Önálló laboratórium beszámoló **BME-TMIT**

Készítette: **Sümeghy Tamás Pál**

Neptun-kód: GFHSRE
Szak: Műszaki informatikus
Szakirány: Internet és infokommunikációs alkalmazásai
E-mail cím: schumy@sch.bme.hu

Konzulensek:

Kardkovács Zsolt Tivadar kardkovacs@tmit.bme.hu
Sárecz Lajos (Oracle – külső konzulens) lajos.sarecz@oracle.com

Tanév: **2008/2009. 1. félév**

Téma címe: **Oracle Spatial alkalmazás fejlesztése**

Feladat:

Az idei féléves feladat egy rendszer implementálása a korábbi félévben létrehozott rendszerterv alapján. Az alkalmazás egy helymeghatározó megoldással ellátott mobil eszközökről (főleg laptopokról vagy kéziszámítógépekről) elérhető szolgáltatás, melynek lényege, hogy a térképet nem a mobil eszközön, hanem egy központi szerveren tárolja, és azt a mobil eszköz távolról éri el. Ezen a térképen a szolgáltatás lehetőséget biztosít hirdetések, forró pontok elhelyezésére, valamint a szolgáltatás többi felhasználójának követésére.

Az implementáció Java nyelven történik, az alkalmazás alapjául pedig egy Spatial opcióval kiegészített Oracle 10*g* adatbázis szolgál majd.

1. A laboratóriumi munka környezetének ismertetése, a munka előzményei és kiindulási állapota

Bevezető/elméleti összefoglaló

A féléves munkám két nagy részre tagolható. A félév elején az előző félévben elkészített rendszerterv módosításával foglalkoztam, a későbbiekben pedig a megtervezett rendszer implementálása volt a feladatom.

Az előző félévben készített rendszerterv módosítása azért volt szükséges, mert több olyan hibát követtem el, amik jelentősen megnehezítették volna a fejlesztés folyamatát. Ilyen hiba volt többek között a szereplők igényeinek, szempontjainak definiálásának hiánya vagy a rendszer illetve az egyes komponensek által elvégzendő feladatok, funkciók, elvárt viselkedési minták nem teljes körű meghatározása. A rendszertervből szinte teljesen hiányzott a tesztelési terv illetve erősen hiányos volt az üzemeltetési terv is, miközben ezek alapvető elemei egy alkalmazás rendszerszintű tervezésének*[1]*. Ezeknek a hibáknak a kijavítása elsődleges fontosságú volt, mivel ezek nélkül az implementáció elkezdése nem volt lehetséges.

A fentiek mellett hiányosságok voltak a dokumentum szerkezetében és mérnöki minőségében is, ezek azonban nem gátolták a fejlesztés folyamatát, mindazonáltal a komolyabb hibák kiküszöbölése során ezeket is figyelembe vettem.

Mint látható, az implementáció erősen kötődik a rendszertervhez. Egy jól megírt rendszerterv segítségével a fejlesztés bizonyos szakaszai automatizálhatóak, lényegesen egyszerűbbé tehetőek. Emellett a fejlesztés folyamata követi a rendszerterv létrehozásának folyamatát, amennyiben mindkettőt jellemzi a hierarchikus felépítés. Rendszertervezéskor először egy nagy egészként tekintünk a rendszerre, és annak tipikus elemeit, feladatait, attribútumait definiáljuk először; majd ezt finomítjuk tovább mindaddig, míg el nem jutunk a legkisebb részmodulokig. Ugyanez az elv követhető a fejlesztés során is, vagyis először a teljes alkalmazás vázát hozzuk létre, majd ennek elemeit dolgozzuk ki egyre részletesebben. Létezik a másik véglet is, melynek során egy nagyon kis méretű, de teljesen működő rendszerből indulunk ki, melyet aztán a specifikációban megadottak szerint bővítünk addig, amíg el nem érjük a teljes funkcionalitást nyújtó alkalmazást.

Természetesen a két véglet között számos középút található, melyek a két elképzelés vegyítésével jönnek létre. Én is egy ilyen köztes megoldást alkalmaztam a fejlesztés során, de elmondható, hogy inkább az első megközelítést alkalmaztam, vagyis törekedtem a rendszer szintű gondolkodásra. A második irányelvet követem ugyanakkor abból a szempontból, hogy a rendszer magjának teljes funkcionalitása előrébb került a kapcsolódó modulok fejlesztésénél (kapcsolódó modul például egy, a szerveralkalmazást elérő weblap).

A fejlesztés során a következő sorrendet követtem, illetve fogom követni:

- Alkalmazás alapjainak létrehozása
  - statikus osztályszerkezet létrehozása (mind szerver, mind pedig kliens oldalon), üres függvényekkel, külső fájl beolvasási és fájlba írási képességekkel
  - bemeneti fájl parancsainak segítségével üzenet- és hívássorrendek tesztelése mind a két fő modulban

- Adatbázissal kapcsolatos feladatok
  - megtervezett adatbázisséma alapján az adatbázis létrehozása, majd tesztadatokkal feltöltése (térképi adatok nélkül)
  - szerver oldalon adatbázis-kapcsolat létrehozása, tesztelése
  - szerver oldali adatbázis elérést szolgáló osztályok függvényeinek létrehozása
  - adatbázist elérő függvények tesztelése

- Működési logika felépítése
  - üzenet- és hívássorrendek alapján a függvények valódi tartalommal feltöltése mind kliens, mind pedig szerver oldalon
  - két fő modul funkcionális tesztelése külön-külön

- Kommunikáció felépítése a két modul között
  - üzenet osztály létrehozása
  - kommunikáció felépítése a két modul között, valódi kommunikáció modellezése
  - kliens oldali kommunikációs függvények létrehozása (üzenetek összeállítása, válaszüzenet feldolgozása)
  - szerver oldali kommunikációs függvények létrehozása (üzenetek feldolgozása, válaszüzenet összeállítása)
  - alapszintű kommunikáció tesztelése

- Valódi kommunikáció létrehozása
  - kapcsolatkezelés kliens oldalon (kapcsolatfelépítés kérése a szervertől, üzenetküldés a kommunikációs csatornán)
  - kapcsolatkezelés szerver oldalon (több kliens kezelése egyszerre, felhasználók kezelése, üzenetek fogadása, küldése)
  - tesztelés

- Alkalmazás felkészítése térképi adatok kezelésére
  - térkép feltöltése az adatbázisba
  - térképi adatok, lekérdezések kezelése kliens és szerver oldalon is
  - térképi adatok és azok elérésének tesztelése

- Kliens oldali felhasználói felület létrehozása
  - megjelenítés, különös tekintettel a térképi adatok megjelenítésére
  - felhasználói akciókezelés megvalósítása

- Szerver oldali felhasználói felületek (adminisztrátori és hirdetői felület) létrehozása
  - a két tervezett weblap összeállítása
  - funkcionalitás rendszerhez illesztése

Az alkalmazás fejlesztésének nyelve a rendszertervben leírtaknak megfelelően a Java lesz. A fejlesztés során a JDK (Java Development Kit) 1.6 Update 3 verzióját használom. Konkrét fejlesztői környezetnek a JDeveloper-t választottam, mivel az alkalmazásfejlesztéshez rengeteg hasznos kiegészítő szolgáltatással rendelkezik. A fejlesztés során én ezek közül főleg az automatikus kódgenerálási és az adatbázis-tervezéssel kapcsolatos szolgáltatásokat veszem igénybe. A fejlesztést a 10.1.2.3.0 verzióval kezdtem el, bár időközben megjelent a 11-es verzió is. A tesztelés során azonban nem találtam annyival hasznosabbnak a fejlesztés szempontjából, hogy az új verziót használjam a továbbiakban.

Az alkalmazás alapját jelentő adatbázis egy Oracle Database 10*g* Release 2 (10.2.0.1.0) verziójú adatbázis. Eredeti terveimben egy 11*g* adatbázis szerepelt, ez azonban a fejlesztői gép erőforrásainak szűkössége miatt a tesztelések során használhatatlan volt, így döntöttem a 10*g* mellett. Funkcionalitás tekintetében ez a választás nem befolyásolja az alkalmazás működését, mivel a 11*g* verzió újdonságait nem tudom kihasználni az alkalmazásban*[3]*.

Az adatbázis eléréséhez az Oracle SQL Developer 1.5.1 (1.5.1.54.40) verzióját használom, ennek segítségével hoztam létre az adatbázist, itt töltöttem fel adatokkal is, valamint SQL utasítások tesztelésére is alkalmas.

A munka állapota, készültségi foka a félév elején

A félév elején rendelkezésemre állt az előző félévben létrehozott rendszerterv. Ez nem csak a komplett rendszer átfogó koncepcióját jelenti, hanem olyan részletesen megtervezett elemeket is, mint például az adatbázisséma terve, a modulok statikus osztálydiagramjai vagy a működésüket leíró folyamatábrák és üzenetszekvenciák.

A korábbi félévek munkáiból rendelkezésre álltak továbbá a használt technológiák felhasználói szintű ismerete, leírások, kézikönyvek; így ebben a félévben a technológia megismerése már nem igényelt többlet időráfordítást.

Konkrét rendelkezésemre álló segédanyagnak tekinthetőek az adatbázis-elérést biztosító Java forráskódok is*[2]*. Ezek segítségével érem el minden esetben az adatbázist, így jelentős segítséget jelentenek.

2. Az elvégzett munka és eredmények ismertetése

Az általam konkrétan elvégzett munka bemutatása

Rendszerterv módosítása

A rendszerterv módosításánál a konzulensem által összeállított hibalista mentén végeztem a javításokat. Első lépésként a funkcionális tervet bővítettem ki, konkrétan az egész rendszer alapvető követelményeit, feladatait, elvárt működési formáit pontosítottam. A felhasználók igényeit, szempontjait, a lehetséges felhasználói módokat újradefiniáltam, így már jobban elkülönülnek egymástól az egyes felhasználói szerepkörök.

Az üzemeltetési tervet új alapokra helyeztem, pontosan leírva egy adminisztrátor feladatait és lehetőségeit. Szintén az üzemeltetési terv részét képezte a napló és a naplóbejegyzések precíz, formális leírása, események definiálása.

Mivel az előző félévben a tesztelési terv nem készült el, így ezt is most pótoltam. Tesztelési stratégiám szerint minden fejlesztési fázis végén fogok teszteléseket végezni minden, az adott fázisban módosított modulon. Ezek a tesztek az alacsony szintű modulok esetén kimerítő tesztek lesznek, míg magasabb szintű egységeken már csak a funkcionalitást tesztelem. Az egyre magasabb hierarchiaszinteken emellett már megjelennek a hibatűrő és kritikus – idő- és erőforrás-kritikus – tesztek is.

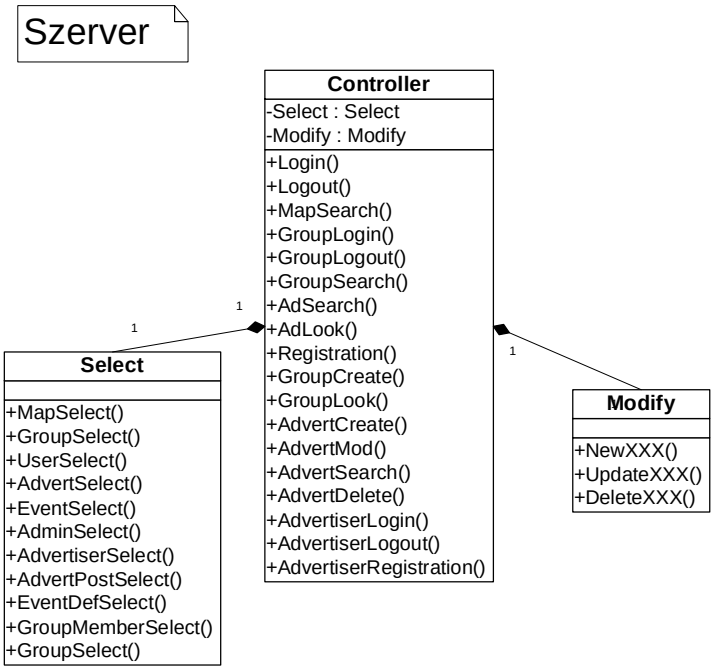
A megvalósítási tervben csak kismértékű módosításra volt szükségem, néhány új metódussal bővültek az osztályok, valamint bővültek és módosultak a viselkedést leíró diagramok.

Alkalmazás alapjai

A fejlesztés legelső lépése az alkalmazás alapjainak összeállítása volt. Mivel a végleges alkalmazás kliens – szerver architektúrájú lesz, ezért tulajdonképpen két külön alkalmazást kell fejlesztenem, melyek majd előre meghatározott interfészeken kommunikálnak egymással.

Elsőként tehát a rendszerterv megvalósítási tervében megalkotott statikus osztálydiagramokból – mint amilyen az 1. ábrán látható szerver oldali statikus osztálydiagram – generáltam Java kódot a JDeveloper segítségével. Minden osztálynak elkészült így egy üres konstruktora és minden függvénye, melyek szintén nem tartalmaznak parancsokat.

Ezt követően a tesztelhetőség alapját jelentő beolvasási és kiírási funkciókat valósítottam meg, vagyis a program egy bemeneti fájlból olvassa be a parancsokat, majd működése során egy kimeneti fájlba ír minden információt. A bemeneti fájl egy sora tartalmaz egy parancsot, a sor első szava a parancs, míg a paraméterek szóközökkel elválasztva következnek. A fájl feldolgozásakor soronként, szekvenciálisan haladok. A sort a szóközök mentén felbontva az első szó alapján dől el, hogy melyik függvényt szükséges meghívni, a paramétereket pedig változatlanul továbbadom a függvénynek.



1. ábra: Szerver statikus osztálydiagram

A program működésének alapját a függvényhívások egymásutánja adja meg. A függvénytörzsekbe bekerültek a különböző függvényhívások a megvalósítási terv alapján, majd a tesztelhetőség érdekében minden függvényben és konstruktorban kiírtam, hogy éppen milyen függvény hajtódik végre, vagy melyik osztály jött létre.

Mivel ekkor még a két modul között nincsen kapcsolat, ezért ezt mindkét modulon külön-külön végre kellett hajtani és le is kellett tesztelni. A 2. ábrán látható egy lehetséges parancsfájl, a 3. ábrán pedig a hozzá tartozó elvárt kimenetet egy része. A ”##”-kal kezdődő sorok jelentik a függvények által generált üzeneteket a függvényhívásokról vagy objektumok létrejöttéről. A többi sor az egyéb eseményeket és üzeneteket jelenti.

```
Registration username password
Login username password
Map
Navigate
GroupSearch csoport*
Logout username
```

2. ábra: Parancsfájl teszteléshez

```
## Object created: MyPackage.Controller_client@3
## Object created: MyPackage.GPS@4
## Object created: MyPackage.View@5
## Controller_client.Registration()
Message send to server
Message received from server
Successful registration
...
## Controller_client.Navigate()
## Controller_client.MapRequest()
## GPS.CoordinateRequest()
Message send to server
Message received from server
## View.Draw()
Draw map
...
```

3. ábra: Kimenet részlete

A teszteléssel ellenőriztem a megtervezett osztálystruktúra helyességét illetve az üzenetszekvenciákat. Elmondható, hogy a tervezés sikeres volt, és a generált kód alapján az üzenetszekvenciák összeállítása is könnyedén megvalósítható volt.

Adatbázissal kapcsolatos feladatok

Az adatbázissal kapcsolatos teendők közül az első a telepítés volt. Korábbi félévek tapasztalatai után ez nem okozott különösebb problémát. A telepítés után létrehoztam egy felhasználót az alkalmazás számára, a megfelelő jogosultságokkal ellátva. Az adatbázis felépítéséhez szintén a rendszerterv adta az alapot. Az ott megtervezett adatbázissémából a JDeveloper segítségével SQL DDL (Data Definition Language – az SQL adatdefiníciós utasításkészlete) utasítássorozatot generáltam, és SQL Developerben futtattam le az így kapott szkriptet.

A táblák nagy részében az elsődleges kulcs egy ID nevű oszlop. Mivel azt szerettem volna elérni, hogy ezek az azonosítók az egész adatbázisban minden egyes sorra egyediek legyenek, ezért létrehoztam egy közös számlálót. Minden ilyen táblára definiáltam egy triggert, mely új sor beszúrásakor a számláló következő értékét szúrja be azonosítóként. Azokban a táblákban, ahol az elsődleges kulcs több oszlopból áll, ott ezek az oszlopok idegen kulcsok is egyben, így ezek egyedisége is biztosított a triggerek által. Egy példa trigger látható a 4. ábrán.

```
create or replace TRIGGER TRG_ADMIN_USER
BEFORE INSERT
ON ADMIN_USER
REFERENCING NEW AS NEW
FOR EACH ROW
BEGIN
SELECT INSERT_SEQ.nextval INTO :NEW.ADMIN_ID FROM dual;
END;
```

4. ábra: SQL trigger az ADMIN_USER táblába beszúráshoz

Az így összeállított üres adatbázist feltöltöttem tesztadatokkal, miközben vizsgáltam, hogy a beállított triggerek és a táblákban lévő idegen kulcs kényszerek megfelelően működnek-e. A megtervezett séma jónak bizonyult, nem kellett változtatást végrehajtanom rajta egyelőre.

Az adatbázis létrehozása után az azt elérő utasítások létrehozása volt a következő feladat. Ezen utasítások két nagy csoportra bonthatóak, lekérdező jellegű és módosító jellegű utasítások. Ezen felbontás alapján az alkalmazásban is két osztályban valósítom meg az SQL utasításokat. A lekérdező típusú utasítások a Select, míg a módosító jellegű utasítások a Modify osztályba kerültek a szerver oldali modulban.

Módosító utasítások esetén minden lehetséges SQL parancsot megvalósítottam, vagyis az adatbázis minden táblájához elkészítettem egy beszúrás, törlés és módosítás függvényt. Ez alól csak az eseményeket tároló tábla kivétel, mivel ebben a táblába csak beszúrni lehet, törölni és módosítani nem, így a naplózás során egyetlen esemény sem veszhet el.

A függvények felparaméterezésénél alapvető szempont volt, hogy tetszőleges feladatra alkalmasak legyenek a módosító utasítások. Így gyakran szükséges egy-egy paraméter helyére *null*-értéket beilleszteni. Ezt szám típusú paraméter esetén a 0 értékkel, míg szövegek esetén a "" értékkel jelezhetjük a nullitást. A függvények visszatérési értéke minden esetben az adatbázisban módosított sor azonosítója, hibajelzésre a -1 értéket választottam.

Lekérdező utasítások esetén is minden lehetséges táblára összeállítottam egy általános lekérdezést, melyet tetszőlegesen lehet paraméterezni. Amelyik oszlopot nem kívánjuk szerepeltetni a lekérdezésben, ott a módosító utasításokhoz hasonlóan a 0 vagy "" értékkel jelezhetjük ezt. Minden lekérdezés egy eredményhalmazzal (ResultSet típus Java-ban*[2]*) tér vissza, hibajelzésre itt a ***null*** érték használható.

Az adatbázist elérő függvények mindegyikében alkalmaztam kivételkezelést, ezt egyébként a fejlesztőkörnyezet kötelezővé teszi, fordítási hibát jelez a hiánya esetén. Viszont hibát nem csak a beépített függvények okozhatnak, hanem futás közben a feldolgozott adatok minősége vagy felhasználói hibák miatt is bekövetkezhetnek. Ezeket az esetek is kiszűröm, új kivétel létrehozásával és eldobásával kezelem a hibát. A függvények visszatérési értékébe nem fér bele a hiba leírása, csak a hiba jelzése (-1 érték szám esetén, ***null*** érték eredményhalmaz esetén). Így a hiba leírását a két modul egy-egy globális változójába mentem, és ha feldolgozás közben hibajelzést észlel az alkalmazás, innen olvassa ki az utolsó hibaüzenetet.

Kimerítő tesztelést végeztem, azaz minden függvényt minden lehetséges paraméterkombinációval (természetesen kombináció alatt azt értem, hogy adunk-e értéket egy paraméternek vagy sem) kipróbáltam. A legtöbb probléma elgépelésből származott, például sokszor okozott gondot listák indexelése vagy az SQL utasítás megfelelő paraméterének kiválasztása. Valódi hibát fedeztem fel viszont a törléseknél több helyen is. Ha olyan sort szeretnék törölni az adatbázisból, amire más táblában hivatkozok, akkor először a hivatkozó sort kell kitörölni, és csak utána lehetséges a hivatkozott sor eltávolítása. Ezt a sorrendet néhány esetben nem tartottam be, de a tapasztalt hibaüzenetből kiindulva megoldható volt a probléma.

Működési logika, kommunikáció

A szerver és a kliens oldal, illetve a szerver és az ahhoz kapcsolódó adminisztrátori és hirdetői oldalak közötti kommunikáció megvalósításához létrehoztam egy üzenet osztályt. Az elküldendő üzenet minden esetben egy sztring, melyben az egyes paraméterek a http kérések paramétereihez hasonlóan jelennek meg.

```
Response=AdvertiserLogin&Result=OK&AdvertiserId=1489721
```

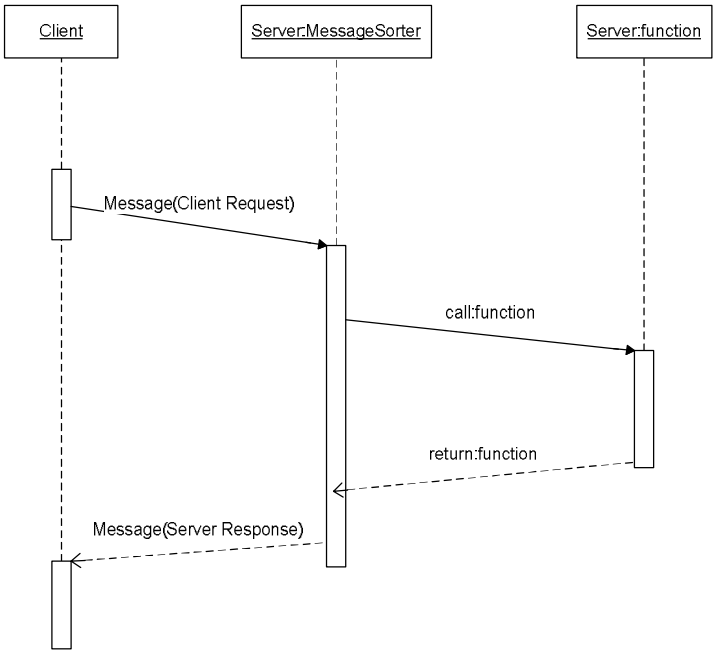
5. ábra: Minta üzenet-sztring

Minden kérés első paraméterének neve ***Request***, értéke pedig annak a funkciónak a neve, amit el kívánunk érni. A további paraméterek a kérés típusától függenek. Válaszokban az első paraméter neve ***Response***, értéke pedig kötelezően ugyanaz, mint a kérésben a ***Request*** paraméter értéke. Minden válasz tartalmaz egy ***Result*** paramétert, ami azt adja meg, hogy a kérés sikeresen végrehajtódott-e vagy sem. Ha egyszerű a válasz, mint például az 5. ábrán látható mintában, ahol egy azonosítót küldünk csak vissza, akkor ezt további paraméterekben tehetjük meg. Összetettebb esetekben, mint például egy eredményhalmaz visszaküldésekor kötelező paraméter a *Count*, ami az őt követő paraméterek számát adja meg. Ilyenkor a további paramétereket 0-tól kezdve sorszámozom, így kizárom annak lehetőségét, hogy két azonos nevű paraméter legyen egy üzeneten belül.

Az üzenet osztályban kétféleképpen is tárolom az üzenet tartalmát. Egyrészt egy privát elérésű sztringként, amit egy-egy metóduson keresztül lehet írni és olvasni. Másrészt egy ***Hashtable*** adatszerkezetben, amelynek lényege, hogy kulcs – érték adatpárokat lehet benne tárolni, ezért kifejezetten alkalmas az üzenet paramétereinek tárolására. Ez a tárolási forma csak

olvasható, mivel az írása bonyolult és felesleges is. A kulcs minden esetben a paraméter neve, míg az érték a paraméter értéke. Kiolvasásra főleg ez utóbbi tárolási formát használom, mivel közvetlenül és az adatszerkezetből kifolyólag gyorsan elérhető bármelyik paraméter értéke a paraméter nevével hivatkozva.

A modulok közötti kommunikáció során a szerver mindig vár bejövő üzenetekre, kérés nélkül sosem küld üzenetet. A 6. ábrán látható üzenetszekvencia zajlik le minden esetben a kliens és a szerver között. Kérés érkezhet a klienstől, az adminisztrátori és a hirdetői oldalról. A kérés küldője a kapott utasítások alapján felparaméterez egy üzenetet. A kérést jelentő üzenet összeállítása után a szerver kontroller moduljának üzenetosztályozójához kerül, ami a kérés alapján eldönti, hogy melyik függvénynek továbbítja azt. Itt megtörténik az üzenet paramétereinek kiolvasása, majd feldolgozása. Az eredmény itt is egy üzenetbe ágyazódik, majd visszakerül az üzenetosztályozóhoz, ami visszaküldi a kérést küldőnek. A küldő feladata az eredmény feldolgozása, kiírása, esetleges további akciók végrehajtása.



6. ábra: Üzenetszekvencia modell

A kommunikáció teszteléséhez a korábban látott bemeneti parancsfájlokat alkalmaztam. a 7. ábrán egy ilyen parancsfájl látható, a 8. ábra mutatja be a hozzá tartozó elvárt, helyes kimenetet. A tesztfájlban azt modellezem, ahogy egy felhasználó regisztrál, majd bejelentkezik a rendszerbe, létrehoz két csoportot, végrehajt egy keresést a csoportok között, belép az egyik csoportba, majd kilép onnan, végül kilép a rendszerből is. Az első két utasítás kivételével mindenütt az első paraméter a felhasználó azonosítója, amit a bejelentkezés után kap meg.

```
Registration tamas tamas
Login tamas tamas
GroupCreate 154 csoport4
GroupCreate 154 csoport5
GroupSearch 154 csop*
GroupLogin 154 csoport2
GroupLogout 154 csoport2
Logout 154
```

7. ábra: Parancsfájl teszteléshez

```
Sikeres regisztráció
Sikeres belépés
User id: 154
Csoport létrehozva
Csoport létrehozva
Csoportok:
   ID    NÉV
   --------------------
   4     csoport1
   110   csoport2
   111   csoport3
   157   csoport4
   159   csoport5
Sikeres csoportjelentkezés
Sikeres csoportlejelentkezés
Sikeres kijelentkezés
```

8. ábra: Várt kimenet

A két tesztfájl nagyon hasonlít a legelső, üzenetszekvenciákat tesztelő fájlokhoz. Ez nem véletlen, célom éppen az volt, hogy érzékeltessem a különbséget a két állapot között. Míg ott a teszt csak a kliens oldali osztályokat tesztelte, és a benne szereplő adatok mind beégetett adatok voltak, addig ez utóbbi esetben már a két modul együttes működésének tesztelése zajlik, valós, adatbázisból vett és oda beírt adatok alapján.

A tesztelés során nem csak a kimeneti fájlt, hanem a napló tartalmát is figyeltem, mivel így tudtam ellenőrizni, hogy megfelelően működik-e az események naplózása. A tesztelés során hibát okozó futtatások nagy többségénél elgépelés okozta a hibát. Ezek mellett származott hiba többek között rossz utasítássorrendből, az kapcsolatleíró és az adatbázis kurzorok nem megfelelő kezeléséből. Minden esetben a hibaüzenetek kellő támpontot adtak a hiba javításához.

Összefoglalás

A korábbi félévek során megismerhettem az irodalom-feldolgozás és a rendszertervezés alapjait, míg az idei félévben az implementáció rejtelmeibe nyerhettem betekintést. Most sikerült igazán megérteni a rendszertervezés lényegét, fontosságát, látva mennyire meghatározza és optimális esetben megkönnyíti az implementációt a terv. Míg a tavalyi félévben a tesztelési terv például a kevésbé fontos részei közé tartozott a rendszertervnek, addig idén a fejlesztés során majdhogynem elsődleges szerephez jutott, és az egyik legfontosabb feladat lett egy használható tesztelési terv létrehozása.

Az előre kitűzött feladatot nem sikerült maximálisan teljesíteni, de így is sok minden elkészült ebben a félévben. Az alkalmazás jelenlegi forráskódja körülbelül 2800 sorból áll, és a végleges változat elkészítéséig ez még bővülni fog. A tesztelések során körülbelül 100 tesztesetet futtattam le, és vetettem össze a kimenetét az elvárttal.

Az alkalmazás fejlesztésében célom elérni egy olyan szintet, melyben a teljes funkcionalitás mellett a megjelenítés és a térképi adatok korrekt kezelése megvalósul. Ez a következő hónapok feladata lesz.

3. Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke:

[1] Rendszerterv – dokumentáció a TMIT Adatbázisok oktatási laborjából (Kardkovács Zsolt) https://onlab.db.bme.hu/Gondolatok/Rendszerterv

[2] Gajdos Sándor (szerk.): Adatbázisok Laboratórium – Oktatási segédanyag a Számítógép labor V. tantárgyhoz, 10. bővített, javított kiadás, Budapest 2007.

[3] Jean Ihm: Oracle Spatial 11*g* – Advanced Spatial Data Management for the Enterprise, Oracle Data Sheet 2007. http://www.oracle.com/technology/products/spatial/pdf/11g/spatial-11g-datasheet.pdf

Csatlakozó egyéb elkészült dokumentációk / fájlok / stb. jegyzéke:

Az előző félévekhez hasonlóan idén is folytattuk a blogok írását. Ide került fel minden a félév során létrejött dokumentáció, forráskód, kép illetve beszámoló. Ez a blog teljesen nyilvános, bárki által elérhető. A blog címe: http://sumeghy-onlab.blog.hu.

Elkészített dokumentációk:

- Rendszerterv081008.pdf – végleges, módosított rendszerterv
- beszamolo_081016.ppt – labortársaknak tartott szóbeli beszámoló
- Fejlesztés.pdf – fejlesztői kézikönyv

Módosított képek, diagramok (rajzok a rendszerterv új változatához):

- ERdiagram.jpg – entitás-relációs diagram az adatbázisról (JDeveloper)
- kliens_activity.vsd – kliens oldal activity diagramja (MS Visio 2007)
- kliens_blokk.vsd – kliens oldal blokkvázlata (MS Visio 2007)
- kliens_obj.vsd – kliens oldal statikus objektumdiagramja (MS Visio 2007)
- szerver_blokk.vsd – szerver oldal blokkvázlata (MS Visio 2007)
- szerver_obj.vsd – szerver oldal statikus objektumdiagramja (MS Visio 2007)
- szerver_sequence_kliens.vsd – szerver oldali szekvenciadiagramok (MS Visio 2007)
- szerver_sequence_pages.vsd (MS Visio 2007)
- teljes_blokk.vsd – teljes rendszer blokkvázlata (MS Visio 2007)

Forráskódok:

- Application.java – keretosztály az alkalmazás futtatásához
- Controller_client.java – kliens oldali működésért felelős osztály
- Controller_server.java – szerver oldali működésért felelős osztály
- GPS.java – kliens oldali GPS kezelő osztály
- Message.java – üzeneteket reprezentáló osztály
- Modify.java – szerver oldali adatbázis módosításokért felelős osztály
- Select.java – szerver oldali adatbázis lekérdezésekért felelős osztály
- View.java – kliens oldali megjelenítés-kezelő osztály