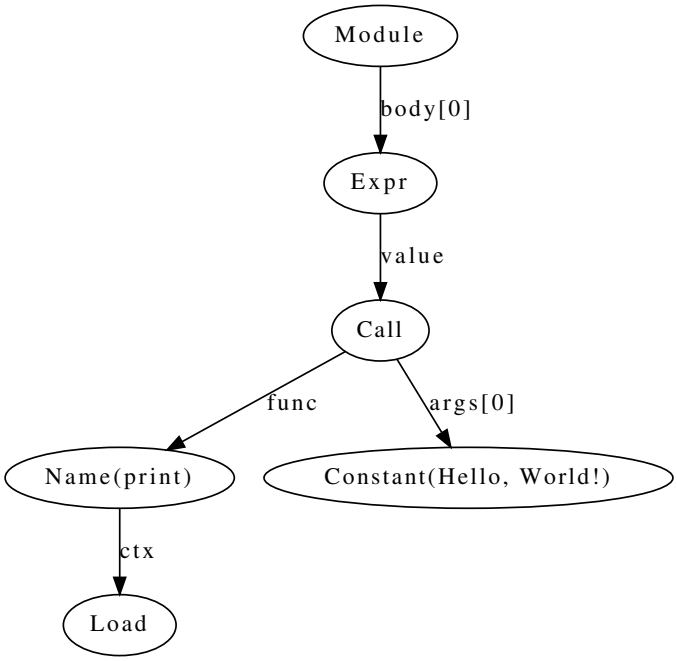
Lehetőségünk van általunk választott szabályok alkalmazására is. A szabályok listája a bal oldali panelen látható. Minden szabály előtt van egy checkbox amivel a szabályt kiválaszthatjuk. Lehetőségünk van egy vagy több szabály kiválasztására is, így könnyen tesztelhetjük egy szabály működését is. Ha vannak kiválasztott szabályok azt a *'rules - custom'* felirat jelzi az állapotsorban.

Átalakításkor a szabályok a bal oldali panelen látható sorrendben, fentről lefele kerülnek végrehajtásra. A panelen a szabályokon kívül található még két checkbox is, ezekkel a *ruff* formatter és az átnevező szabályok alkalmazását tudjuk beállítani.

Miután átalakítottunk egy forráskódot kimenthetjük az átalakítás eredményét az adatbázisba (ha van adatbázis kapcsolat). Ezt a refaktoráló nézet bal alsó sarkában elhelyezkedő *'Save Change'* gombbal tehetjük meg. Ha az átalakítást nem lehet elmenteni azt az alkalmazás párbeszéd-ablakban jelzi.

2.4.4. AST-k vizualizálása fagráffal

Az alkalmazás fagráfként is tud AST-ket vizualizálni. Az általunk megadott vagy átalakított kód AST-jének fagráfját a menüben látható *Show AST* lenyíló menügombbal vizualizálhatjuk. A gombra klikkelve két opció közül választhatunk: a *Source* gomb az általunk megadott kód, a *Transformed* gomb pedig az átalakított kód AST-jét vizualizálja, ha ezek léteznek. Az alkalmazás az elkészült fagráf ábráját egy felugró ablakban nyitja meg. Az alábbi ábrán például a *helloworld* Python kódjának AST-jét láthatjuk:

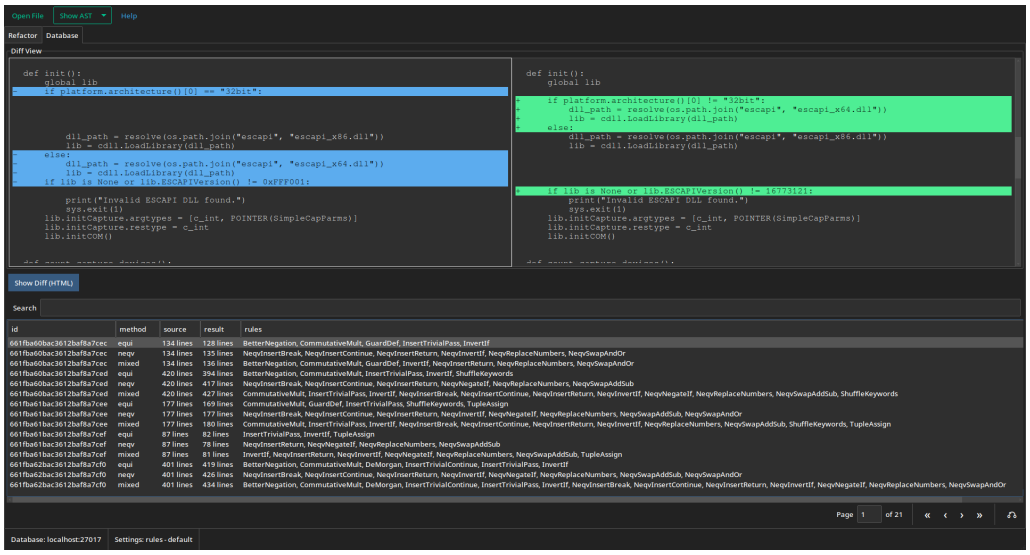


2.6. ábra. A *helloworld* program AST-je

2.4.5. Adatbázis-böngésző nézet

Ebben a nézetben megtekinthetjük az adatbázisba bekerült forráskód-párokat. A nézet csak akkor jön létre ha az alkalmazásnak van adatbázis elérése, ha nincs azt a nézeten a *"No database connection."* felirat jelzi. A nézet feladata a forráskód-párok listázása és a párba állított forráskódok különbségeinek megjelenítése.

A különbségeket két forráskód között könnyen vizualizálhatjuk egy diffel, egy olyan szövegösszehasonlító programmal ami két szöveg között a különbségek listáját állítja elő. A különbségeket a forráskód-párokban ezzel a módszerrel vizualizálom.



2.7. ábra. Adatbázis-böngésző nézet

A nézet tetején találhatók a diffeket megjelenítő szövegdobozok, ezek alatt egy táblázat látható, soraiban az adatbázisba bekerült forráskód-párokkal. A táblázat soraiban található adatok sémáját az alábbi táblázat írja le:

Oszlop	**Típus**	**Magyarázat**
id	object id	az eredeti forrráskód azonosítója
method	string	az átalakításra használt módszerre utal (például a szabályhalmazra)
source	string	az eredeti forráskód sorainak száma
result	string	az átalakított forráskód sorainak száma
rules	string	az átalakításnál alkalmazott szabályok listája

2.1. táblázat. Adatbázis-böngésző nézet táblázatának oszlopai

Ha a táblázat egy sorára, vagyis egy forráskód-párra klikkelünk akkor a diff nézetben megjelennek az eredeti (bal oldali) és az átakított (jobb oldali) kód közti különbségek.

A táblázat felleti keresőt használathatjuk a forráskód-párok böngészéséhez. A kereső az összes sorban és oszlopban szereplő adatok közt keres. Például megkereshetjük, hogy az adatbázisban mely forráskódokon kerültek for ciklussal kapcsolatos átalakítások alkalmazásra.

3. fejezet

Fejlesztői dokumentáció

Ebben a fejezetben a szoftver működését részletezem. Például szoftver felépítésére, a szoftverben használt tervezési mintákra és a fontosabb algoritmusok működésére is kitérek. A szoftver forráskódja a jövőben változhat, a frissített API dokumentáció ezért a személyes oldalamon is elérhető [2].

3.1. Csomagok és modulok

A szoftver forráskódja több csomagban és modulban található. A forrráskód jelentős része öt fő csomagba van szerveze, ezek az alábbi táblázatban láthatók.

Csomag	**Rövid leírás**
app	GUI alkalmazás csomagja
client	adatbázis kliens
model	adatok modellezése és mentése
tests	egység és egyéb tesztek
transformations	átalakítások forráskódja és API az átalakításokhoz

3.1. táblázat. A szoftver fő csomagjai

A fő csomagok (a *tests* csomag kivételével) nem tartalmaznak "futtatható" fájlokat, a belépési pontok külön modulokba vannak szervezve.

Modul	**Rövid leírás**
launch	GUI alkalmazás belépési pontjának modulja
persistor	CLI program belépési pontjának modulja
utils	utility függvények modulja (pl. fájlok olvasásához)

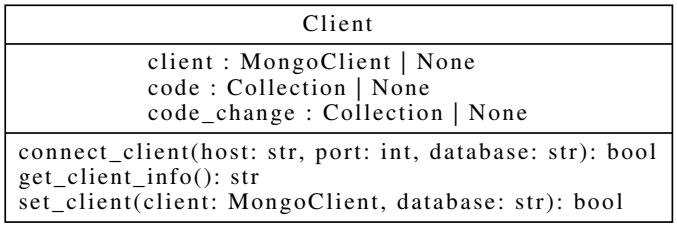
3.2. táblázat. A szoftver fő moduljai

3.2. A *client* csomag

A *client* csomag feladata az adatbáziskapcsolat és az indexek létrehozása. A kliens a *pymongo* könyvtárat használja, implementációja a *Client* osztályban van. Ha szoftvernek az adatbázisra van szüksége azt ezen az osztályon keresztül érheti el. Például az adatbázis forráskódokat és forráskód-párokat tartalmazó kollekciói a kliensen keresztül elérhetők.

A *Client* osztály a Python-ban gyakori *monostate* [3] tervezési mintát használja, a minta a *singleton*-hoz hasonló, de a több példány létrehozását is megengedi. Egy *monostate* osztálynak van egy belső (statikus) állapota, példányosításnál ezt a belső állapotot adja vissza. Ez hasznos, mert a példányok egy közös állapoton osztoznak, ami a program több részéről is elérhető.

Python-ban az objektumok állapota reprezentálható egy dict segítségével, ezért a *monostate* mintát nagyon egyszerű implementálni: a belső állapot egy dict lesz, példányosításnál a belső állapot dict-je alapján létrehozunk egy objektumot.



3.1. ábra. A *Client* osztály UML diagramja

Amikor először példányosítunk a *Client*-ből a *client*, *code* és *code_change* attribútumai *None* értékeket vesznek fel (ez a kezdelteges belső állapot).

Ha ezután meghívjuk a *connect_client* metódust az adatbázis paramétereivel és a kapcsolat 10 másodpercen belül létrejön, akkor a kapcsolat sikeres. Ekkor a *client* attribútum az adatbáziskliens, a *code* és *code_change* attribútumok pedig rendre a kódokat és kód-párokat tartalmazó kollekciók lesznek.

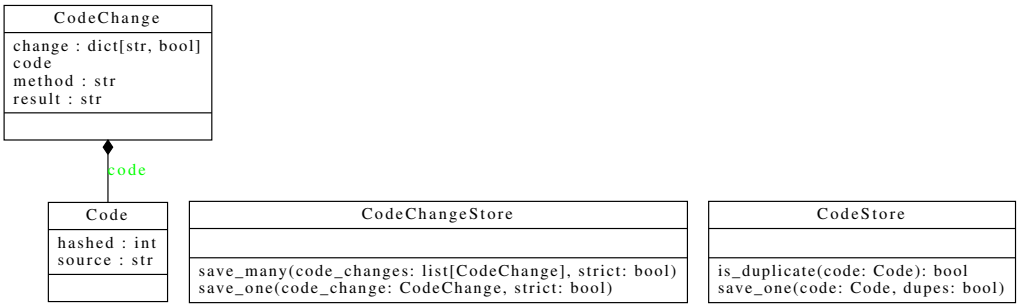
A kollekciók akkor is létrejönnek a kliens szintjén ha az adatbázisban még nem szerepelnek. Ebben az esetben a kollekció az adatbázisban akkor jön létre ha a kliensen keresztül elmentünk egy dokumentumot. A kliens a szükséges indexeket is definiálja.

3.3. A *model* csomag

A *model* csomag a feladata a forráskód-párok modellezése, három modulból áll:

- *datatypes* - az adattípusokat definiáló modul
- *serializers* - az adattípusokat szerializáló modul
- *stores* - az adattípusokat elmentő modul

A csomagban két adattípust definiálok a *Code* és *CodeChange*. A *Code* a forráskódok modellje, a *CodeChange* pedig a forrráskód-párokat modellezi.



3.2. ábra. A model csomag osztályainak UML diagramjai

A forráskód a duplikátumok kiszűrése miatt rendelkezik saját modellel. A duplikált forráskódok szűrése azért szükséges, mert GitHub-on a kódok jelentős része duplikátum [4], vagyis ha GitHub-ról bányászunk kódokat akkor a generált adathalmaz minőségén javíthatunk, a duplikátumok kiszűrésével.

A kódok modellje ezért a kód mellett a kód *sha256*-os hash-ét is tárolja. Mielőtt a kódot elmentjük az adatbázisba megnézzük, hogy a hash ütközik-e. A hash attribútum indexelve van a kódokat tartalmazó kollekcióban, ez biztosítja a gyors lekérdezést. Ha nincs ütközés, akkor a kódot egyből hozzáadhatjuk az adatbázishoz, ha vannak ütközések, akkor csak az ütköző kódokat kell összehasonlítani.

A duplikátumok kiszűrését a *CodeStore* osztály *save_one* metódusa végzi, a *CodeChangeStore* osztály segítségével pedig a kód-párokat tartalmazó kollekcióba szúrhatunk be dokumentumokat.

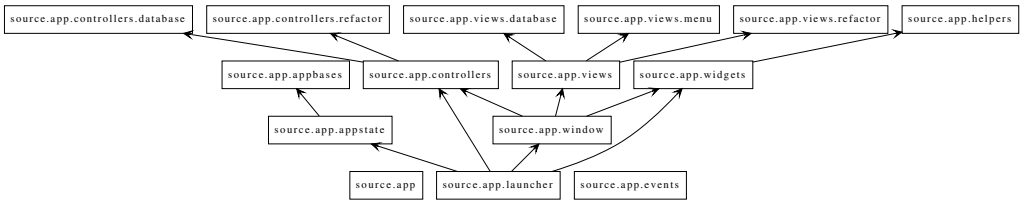
A *CodeStore.save_one* metódusában a duplikátumok szűrése opcionális, tehát ha duplikátumoktól mentes kódokból is generálhatunk adathamlazt ha szeretnénk.

3.4. Az *app* csomag

Az *app* csomag feladata az átalakításokat szemléltető GUI-s alkalmazás megvalósítása. Az alkalmazás architektúrája modell-nézet-kontroller (MVC) szerű. Egy nézet rendelkezik egy kontrollerrel és a kontroller pedig egy modellel.

A nézet feladata a GUI definiálása és frissítése, a modell feladata az adatelérés vagy az alkalmazás állapotának modellezése. A kontroller ezt a két réteget köti össze, így a nézet nem függ a modelltől és a modell sem a nézettől.

Az alkalmazás csomagjainak UML diagramját az alábbi ábrán láthatjuk.



3.3. ábra. Az alkalmazás csomagjai

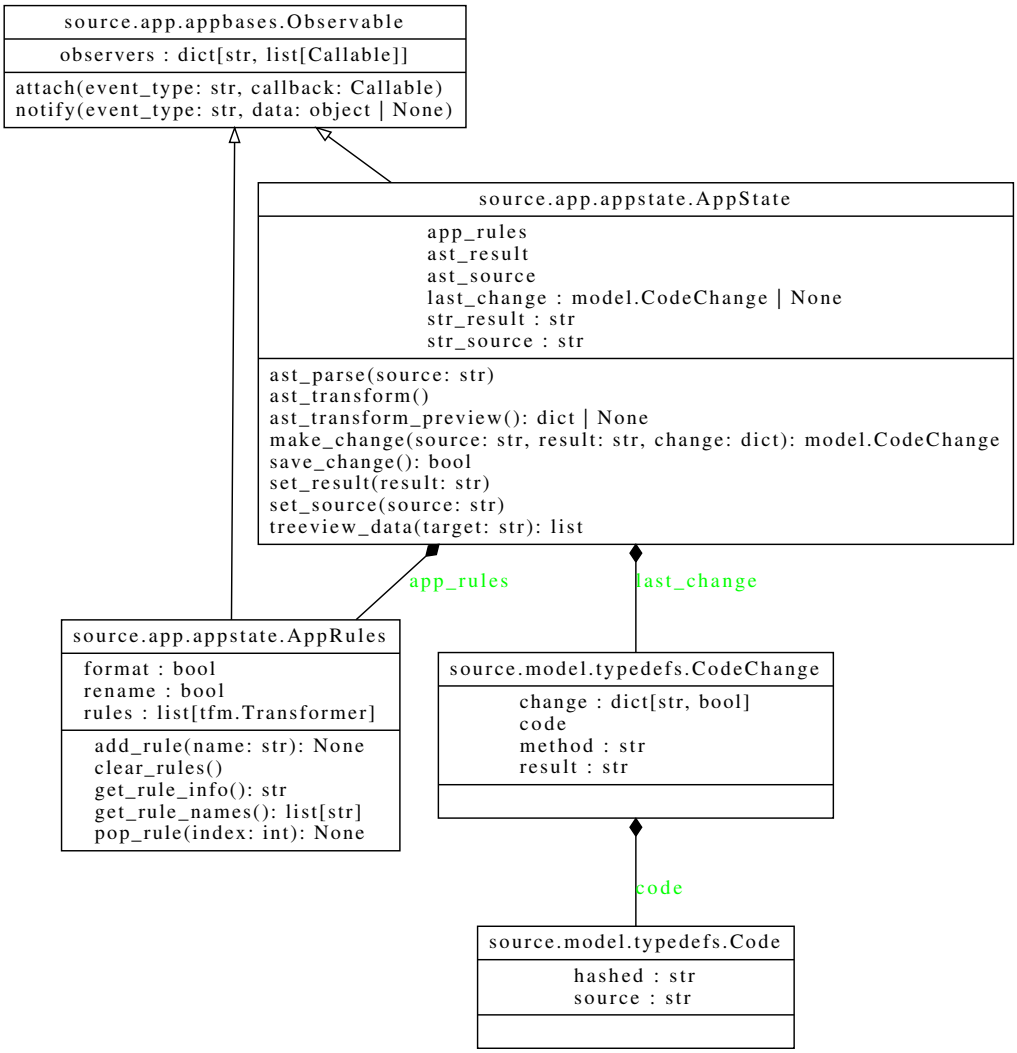
3.4.1. Állapotmodell

Az alkalmazásban kétféle modell különböztethető meg: az adatelérési modellek, amik működéséről az előző szekcióban olvashatunk és az alkalmzás állapotmodellje ami az *app.appstate* modulban található.

Az állapotmodell az *AppState* osztályban van definiálva és az *observer* tervezési mintát használja a nézetek frissítésére. Az *AppState* osztály az *Observable* osztályból származik, ezért rendelkezik megfigyelők (*observers*) egy listájával, ami esemény-eseménykezelő párok listája.

Ha a nézet egy komponenesét az állapotmodell változásának hatására szeretnénk frissíteni, akkor azt az eseménykezelőt, ami frissíti, hozzárendelhetjük az állapotmodell egy eseményéhez az *app.events* modulból.

Hozzárendelni egy eseménykezelőt egy eseményhez az *Observable* osztály *attach* metódusával lehet. Ha az adott eseményt kiváltja egy változás a modellben, akkor a modell értesíti a nézeteket, vagyis az *observers*-ben az eseményéhez rendelt eseménykezelőket meghívja, a frissítéshez szükséges adatokat paraméterként továbbítva.



3.4. ábra. Állapotmodell és kapcsolódó osztályok UML diagramjai

Az alkalmazás egy állapotát a bemeneti és kimineti AST-k, az ezekből generált kódok, az átalakításért felelő szabályok listája, és az utolsó átalakítás írják le.

Az AST-k és a belőlük generált kódok az *AppState* osztály példány szintű változói. Ezeken kívül az állapotmodell az *AppRules* és a *CodeChange* osztályok egy-egy példányát is tartalmazza, ezek rendre az átalakításért felelő szabályokat és az utolsó átalakítást tárolják.

Az *AppRules* a szabályok állapotát modelezi, szintén az *Observable*-ből származik. Ez az osztály tartalmazza a kiválasztott szabályokat a *rules* listában, a *format* és *rename* bool-okkal pedig azt tárolja, hogy az átalakított kódot kell-e formatálni és az átnevezéseket végre kell-e hajtani.

Az utolsó valid átalakítást a *CodeChange* osztály egy példánya tárolja. Ahogy azt az előző szekcióban is említettem ezzel az osztállyal, lehet kimenteni az átalakítást az adatbázsiba.

Az *AppState* metódusai segítségével változtathatjuk a modellt. A metódusok az AST-k átalakítását és az utolsó átalakítás mentését valósítják meg, ezek a metódusok váltják ki a modellel kapcsolatos eseményeket is.

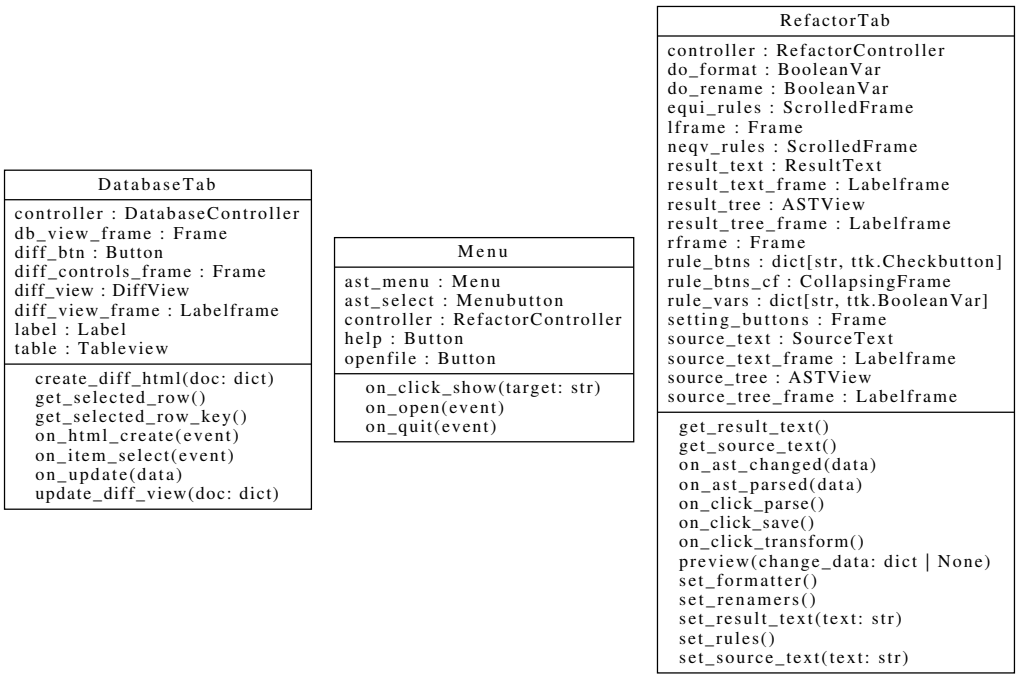
3.4.2. Nézetek

Az alkalmazás grafikus felhasználói felületének megvalósításához a Python-ban alapból megtalálható *tkinter* könyvtárt használom, amit az erre építő *ttkbootstrap* könyvtárral egészítek ki. A felhasználói felület forráskódja az *app.views* csomagban és az *app.widgets* modulban található.

A *tkinter* könyvtárban a GUI elemeket widget-eknek hívják, az *app.widgets* az ilyen widget-eket definiálja, például a Python szintaxis kiemelést támogató szövegdobozt vagy az AST-ket ábrázoló fa-nézeteket.

Az applikáció komplexebb nézeteinek definíciói az *app.views* csomagban vannak. Ezek a nézetek szintén widget-ek, de az *app.widgets* widget-eivel ellentétben egy kontrollerel rendelkeznek, amit a modellel való kommunikációhoz használnak (az *app.widgets* widget-ei nem férnek hozzá a modellekhez).

Az alkalmazásnak három kontrollerrel rendelkező widgete van, ezek osztálydiagrammjait az alábbi ábrán láthatjuk.

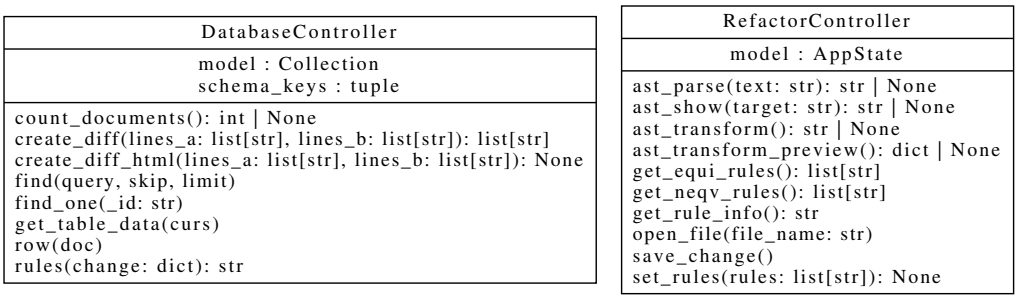


3.5. ábra. A nézetek osztálydiagrammjai

3.4.3. Kontrollerek

A kontrollerek feladata a kommunikáció a modellek és nézetek között. Csak a felhasználói felület két fő nézete (*DatabaseTab* és *RefactorTab*) illetve a menü nézete (*Menu*) rendelkeznek kontrollerrel.

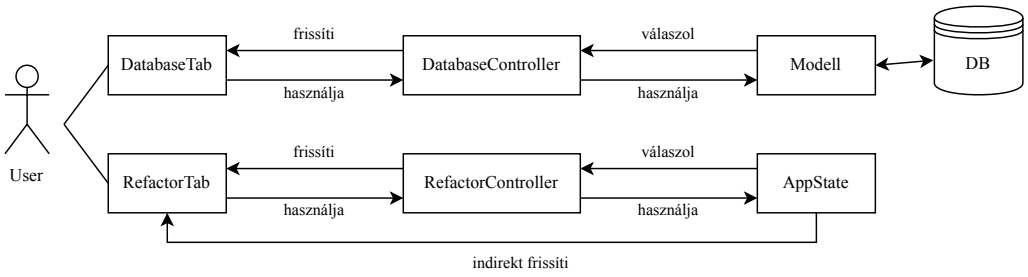
Ahogy azt a 3.5. ábrán láthatjuk az applikációban két kontroller van. A *DatabaseController* az adatelérési modellekkel, a *RefactorController* az alkalmazás állapotmodelljével kommunikál.



3.6. ábra. A nézetek osztálydiagrammjai

Ha a nézeten olyan GUI esemény, történik aminek az eseménykezelője a modell vagy az állapotmodell használatát igényli, akkor az eseménykezelő a nézethez tartozó kontroller megfelelő metódusát hívja meg. Ha az eseményhez input is tartozik (pl. egy szöveges doboz tartalma) akkor azt is továbbítja a kontroller metódusának paraméterként.

A kontroller használja a modellt, lekérdezéseket vagy változtatásokat végez benne, ha ezek megtörténtek a nézetet direk vagy indirekt módon frissíti. Direkt módon frissíti, ha a modelltől kapott adatokat a nézetnek továbbítja, ami azokkal frissül. Ha a kontroller egy eseményt vált ki a modellben, aminek hatására a nézet frissül, akkor indirekt frissíti. Ezt a működést az alábbi ábrán láthatjuk.



3.7. ábra. Egy esemény kezelése az MVC architektúrában

Az alkalmazásban csak a *RefactorController* végez indirekt frissítést a 3.4.1. alcím alatt részletezett *Observable* tervezési minta segítségével.

3.5. A *transformations* csomag

TODO

3.6. Tesztelés

TODO

4. fejezet

Összegzés

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Köszönetnyilvánítás

Amennyiben a szakdolgozati / diplomamunka projekted pénzügyi támogatást kapott egy projektből vagy az egyetemtől, jellemzően kötelező feltüntetni a dolgozatban is. A dolgozat elkészítéséhez segítséget nyújtó oktatók, hallgatótársak, kollégák felé is nyilvánítható külön köszönet.

A. függelék

Szimulációs eredmények

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus,

sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

Irodalomjegyzék

[1] mongodb docs. *MongoDB Installation*. URL: https://www.mongodb.com/docs/manual/installation/.

[2] Verebics Péter. *API dokumentáció*. URL: http://szonyegxddd.web.elte.hu/szakdolgozat_api_doc/index.html.

[3] VasileAlaiba. *Monostate Pattern*. 2014. URL: https://wiki.c2.com/?MonostatePattern.

[4] „DéjàVu: a map of code duplicates on GitHub". *ACM Journal* (2017). URL: https://dl.acm.org/doi/10.1145/3133908.

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusjegyzék

Forráskódjegyzék