DEBRECENI EGYETEM
INFORMATIKAI KAR

Webes interfészek generálása UML-diagram alapján

Témavezető:
Kollár Lajos
Egyetemi tanársegéd

Készítette:
Papp Nóra
Programtervező informatikus

Debrecen
2009

Köszönetnyilvánítás

Ezúton szeretném megköszönni tanáraimnak a tanulmányaim alatt nyújtott segítséget, szakmai támogatást, állandó tanácsokat, melyek nélkül szakdolgozatom valószínűleg nem készült volna el. Szeretném megköszönni Kollár Lajosnak a munkám során nyújtott építő jellegű kritikát és a rám fordított időt.

Tartalomjegyzék

1 Bevezetés

A 21. század a számítógépek kora. Manapság szinte nem találni olyan háztartást ahol ne lenne legalább egy komputer sőt, szinte nem találni olyan háztartást, amely ne rendelkezne internet hozzáféréssel. Ma elvárás az, hogy tudjuk kezelni legalább alap szinten a számítógépet, hisz a legtöbb munkahelyen része azon eszközöknek, amelyek segítik munkánkat. Vannak speciális számítógépes alkalmazások, amelyek direkt feladatok elvégzésére készültek, de a hétköznapok internetes ügyintézése is megköveteli, hogy az ember értse annak működését. Ahhoz, hogy egy asztali-, vagy akár egy webes alkalmazás is jól, a célnak megfelelően működjön fontos, hogy mind a háttérben álló rendszer, mind a felhasználó megkapja a megfelelő információkat. A rendszer és a felhasználó közötti kulcsfontosságú kapocs, amelyen keresztül kommunikálnak egymással, a felhasználói felület. A rendszer használói magát a rendszert gyakran a felhasználói felületről ítélik meg, ami ha rosszul van megtervezve (például nehézkes a használata, vagy bonyolult a navigáció)a felhasználók inkább egy másik, ugyanerre a célra fejlesztett programot alkalmaznak, még akkor is, ha ugyanaz, esetleg korlátozottabb funkcionalitás van mögötte.

Szakdolgozatomban azt a célkitűzést választottam, hogy olyan automatizálható eljárást hozzak létre, amellyel webes interfészt lehet készíteni. Mivel a tervezés mindennek az alapja, ezért az automatizálás középpontjában a tervezési fázis végén előállított diagramok álltak, ezzel biztosítva a későbbi felület átgondoltságát. Elsődleges szempont volt az is, hogy a felületen megjelenő információk dinamikusan, a lapok újratöltése nélkül jelenjenek meg. Olyan technológiákat választottam a megvalósításhoz, amelyeket a tanulmányaim során még nem alkalmaztam, de ismeretük elengedhetetlen a mai internet központú világban. Célom e technológiák tanulmányozása és elsajátítása volt.

Az internetezés története

Az elmúlt évtizedben szédületes növekedés volt tapasztalható az Internet világában. Nagyságrendekkel megnőtt a rendszeresen internetezők, és a hálózatba bekötött számítógépek száma. Az Internet fejlődésének három szakaszát tudjuk megkülönböztetni.

Az első szakasz az 1970 előtti időszak**,** melyben az Internet kialakulásához szükséges technológiák elvének kidolgozása és néhány próbálkozás történt az elvek gyakorlati használhatóságának bizonyítására.

A fejlődés második szakaszának a hetvenes-nyolcvanas éveket számítjuk, ekkor a kísérleti számítógép hálózatok kialakulásától kezdve a nagyméretű hálózatok kialakulásáig a technológia elerjedése volt a legfontosabb jelenség.

A kilencvenes évektől napjainkig számítjuk a harmadik szakaszt, a Web korszakát. A hidegháborús korszak közepén mindenki egy újabb világháború kirobbantásától tartott. Az Egyesült Államok katonai vezetése a második világháborús tapasztalatok alapján tudta, hogy a hírközlő hálózatok és a számítógépek a leginkább sérülékeny elemei a hadseregnek. Olyan adattovábbítási módszerre volt szükség, amely a hírközlő csatornák részleges sérülése esetén is nagy biztonsággal juttatja célba a szükséges információkat. Azt, hogy a számítógépek összekötésének ekkora jelentősége lehet, J.C.R. Lickrider fogalmazta meg 1962-ben, amikor olyan számítógép hálózatról írt, melynek segítségével bárki, aki egy számítógépes terminál előtt ül, képes távoli számítógépekben tárolt adatok elérésére, és a másutt elhelyezett programok futtatására.

Naplófeljegyzés igazolja, hogy *a világon első alkalommal 1969.október 25-én sikerült létrehozni kapcsolatot két számítógép között a csomagkapcsolt módszerrel*. Idő múlásával az egyetemek és katonai intézmények mellett megjelentek a legfontosabb ipari és szolgáltató cégek az épülőfélben lévő világháló körül. Lassan, de biztosan beépül az akkor informatikai elit tudatába, hogy egy nagy robbanás van készülőben.

A kilencvenes évek kezdetekor az Internet alapját adó távközlési szolgáltatók nagyságrendekkel nagyobb sebességről, több számítógépről beszéltek. A legfontosabb azonban az, hogy a kialakult technológia és az egyre olcsóbb elektronikai eszközök sokkal nagyobb felhasználószámot jeleztek, mint az remélték. Az áttörést az egymással párhuzamosan keletkező és eltűnő adattárolási és keresési módszerek között Tim Berners-Lee ötlete hozta. Az ötlet lényege, hogy a szöveg "mögött" mindig a változásoknak megfelelő legfrissebb tartalom álljon. Ennek az ötletnek a tovább fejlesztésével keletkezett a ma ismert

www (world wide web) szolgáltatás. Ahogyan gyorsabbá vált az adattovábbítás, ahogy egyre nőtt az Internetre csatlakozó számítógépek száma, úgy tolódott el a fejlődés hangsúlya az Interneten elérhető alkalmazások/szolgáltatások fejlesztésének irányába.

Az Internet története korántsem ért véget. Mivel rohamosan nő a felhasználók száma, az új szolgáltatások újabb és több számítógépet, egyre nagyobb adatátviteli kapacitást igényelnek. A szakemberek folyamatos technológiai újításokkal készülnek a jövőre.

Honnan ered a *szörfözés* kifejezés? Jean Armour Polly, író és könyvtáros találta ki 1992-ben. Polly olyan kifejezést keresett, ami egyszerre fejezik az örömöt, amit az Internet használata jelent, és azt, hogy igenis képzettségre, kitartásra is szükség van, mire az ember igazán jól tud élni az Internet nyújtotta lehetőségekkel. Olyan elnevezést keresett, ami a véletlenek jelentőségére és a veszélyes helyzetek lehetőségére is felhívja a figyelmet. Tulajdonképpen egy, a könyvtárosok részére kiadott egéralátét segített neki.

2 Felhasználói felület

A felhasználói felület (UI – User Interface) egy berendezés (például a számítógép), vagy egy számítógépes program (például operációsrendszer) azon elemeinek összessége, amelyek a felhasználóval való kommunikációért felelősek, és a berendezés vagy program vezérlését teszik lehetővé.

2.1 Felhasználói felületek csoportjai

Egy gép irányításánál a parancsbevitel egyszerű, szöveges feliratú kapcsolókkal (nyomó- vagy érintőgombok, csúszkák stb.) történhet, míg az üzenetek kijelzése ledekkel, lámpákkal – ez a fénykijelzős parancsgombos felület.

Szoftveres irányításnál a felületeknek 3 fő fajtája van:

- Parancssoros felhasználói felület (CLI – Command Line Interface)

  A felhasználóval való kommunikáció parancsok segítségével történik. A felhasználó a billentyűzeten parancsokat gépel, melyeket a számítógép értelmez, végrehajt, és az eredményt az alapértelmezett kimeneten, többnyire a képernyőn jeleníti meg (a kimenet lehet tetszőleges fájl is), esetleg hangjelzéssel jelzi a parancsvégrehajtás befejezését.
  Parancssori felhasználói felülettel szinte mindegyik operációs rendszer rendelkezik, mert sok olyan feladat is megoldható vele, amelyekre a grafikus felhasználói felület nem ad lehetőséget, vagy a feladat annyira egyszerű, hogy egy bonyolultabb felhasználói felület készítése teljesen felesleges lenne.

- Szöveges felhasználói felület (TUI – Text User Interface)

  Ezen felület esetén, a monitoron szöveges feliratú karaktercellák jelennek meg. A fő beviteli eszköz a billentyűzet, a kurzor pozicionálásához speciális mutatóeszköz (TAB billentyű vagy egér) is használható. A kijelzés nagy részben karaktersoros formában

történik. Ez a típus nem a képernyő pixel alapú grafikus, hanem a szöveges üzemmódját használja.

- Grafikus felhasználói felület (GUI – Graphical User Interface)

  A számítógép és felhasználó közötti kapcsolatot megvalósító elemek a monitor képernyőjén szöveges és rajzos elemek együtteseként jelennek meg. A grafikus felhasználói felületeken alapvető szerepe van a mutatóeszközöknek (például az egérnek), amelyekkel a grafikus felület elemei intuitív módon kezelhetők. A leggyakoribb grafikus felhasználói elemek az ablakok, menük, választógombok, jelölőnégyzetek és ikonok, valamint a mutatóeszközhöz kapcsolódó egérkurzor.

Napjainkban további alkalmazott felhasználói felülettípusok például:

- Touch User Interface (TUI)

  Olyan grafikus felhasználói felület, amely kijelzője egy kombinált input-output eszköz.(például iPhone)

- Zooming User Interface (ZUI)

  Ez a felület a GUI logikus továbbfejlesztése, amely összevegyíti a 3D-mozgást a 2D vagy „2 és fél D” vektorobjektumokkal. Tehát olyan grafikus felhasználói felület, amelyben az információs objektumok különböző szintű skálával vannak reprezentálva, ahol a felhasználó ennek a skálának a nézetét tudja változtatni aszerint, hogy az mennyi részletet mutasson.

- Web User Interface / Web-based User Interface (WUI)

  Az inputból egy weblap generálódik, amely az interneten keresztül továbbítható a felhasználó felé, aki webböngészője segítségével tudja megtekinteni azt.
  Az újabb implementációk Java, AJAX, Adobe Flex, Microsoft ASP.NET és hasonló technológiákat használnak a valós idejű kontroll megvalósításához – kiküszöbölve a hagyományos HTML-alapú weboldalak állandó frissítési szükségét.

2.2 Felhasználói felület tervezése

A tervezés első lépése mindig a követelmények feltárása a rendszerrel, interfésszel szemben. Ez azt jelenti, hogy tudni kell, mik a későbbi felhasználó elvárásai. A felhasználói felület fontos, mert egy rendszer megítélése múlik rajta.

2.2.1 Az interfésztervezés emberi tényezői

Egy rendszer megtervezésénél vannak már bevált szempontok, amelyek figyelembe vételével felhasználóbaráttá tehető a szoftver.

- Az emberi memória rövidtávon korlátozott, azaz az emberek általában csak 7 információs egységet tudnak fejben tartani. Ha ennél többet ajánlunk fel, akkor nagyobb eséllyel vétenek hibát.

- Amikor emberi hibák miatt rendszerhiba lép fel, a nem megfelelő riasztások és hibaüzenetek növelik a stresszt, ami újabb hibákhoz vezethet. Fontos odafigyelni tehát, a megfelelő információs és hibaüzenetekre a rendszer és a felhasználó között.

- Minden ember különböző, más-más képességekkel rendelkezik. A tervezőnek nem szabad csak a saját képességeiből kiindulnia.

- Az emberi különbözőségéből adódik az interakció formájának megválasztási kérdése. Vannak, akik a képeket, mások a szöveges üzeneteket szeretik.

2.2.2 Tervezési elvek

Egy szoftver megítélése azon az interfészen múlik, amelyen keresztül a felhasználó használja a mögöttes rendszert.

- Az interfész felhasználó-orientált, és ne számítógép-orientált kifejezéseket és elveket alkalmazzon. A felhasználó nem gép, hozzá kell alkalmazkodni.

- A rendszer mutasson konzisztens (következetes, ellentmondásmentes) képet. Az utasítások, menük legyenek ugyanolyan kinézetűek. Legyen a rendszer viselkedése logikus.

- Ha egy utasítás ismert módon működik, akkor egy hasonló utasítás viselkedése megjósolható legyen. Ez a minimális meglepetés elve

- A rendszer legyen a felhasználói hibák ellen valamelyest ellenálló, és adjon lehetőséget ezen hibák helyreállítására. Ezek lehetnek „*vissza*” (undo) jellegű funkciók, destruktív akciók előtt jóváhagyás kérése, „*lágy*” törlések, stb.

- Legyen segítségnyújtási (Súgó) rendszer, on-line kézikönyvek, amelyek segítenek a szoftver működését, funkcióit megismerni.

- Különféle típusú felhasználók számára is legyenek megfelelő interakciós eszközök. Pl. látáskárosultaknak legyen elérhető a nagyobb betűméret, halláskárosultaknak pedig a „beszélő felület”.

2.3 Interaktív rendszerek

Interaktív rendszerek tervezése során két alapvető problémát kell megoldani:

- A felhasználó hogyan közöl információkat a számítógéppel?

- A számítógép hogyan közöl információkat a felhasználóval?

A koherens felhasználói interfész a felhasználói interakciókat és az információközlést és megjelenítést egységes keretbe integrálja.

2.3.1 Az interakciók fajtái

- Közvetlen (direkt) manipuláció

  A felületen objektumok, általában ikonok jelennek meg. A felületet a felhasználó valamilyen eszközzel akár közvetlenül is manipulálhatja. Előnye hogy gyors, intuitív és könnyen tanulható. Viszont nehéz implementálni.

- Menü kiválasztás

  A menüben megadott választási lehetőségek kizárják a felhasználók hibázási lehetőségét és a gépelési hibákat, viszont a gyakorlott felhasználók számára lassú és unalmas. Ha sok menüpont van, akkor bonyolulttá válhat.

- Kitöltendő form

  Az adatbevitel egyszerűen a billentyűzetről történik, könnyű tanulhatóság jellemzi, van ellenőrzési lehetőség, viszont nagy képernyőfelületet igényelhet és az is problémát okozhat, ha a mezők nem illeszkednek a felhasználó szándékaihoz.

- Parancsnyelv

  Alkalmazása hatékony és flexibilis, de megkívánja a parancsok ismeretét, hibakezelése többnyire gyenge.

- Természetes nyelv

  Tulajdonképpen egy nem szabványos parancsnyelv. Alkalmi felhasználók is tudják használni, könnyen megérthető. Hátránya, hogy sok gépelést igényel valamint, hogy a természetes nyelv gépi megértése bonyolult.

- Webes felület

  Manapság nagyon divatos, ugyanazan elemek alkalmazhatók rajta valamilyen szkriptnyelv segítségével. Probléma az adatforgalom, a felületen bevitt input továbbítása a szerver felé, majd a válasz visszajuttatása.

2.3.2 Webes felület

Manapság egy webes felület kialakításánál alapvető követelmény, hogy az szép, mégis egyszerű felépítésű, gyorsan reagáló legyen, és viszonylag kevés adatforgalmat használjon: ne töltődjenek újra az oldalak minden egyes lépésnél. Ennek megvalósítására ma már rendelkezésre állnak olyan ingyenesen felhasználható technológiák, mint például a Google Web Toolkit (GWT), ami az AJAX (Asynchronous JavaScript and XML) technikára épülve biztosít lehetőséget a funkciók háttérben (a böngészőben a lap frissítése nélkül) történő hívására, valamint a legkülönfélébb előre lekódolt webes komponensek használatára.

2.4 A felület ergonómiája, színei

A jó felhasználói felületet kényelmesen, kevés fáradtsággal, élvezettel lehet használni. Az ergonómia minőségi szempont. Olyan apró dolgokon is sok múlik, mint például a helyközök kihagyása vagy a hasonló funkciógombok csoportosítása, vagy annak elmulasztása. Manapság a kultúraspecifikusság, mint fogalom az informatikában is megjelent. Tervezéskor a különböző (nép)szokású emberekhez kell/lehet alakítani a felületet.
Színek megválasztásával is növelhetjük a felület harmóniáját, nemcsak az elrendezéssel. Bár ez már tervezési probléma, másrészt kognitív az emberek különbözősége miatt.
Statisztikák alapján 3-4, de maximum 7 szín alkalmazása még nem zavaró. Persze a színek megválasztása legyen következetes, igazodjanak a felhasználói interakciókhoz.

Néhány szabály a színek alkalmazására:

- egyszerre több pasztellszín használata szemrontó
- bizonyos színek kizárják egymást (komplementer jelenség)
- a színek megjelenítése eszközfüggő

3 Rendszerfejlesztés

A rendszerfejlesztés meghatározott elvek, módszerek, eljárások és eszközök tudatos, a rendszer céljának megfelelő alkalmazása, amelynek során a felhasználói igényeket, minőségi követelményeket kielégítő, az alaptevékenység hatékonyságát növelő, számítógéppel támogatott megoldást hozunk létre.
A rendszerfejlesztés a programozást (az implementációt) a szoftverfejlesztés egyik lépéseként kezeli.

A rendszerfejlesztés lépései

A rendszerfejlesztés fő lépései:

1. A megoldandó probléma meghatározása (vízió), felmérése a majdani felhasználók igényei alapján.(A követelmények feltárása.)
2. Valamely programtervezési módszerrel a programszerkezet megalkotása és a használandó eszközök kiválasztása. (Hardver platform, nyelvek, adatok stb.)(Az architekturális tervezés és tervezés lépése.)
3. Forrásprogram elkészítése (Implementálás).
4. A kész program tesztelése.
5. Dokumentáció készítése, mely tartalmazza a szoftvertervezés fázisaiban keletkezett adatokat (felhasználói leírás, igényfelmérés, programtervek, algoritmusok, forráskód, tesztelési jegyzőkönyvek stb.), fő célja a szoftver későbbi fejlesztésének elősegítése.
6. Üzembe helyezés
7. Üzemeltetés
8. Karbantartás
9. Evolúció
10. (Üzemen kívül helyezés)

4 Object Management Group

1.ábra – OMG logó

Az Object Management Group[1] (OMG) konzorciumot 1989-ben alapították azzal a céllal, hogy az elosztott, objektum-orientált rendszerek elterjedését elősegítsék. A tevékenységi köre az alapítás óta kibővült a modell-alapú tervezéssel, illetve a modell-alapú szabványok készítésével. Alapításkor 11 cég tartozott a konzorciumhoz, köztük az IBM, az Apple és a Sun, mára több mint 800 tagú szervezetté bővült. Az általa létrehozott szabványok nemzetközi elismertséget értek el.

Az OMG legismertebb fejlesztései a Common Object Request Broker Architecture (CORBA), amely a heterogén környezetben működő elosztott alkalmazások fejlesztését jelentősen megkönnyíti, illetve a Unified Modeling Language (UML), amely lehetővé teszi, hogy grafikus nyelv, illetve szintaxis segítségével dokumentáljuk és modellezzük az objektum-orientált rendszereket.

A 4.1 illetve 4.2 fejezetekben az OMG két olyan nagy fejlesztését mutatom be röviden, amely a szakdolgozatommal összefüggésben áll.

4.1 Az UML

Az Unified Modeling Language[2] (Egységesített Modellező Nyelv), röviden UML, az objektumorientált programozás szabványos specifikációs nyelve. Implementációtól független tervezést tesz lehetővé, a teljes szoftverfejlesztési életciklust támogatja. Az Object Management Group által fejlesztett modellező nyelv, mely segítségével egy rendszer megtervezése során grafikus jelölés használatával a rendszer absztrakt modellje írható le, vizualizálható. Az UML egy elemző és tervező „eszköz” (egy eszköz a fejlesztő kezében), mely diagramok segítségével mutatja meg a tervezett rendszer vázát, nem a teljes

implementációját. Az UML eszközrendszere nemcsak a modell elkészítéséhez szükséges eszközöket biztosítja, hanem a szoftver életciklusaihoz dokumentációt is szolgáltathat.

„UML is not a development method, that means it does not tell you what to do first and what to do next or how to design your system, but it helps you to visualise your design and communicate with others.”

Tehát az UML nem egy fejlesztési módszertan. Ez azt jelenti, hogy az UML nem azt szabja meg, hogy mit kell tenni először és mit kell tenni aztán, vagy, hogyan tervezzünk meg egy rendszert, hanem segít megjeleníteni a készülő rendszer/rendszerelem vázát nemcsak a programozó, hanem a végfelhasználó, megrendelő számára is érthető módon.

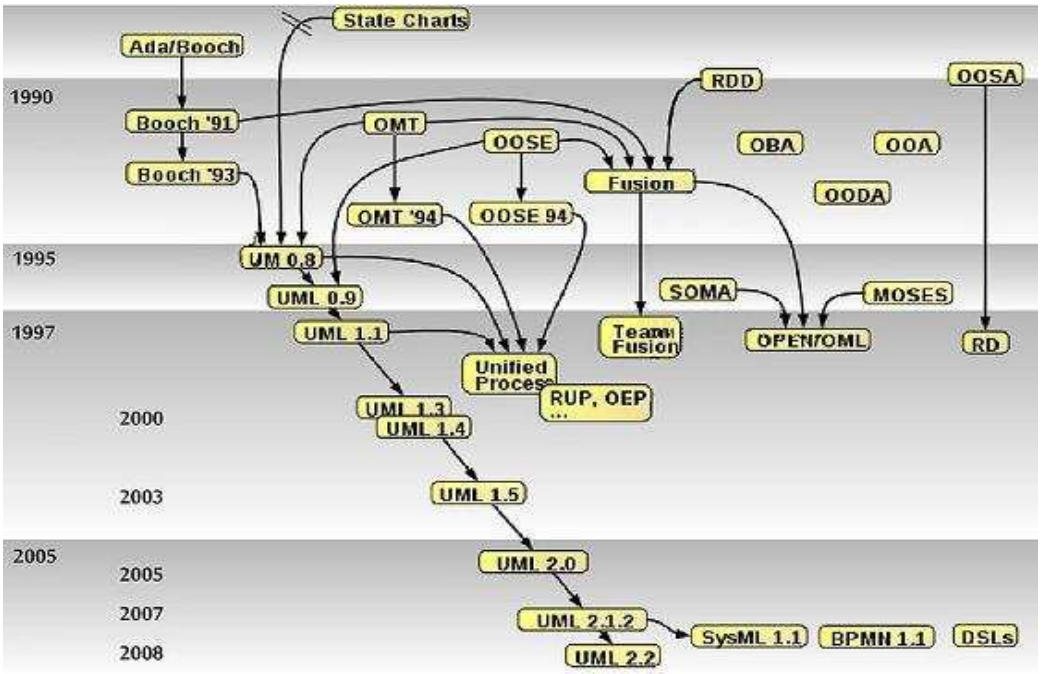
4.1.1 Az UML-ről

A szoftverkrízis, a szoftverfejlesztés válsága, miszerint egy hagyományos módszer (a 70-es évek rendszerfejlesztési modelljei) már nem képes igényeknek megfelelő, minőségi szoftver előállítására.
A 80-as évek elején az objektumorientált paradigma beköszöntével nemcsak a programozásban, de a tervezésben is új szemlélet következett. (Az objektumorientált tervezés lényege, hogy a valós világot modellezzük. A valós világban egyedek élnek, ezeket az egyedeket osztályozzuk, csoportba foglaljuk valamilyen közös tulajdonság alapján.)

A 90-es évek elejére már több különböző módszertan létezett(például az OMT – Object Modeling Technique -, amely James Rumbaugh és az OOSE – Object-oriented software engineering-, amely Ivar Jacobson nevéhez fűződik). Az UML, leginkább Grady Booch, Rumbaugh és Jacobson módszereire épül, ugyanakkor jelentőségét tekintve túlmutat azokon. Már az UML nyelvre 1995-96-ban megfogalmazott első javaslatok megjelenése nagy szakmai visszhangot váltott ki, az OMG azonban szinte ezzel egy időben megkezdte a szabványosítását, a szabványosítási folyamathoz példátlan módon a világ összes nagy szoftvergyártója csatlakozott, köztük a Microsoft, az IBM, az Oracle, a HP, a Sun, a DEC, a Compaq.

„Az egységesítés

Három élvonalbeli fejlesztő (Grady Booch, Ivar Jacobson és James Rumbaugh) felismerte az igényt egy egységes módszertan kidolgozására, így összefogtak, hogy az addig megszületett metodikák előnyös tulajdonságait kiemeljék, és azok alapján egy közös koncepciót dolgozzanak ki. Az ő munkásságuknak köszönhetően, több más szerző módszertanának és tapasztalatainak felhasználásával született meg 1997-ben az UML (Unified Modeling Language) modellezőnyelv, illetve 1998-ban a RUP (Rational Unified Process) módszertan (később több, UML-re építő módszertant is kidolgoztak a szoftvercégek).” (Molnár Ágnes, Objektumorientált alkalmazásfejlesztés)



2. ábra – Az UML története

4.1.2 Szükséges alapfogalmak

- Osztály: Az osztály nem más, mint egy absztrakt adattípus, ami megvalósítja az egységbezárást. Az osztály valójában az azonos adattípussal rendelkező elemek leírására szolgál, azok tulajdonságait (attribútumok) és viselkedésüket (metódusok) definiálja.

- Objektum**:** Az osztály egy példánya. Minden objektum rendelkezik állapottal, amit az attribútumainak értékei határoznak meg, viselkedését a metódusain keresztül gyakorolja. Az objektum identitása az egyediségre utal, minden objektum különböző.

- Öröklődés: Egy osztály létrehozásához, egy már meglévő osztályt is felhasználhatunk. Az új osztály örökli az ősosztály attribútumait, metódusait, de e mellett lehetősége van új attribútumok, metódusok definiálására, a meglévő metódusok felüldefiniálására.
    - Általánosítás: Több osztály közös részstruktúráját és –viselkedését egy közös ősosztályba helyezzük.
    - Specializálás: Származtatott osztályokat hozunk létre, melyek finomítják az ősosztályt.

4.1.3 Az UML diagramcsoportjai

A diagramok olyan gráfok, amelyek csomópontjai elemeket, élei az elemek közötti kapcsolatokat képviselik. A különböző diagramok közös elemeket is tartalmazhatnak, hiszen ugyanazt a rendszert ábrázoljuk többféle megközelítésben.

Az UML elemeinek csoportra bontása:

- Statikus
- Viselkedési (dinamikus)

A statikus diagramoknak öt fajtáját különböztetjük meg, ezek a következők:

1. Osztálydiagram (class diagram)

   Az osztályok és interfészek, valamint azok együttműködésének és kapcsolataiknak ábrázolására szolgál. Az osztályok jelölése olyan téglalapokkal történik, amelyek három részből állnak: felső harmadában az osztály neve, középen az osztály attribútumai, alul pedig az osztályhoz tartozó metódusok szerepelnek. Az attribútumok láthatóságát a +(public), -(private), #(protected) jelek jelzik.

2. Objektumdiagram (object diagram)

   Az osztálydiagram elemeinek pillanatnyilag létező példányait, azok kapcsolatait szemléleti.
   Az objektumokat két részből álló téglalapok reprezentálják, amelyek felső felében az objektum nevét és osztályát, alsó felében attribútumait tároljuk (aktuális értéküket is feltüntetve).

3. Komponensdiagram (component diagram)

   A komponensek egymáshoz való viszonyát fejezi ki. Ha egy komponens osztályok, interfészek és köztük lévő kapcsolatok együttese, ez az ábrázolásmód szoros kapcsolatban áll az osztály-diagrammal. A komponenseket téglalapokkal jelöljük, bal oldalukon két kis „fogacskával". Szaggatott nyilak mutatják, hogy mely komponens melyiknek szolgáltat valamilyen információt, eljárást, stb.

4. Telepítési diagram (deployment diagram)

   A futás közben igényelt erőforrásigényt, és a csomópontokon működő komponenseket ábrázolja. Az erőforrásokat téglatestek szemléltetik.

5. Használati eset (use case) diagram

   A valós rendszer szereplőit, ezek kapcsolatát és tevékenységeit mutatja be. A rendszer szervezése, viselkedésének leírása és ellenőrzése szempontjából létfontosságú!

Az UML diagramok másik csoportja, a dinamikus (más néven viselkedés-) diagramok az objektumok egymásra hatását, kommunikációját, üzenetváltásait mutatják be. Négy típus sorolható ide:

1. Szekvencia diagram (sequence diagram)

   Az üzenetek küldésének és fogadásának időrendi sorrendjét határozza meg, a használati esetekből kiindulva. Az üzenetváltás szereplőit nagy téglalapokkal jelöljük az ábra tetején. Az idő múlását a függőleges tengely szemlélteti, a cselekvéseket a

szaggatott vonalú tengelyeken lévő, a cselekvés időtartamával arányosan hosszú téglalapok. Az üzenetek a küldőtől a fogadóig húzott nyíllal szerepelnek az ábrán.

2. Együttműködési diagram (collaboration diagram)

   Az üzeneteket váltó objektumok kapcsolatát, és az üzenetváltás struktúráját ábrázolja. A szekvencia diagramból egyszerű algoritmus alapján megkapható.
   Az üzenetküldőket és –fogadókat egyszerű ellipszisek, az üzenetek nyilak jelképezik, rajtuk az üzenet küldésének relatív időpontja (az üzenet sorszáma) és az üzenet neve szerepel.

3. Állapot- vagy állapotátmenet diagram (state-chart)

   Rendkívül fontos az eseményorientált viselkedés vizsgálatánál. A rendszer állapotait modellezi, amelyek különböző ingerek hatására mennek át egyik állapotból a másikba. Informatikai jellegű diagram, kevésbé intuitív.

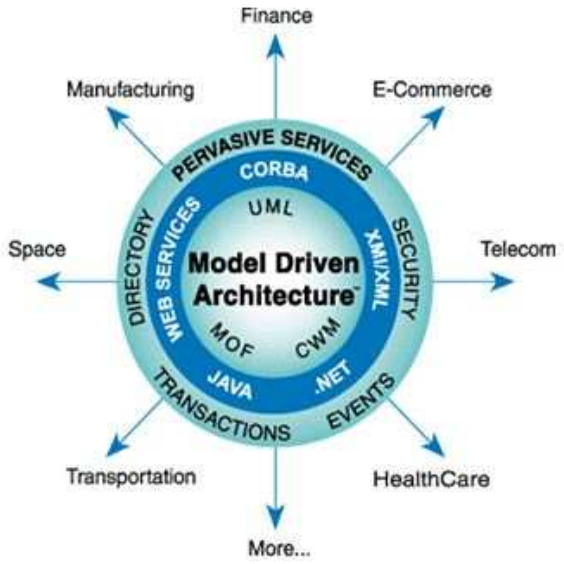
4. Tevékenységdiagram (activity diagram)

   Speciális állapotdiagram, amely a végrehajtandó tevékenységek folyamatát mutatja. Nagy jelentőségű az objektumok vezérlési folyamatainak tervezésénél.

4.2 A Modellvezérelt Architektúra

A Modellvezérelt Architektúra[3] (Model Driven Architecture, röviden MDA) egy alkalmazásfejlesztési megközelítés. Egy MDA specifikáció egy platformfüggetlen UML modellből (Platform Independent Model, röviden PIM) és egy vagy több platform specifikus modellből (Platform Specific Model, röviden PSM) áll. Az MDA szemlélet szerint először a vizsgált rendszer funkcionalitására és viselkedésére kell koncentrálni, eltekintve a technológiai környezettől, amelyben implementálásra kerül. Ily módon egy alkalmazási rendszer teljes modelljét csak egyszer kell megalkotni.
Az MDA elkülöníti az alkalmazásarchitektúrát a rendszerarchitektúrától. Az alkalmazás-architektúra az alkalmazás funkcionális céljait specifikáló komponenseket és kapcsolatokat tartalmazza. A rendszerarchitektúra pedig az alacsonyabb szintű komponensekből és

kapcsolatokból áll, amelyek az alkalmazásarchitektúra végrehajtását teszik lehetővé. Ez az elkülönítés az MDA alapvető eleme.



3. ábra - MDA

Egy PIM tulajdonképpen a teljes alkalmazás specifikációja, amely platformfüggetlen. Egy PIM egy PMS-be képződik le, amely a rendszerarchitektúra infrastruktúráját biztosítja. Ez a leképezés a PIM implementációját jelenti, azaz a végrehajtható alkalmazás generálását. A PIM lehetővé teszi számunkra a megoldás magas absztrakciós szinten történő vizuális modellezését.

Amikor egy új technológia megjelenik, szükségtelen az alkalmazás újraírása, csak újra kell generálni, azaz elvégezni a modell leképezését az új környezetbe. Az üzleti logika független az implementálási technológiától. A szoftverfejlesztés ezek után tehát nem más, mint transzformáció: PIM → PSM, PSM → kód. A szoftverfejlesztés automatizálható.

Több, manapság népszerű technológiai platform, mint a CORBA, J2EE vagy .NET számára lehetséges PIM→PSM leképezést definiálni. Az OMG szabványai, a MOF (Meta Object Facility), XML (Extensible Markup Language), XMI (XML Metadata Interchange) és CWM (Common Warehouse Metamodel) együttműködése biztosítja, hogy az MDA teljes szoftverfejlesztési megközelítés legyen.

5 A fejlesztés háttere

Ebben a fejezetben a fejlesztés során alkalmazott technológiákról és eszközökről adok egy rövid, általános leírást.

5.1 Alkalmazott technológiák

5.1.1 XML

Az Extensible Markup Language[4], vagy közismertebb nevén az XML a Word Wide Web Consortium[9] (W3C) terméke. Az XML, mint ahogy a HTML is, az SGML-ből leszármaztatott jelölőnyelv. Annak ellenére, hogy mindkettőnek ugyanazon nyelv az ősatyja, mégis alapjaikban különböznek. Míg a HTML megadott elemhalmazból épül fel, addig az XML-ben saját magunk hozzuk létre az egyes elemeket.

Az SGML-ből (Structured Generalized Markup Language vagy magyarul Általános Jelölőnyelv) lehet saját elemkészlettel rendelkező dokumentum típus leírást létrehozni, s ezek létrehozásának legismertebb terméke az 1991-ben létrehozott HTML volt.

Mivel az SGML-ben elég tág korlátok közé vannak szorítva az egyes elemek létrehozásának feltételei, nehézkes lenne olyan programot írni, amely tökéletesen fel tudná dolgozni ezen dokumentumokat. A HTML nyelvben viszont kezdett nehézkessé válni a bonyolultabb felépítésű adatok tárolása illetve megjelenítése. Ezért volt szükség egy, a webre optimalizált jelölőnyelv létrehozására, amelyet a W3C fejlesztőcsapata alkotott meg 1996-ban, Kiterjesztett Jelölőnyelv néven (Extensible Markup Language). Mivel az XML-ben az elemek létrehozására szűkebb szabályok vonatkoznak, mint az SGML-re, ezért könnyebben tanulható, olvasható és implementálható.

Ahhoz, hogy egy XML dokumentum helyes legyen, a következő követelményeknek kell megfelelnie:

- *Jól formázottság*. Egy helyesen formázott XML dokumentum megfelel minden szintaktikai szabálynak. Például ha egy nem üres elem rendelkezik nyitó tag-gel, de nem rendelkezik záró tag-gel, akkor nem jól formázott. Az a dokumentum, ami nem

jól formázott, nem tekinthető XML-nek. Az elemzőnek meg kell tagadnia a feldolgozását.

- *Érvényesség*. Egy érvényes dokumentum olyan adatot tárol, ami megfelel a felhasználó által definiált tartalmi szabálynak, ami leírja a helyes adat értékeket és helyeket. Például ha a dokumentum egy elemének olyan szöveget kell tartalmaznia, ami egész számként értelmezhető, és e helyett a szöveg "helló", üres, vagy más elemeket tartalmaz, akkor a dokumentum nem érvényes.

5.1.2 XSLT

Az Extensible Stylesheet Language Transformations[5] (röviden XSLT) egy XML-alapú fájlformátumot illetve a hozzá tartozó feldolgozó rendszert jelöli. Az XSLT-feldolgozót XML dokumentumok más formátumra alakításához használják. A konverzió során az eredeti fájl megmarad, s annak tartalma alapján létrejön egy új fájl a célformátumban.

A keletkező dokumentum formátuma lehet többek között XML, HTML, XHTML, PDF vagy egyszerű szöveg például. Az XSLT-t gyakran használják különböző sémájú XML dokumentumok közötti konverzióra, dinamikus weboldalak létrehozására és PDF dokumentumok generálására.

Az XSLT feldolgozási modell részei a következők:

- egy vagy több XML *forrás*dokumentum
- egy vagy több XSLT *stíluslap* modul
- az XSLT sablonfeldolgozó szoftver
- egy vagy több *cél*dokumentum

Az XSLT-feldolgozó tipikus esetben két inputfájlt olvas be, nevezetesen egy forrásfájlt és egy XSLT stíluslapot, majd egy outputfájlt hoz létre. Az XSLT stíluslap tartalmazza a konverziós szabályokat a feldolgozó számára. A forrásfájl jellemzően XML formátumú, de a specifikáció nem zár ki más formátumokat sem. Az XSLT stíluslap formátuma XML-alapú.

A konverziós szabályok feldolgozása:
Az XSLT nyelv deklaratív, ilyen tekintetben hasonlít a funkcionális programnyelvekre. A konverziós szabályok azt írják le, hogy a feldolgozónak hogyan kell kezelnie az XML-fa adott XPath kifejezésre illeszkedő csomópontjait. Jelen kontextusban az XPath az XML

dokumentum szerkezetének bejárására szolgáló specifikáció. A feldolgozó a következő algoritmus szerint működik: Beolvassa a stíluslap szabályait és létrehoz a forrásdokumentumból egy fa adatszerkezetet, egy úgy nevezett forrásfát. A forrásfát a gyökerénél kezdi el feldolgozni, s minden csomópontra végrehajtja a rá illeszkedő XPath kifejezésekhez tartozó utasításokat. Az utasítások jellemzően vagy csomópontokat hoznak létre a célfában, vagy tovább olvasnak a forrásfában. A feldolgozó végül a célfából előállítja kimenetet.

5.1.3 HTML

A HTML dokumentum - formátumot tekinthetjük az ún. hyper-text egyik megvalósítási formájának is. A HTML dokumentum egy olyan szövegfájl, amely a szövegen kívül tartalmaz formázóutasításokat – ún "HTML-tag"-eket -, valamint megjelenítendő objektumokra történő hivatkozásokat is. Ezek a formázóutasítások befolyásolják a dokumentum megjelenítését, kapcsolatait. Ezeket az utasításokat a böngészőprogram értelmezi és végrehajtja. Ezen okból a formázóutasítás mindig megelőzi azt a részét a dokumentumnak, amelyre vonatkozik. A dokumentumkészítéshez használható HTML utasítások köre állandóan bővül, a nyelv fejlődik. A szabványosítás csak lassan követi a fejlődést. Ezért nem minden böngészőprogram tudja a HTML utasítások mindegyikét értelmezni. Egy böngésző, ha számára értelmetlen utasítással találkozik, akkor kihagyja, így nem okoznak problémát az újabb keletű - még szabványosítatlan - utasítások a régebbi kiadású WWW-böngészőknek sem. Sajnos a fentiek miatt ugyanazt a dokumentumot két különböző program nem biztos, hogy azonos formában fogja megjeleníteni. A HTML-ben mégis az a nagyszerű, hogy nagymértékben megközelíti a platformfüggetlenséget. Egy HTML dokumentum - ha nem is azonos módon, de - mindenki számára megtekinthető.

5.1.4 AJAX

A webre fejlesztők közös nyelve a HTML, ennek segítségével tudják megfogalmazni gondolataikat. Azonban a HTML csak korlátozott lehetőséget nyújt a felhasználói interakcióra, konkrétan hivatkozások, gombok és űrlapok formájában. Sok fajta technológiai megoldás született ennek kiküszöbölésére, a felhasználói élmény fokozására a böngészőn belül. Mindegyiknek megvan a saját előnye és hátránya.

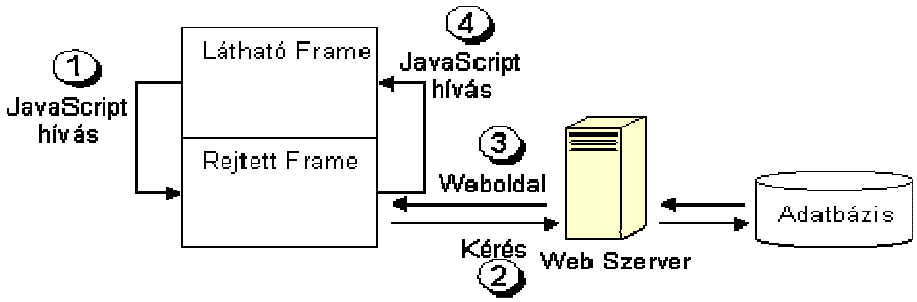
Az AJAX[6] technológia egy betűszó, az "Asynchronous JavaScript and XML" rövidítése, azaz aszinkron JavaScript és XML. Az aszinkronitás, amivel eddig a web programozóknak nem kellett törődnie, azaz, hogy kérés a háttérben fut. A JavaScript ebben a kontextusban azt jelenti, hogy az Ajax "programozási nyelve" a JavaScript. Tehát ez egy böngésző oldali technológia. Az XML egy mostanában futótűzként terjedő technológia, amivel egyszerűen, több rendszer között is viszonylag könnyen értelmezhető, szabványos módú kommunikációs üzenetküldés valósítható meg.

Az AJAX egy új funkció a böngészőkben, amely segítségével a háttérben kérés küldhető a szerver felé. Mindezt anélkül, hogy frissítenénk az oldalt vagy elnavigálnánk onnan. A kérésre kapott választ JavaScript-ből feldolgozhatjuk és az oldal egyes részeit frissíthetjük.

5.1.4.1 Ajax kommunikáció

A böngésző és a kiszolgáló közötti párbeszéd a HTTP (HyperText Transfer Protocol) segítségével valósul meg, melynek során kérések és válaszok tömkelege vándorol a weben. Alapesetben egy kérés csak akkor generálódik, amikor a felhasználó valamely módon ezt kiváltja.

5.1.4.2 A rejtett keret (hidden frame) technika



4. ábra – Hidden Frame technika

Az alapötlet egy olyan frameset készítése, mely egyik tagjának szélessége és magassága is egyaránt 0 pixel, ebből következően nem látszik, és kizárólag a szerverrel történő kommunikációra használjuk.

A kommunikáció minden esetben az 4. ábrán látható 4 lépéses minta alapján zajlik. Az első lépés a látható keretben történik, ahol a felhasználó, mit sem sejtve arról, hogy létezik a másik

keret, a szokásos módon használja az oldalt, és olyan műveletet kezdeményez, aminek a kiszolgálásához további adatokra van szükség a szerverről.

Ebben a pillanatban egy JavaScript hívás történik a rejtett keretre. Ez a hívás lehet egy egyszerű átirányítás, de akár egy űrlap adatainak elküldése is, a lényeg, hogy bekövetkezik a folyamat második lépése, a HTTP kérés.

A harmadik lépés a szervertől kapott válasz fogadása. Mivel frame szerkezetben dolgozunk, a válasz nyilván egy újabb weboldal lesz, ami tartalmazza a szervertől kért adatokat, és azt a JavaScript kódot, ami átadja azokat a látható keretnek.

Végül, a negyedik lépésben az előbb említett JavaScript kód lefut. Tipikusan ez úgy történik, hogy a visszaküldött oldal *onload* eseménykezelőjében hívunk meg egy függvényt a látható keretben, és átadjuk a megkapott adatokat, amelyek sorsáról már a látható keret feladata lesz gondoskodni.

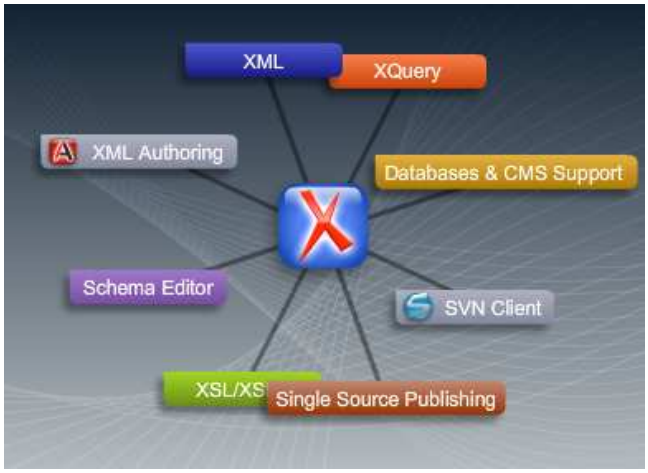
5.2 Alkalmazott eszközök

Ebben az alfejezetben a fejlesztés során alkalmazott eszközöket mutatom be. A NetBeans-t már tanulmányaim során használtam, alkalmazása kézenfekvő volt. Az Oxygen-t, mint XML szerkesztőt azért választottam, mert minden általam alkalmazott webes technológiát támogat, mindemellett kategóriájában a legjobb ajánlásokkal rendelkezik.

5.2.1 NetBeans IDE 6.5

Az NetBeans egy Java nyelven alapuló, ezáltal platformfüggetlen integrált fejlesztőkörnyezet (Integrated Development Environment, IDE). Az elérhető pluginok segítségével sokféle (nem csak Java nyelvű) alkalmazás elkészítésére használható. Az UML plugin letöltése után bármely, a projektünk számára szükséges diagramot elkészíthetünk vele.

5.2.2 Oxygen XML Editor



5. ábra – Oxygen XML Editor

Az Oxygen XML Editor[7] – stílusosan <oXygen/> - egy Java nyelven írt platformfüggetlen XML szerkesztő (Van olyan verziója amely Eclipse IDE beépülő moduljaként futtatható.) Habár maga az alkalmazás fizetős, de 31 napos próbaverziója egy e-mail cím megadásával letölthető. Ez az egyetlen eszköz, amely ismeri az összes XML sémanyelvet. Az XSLT és XQuery támogatás mellett erőteljes hibakeresőt és profilelemzőt biztosít, alkalmazható az összes XML-technológiához.

5.2.3 IIS

Az általam használt webszerver az IIS[8], Internet Information Services, korábbi nevén Internet Information Server, ami a Microsoft cég terméke. Feladata hálózati szolgáltatások megvalósítása. Felépítése szerint rendszer szinten regisztrált szolgáltatásokból és beállító-vezérlő konzol részekből áll. A szolgáltatások egyenként látnak el egy-egy szolgáltatást, a konzol pedig ezeket kezeli.

A szolgáltatások között található HTTP vagyis a web kiszolgáló, FTP kiszolgáló, illetve levelezőszerver. Az IIS a Microsoft szerver rendszereinek alapja. Az IIS a világon a második legnépszerűbb web szerver a piacvezető Apache HTTP Szerver után.

A szoftver egy egyszerűbb, szolgáltatásokban korlátozott verziója megtalálható a Windows operációs rendser összetevői között.

Ahhoz, hogy az IIS-t mint webszervert tudjuk használni, installálni kell. Telepítéskor létrejön egy *inetpub* nevű könyvtár, amelynek ha a *wwwroot* alkönyvtárába másoljuk a szükséges fájlokat, biztosítja a kapcsolatot.

6 A fejlesztés

Gyakorlati feladatként egy HTML alkalmazás UML diagramok alapján történő, automatizált létrehozását választottam.

Lépések:

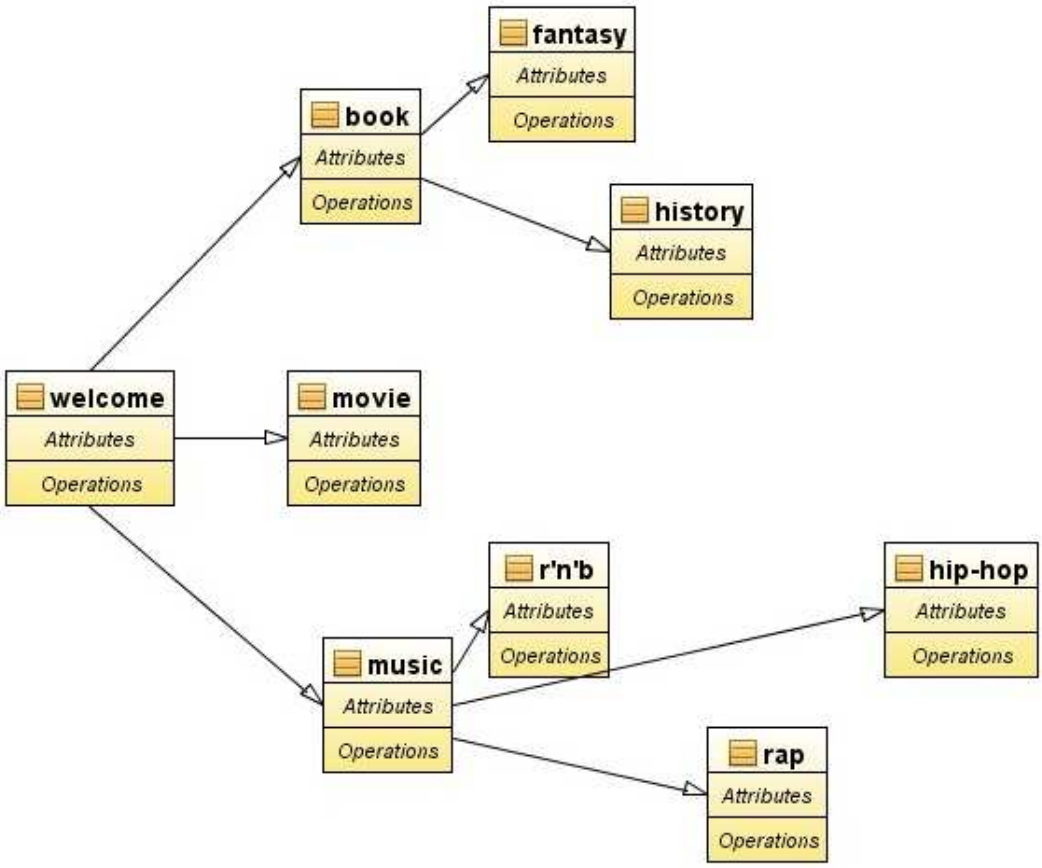
- Vízió
- UML-diagram
- Az osztálydiagram XMI leírásának értelmezése
- Az osztálydiagram XMI fájljára alkalmazott XSLT bemutatása
- Az osztálydiagramból generált HTML váz áttekintése
- Egy weblap hogyan képes információt nyerni XML fájlból AJAX technológia segítségével
- A kész webes interfész

6.1 A Vízió

A webes alkalmazás, amelyen keresztül bemutatom az általam kitalált generálási technikát, egy egyszerű a kedvenc könyveimet, zenéimet és filmjeimet bemutató weboldal, amely AJAX technológiát használ a megtekinteni kívánt információk megjelenítésére.

6.2 UML-diagram

A diagram az 5.2.1 fejezetben már bemutatott NetBeans fejlesztőkörnyezet segítségével lett megszerkesztve. Célom egy olyan fa szerkezet létrehozása volt, amelyben logikusan lehet lépkedni a műfajok és azok kategóriái között.



6. ábra - Osztálydiagram

A célom egy webes felület létrehozása, így ebben az esetben az osztálydiagramot speciálisan használtam fel úgy, hogy az ábrában minden egyes osztály egy-egy html oldalt képvisel, az osztály-alosztály kapcsolat pedig az oldalak közötti kapcsolódást (navigációt) jelenti. Így például a *welcome* osztály szülőosztálya a *book, movie* és *music* osztályoknak, ez a HTML oldalaknál azt fogja jelenteni, hogy a *welcome* lapról továbbjuthatunk a *book, movie* illetve *music* oldalakra.

Ha a NetBeans-ben létrehozunk egy UML projektet, akkor a projektben szereplő diagramok leírása több fájlban tárolódik. (A 6.5 verziószámú NetBeans IDE UML pluginja használja pontosan ezt a fajta diagram tárolást és leírást.) Az *etd* kiterjesztésű XMI fájl az, ami a projektben szereplő diagram összes elemének modellbeli szerepét, tulajdonságát leírja(mi a neve,van-e alosztálya,milyen attribútumai és műveletei vannak). XML Metadata Interchange, azaz az XMI egy OMG (Object Management Group) szabvány metaadat cserére XML-en keresztül, használható bármilyen metaadatra ami kifejezhető MOF-ban.

6.3 Az osztályok.etd fájl bemutatása

```
<?xml version="1.0" encoding="UTF-8"?>
<XMI xmlns:UML="omg.org/UML/1.4" xmi.version="2.0">
        <XMI.header>
                <XMI.metamodel xmi.name="UML" xmi.version="1.4"/>
                ...
        </XMI.header>
        <XMI.content>
            <UML:Project xmi.id="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B" mode="Design"
                    defaultLanguage="UML" projVersion="6.1.4.837" name="Favorites">
                        <UML:Element.ownedElement>
                                <UML:Class ...
                                ...
                                </UML:Class> ...
                        </UML:Element.ownedElement>
            </UML:Project>
        </XMI.content>
</XMI>
```

kódrészlet az osztalyok.etd fájlból

Mivel az XMI egy speciális XML fájl, így első sora XML deklaráció, ami a használt XML szabvány verzióját és a fájl karakterkódolását adja meg. A második sor tartalmazza a használt szabvány verzióját és tartalmaz egy *xmlns* attribútumot, ami a fájlban használt névteret rögzíti. Az <XMI.content></XMI.content> tag-ek között helyezkedik el az UML diagram leírása. Az <UML:Project> tag attribútumai többek között a projekt azonosítóját és nevét adják meg. Ezen belül találhatóak az <UML:Class></UML:Class> tag párok amelyek az egyes osztályokról tartalmaznak információt.

```
<UML:Class xmi.id="DCE.B6B29BAB-3E94-C9D5-C1EF-619887D8CE6F" name="book"
        owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
        specialization="DCE.7B3DC9F8-6E61-A537-3DA2-409A009E9194">
                <UML:Element.presentation> </UML:Element.presentation>
                <UML:Classifier.generalization>
                        <UML:Generalization xmi.id="DCE.1498D0F7-5F4D-F69E-EAE0-C79730CE6ACE"
                                specific="DCE.B6B29BAB-3E94-C9D5-C1EF-619887D8CE6F"
                                general="DCE.C266A1D3-8619-378E-9775-819F86EB25C1">
                                        <UML:Element.presentation> </UML:Element.presentation>
                        </UML:Generalization>
                        <UML:Generalization xmi.id="DCE.0AC89F61-74F9-B325-FD20-CA10077BCBE0"
                                specific="DCE.B6B29BAB-3E94-C9D5-C1EF-619887D8CE6F"
                                general="DCE.32F692B0-2A64-6FE0-8325-5B07BCBC28C8">
                                        <UML:Element.presentation> </UML:Element.presentation>
                        </UML:Generalization>
                </UML:Classifier.generalization>
 </UML:Class>
```

kódrészlet az osztalyok.etd fájlból

A fenti kódrészlet a *book* osztály és a hozzá kapcsolódó generalizációk leírása. Az <UML:Class> attribútumaként jelenik meg az osztály neve (name) és az osztály azonosítója (xmi.id). Ahogy az a korábban szereplő 6. ábrán látható, a *book* osztálynak 2 alosztálya van, ez itt a <UML:Generalization> </UML:Generalization> tag-ek közt jelenik meg, ahány alosztály, annyi tag. A tag-nek 3 attribútuma van: az *xmi.id*, ami az osztályok közötti (osztály-alosztály) kapcsolat azonosítója, a *specific*, ami a kapcsolat szülőosztályának (aktuálisan mindig annak az osztálynak az azonosítója, amelyet vizsgálunk), és a *general*, ami a kapcsolódó alosztály azonosítója.

6.4 Az osztalyok.etd-re alkalmazott XSLT fájl bemutatása

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:UML="omg.org/UML/1.4"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"   version="2.0"
>
```

kódrészlet az osztályok.xsl fájlból

Az XSLT fájl elején elhelyezkedő deklarációban a használt névterek és a verziószám szerepelnek. Az xmlns, az XML namespace rövidítése.

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

Az XSLT névterére mutat. Az XSLT-feldolgozónak ismernie kell az XML névterek mechanizmusát ahhoz, hogy felismerje a kölünböző névtérből származó elemeket és attribútumokat. Az XSLT névtér elemeit csakis az XSLT dokumentumon belül ismeri fel a feldolgozó. A kódban az *xsl:* prefix jelzi a névtér elemeit.

```
xmlns:UML="omg.org/UML/1.4"
```

Ahhoz, hogy a feldolgozó a forrásfájl névtereit is felismerje, a forrásfájlban alkalmazott névtér deklarációkat itt is szerepeltetni kell.

```
"version="2.0"
```

Különösen fontos megadni a használt XSLT technológia verzióját, mivel több HTML oldal létrehozására az 1.0 nem ad lehetőséget.

„Az XSLT-feldolgozót XML dokumentumok más formátumra alakításához használják.” Ezért a fájlban rögzíteni kell, hogy milyen formátumot is szeretnénk létrehozni:

```
<xsl:output method="html" indent="yes" name="html"/>
```

Az <xsl:output> az XSLT névtér eleme, attribútumaival definiálja a kimenetet. A *method* attribútum adja meg a kimeneti formátumot.

```
<xsl:result-document href="{$filename}" format="html">
```

Az osztályok feldolgozásánál ez az XSLT névtérbeli elem biztosítja, hogy az osztály nevének megfelelő HTML formátumú kimenet létrejöjjön.

Ezek után következik az a rész, amely már valóban az .etd fájl feldolgozásával foglalkozik. Az <xsl:template> *match* attribútumaként megadható egy elérési útvonal, amely az UML diagramot leíró fájl azon csomópontjára (a tag-ek egymásba ágyazott rendszere a leíró fájlban) mutat, amelyet a programozó kiindulópontnak jelöl ki. Mivel a célom az, hogy minden egyes diagrambeli osztályból egy-egy HTML oldalt transzformáljak, célszerűen az

osztályokat tartalmazó csomópontot választom kiindulópontnak. Majd egy *<xsl:for-each select="//UML:Class">* ciklussal egyenként dolgozom fel az osztályokat.

6.4.1 Osztály-alosztály viszonyok feltárása

Az egyes osztályok feldolgozásánál változókban tárolom a nevet - amely a későbbi HTML oldalon szerepelni fog, mint az oldal címe - és az xmi.id azonosítót(amiről biztosan lehet tudni, hogy egyedi). Ha van az osztálynak alosztálya, akkor egy ciklussal : <xsl:for-each select="UML:Classifier.generalization/UML:Generalization"> bejárva azokat visszakeressük az xmi.id alapján az alosztály nevét, amelyet a HTML oldalon az alosztályra mutató linknek használtam fel.

Az XSLT dokumentumban szerepel még egy javascript kód, amely csupán a kényelem kedvéért az előzőleg megtekintett HTML oldalra linkel vissza.

Vissza

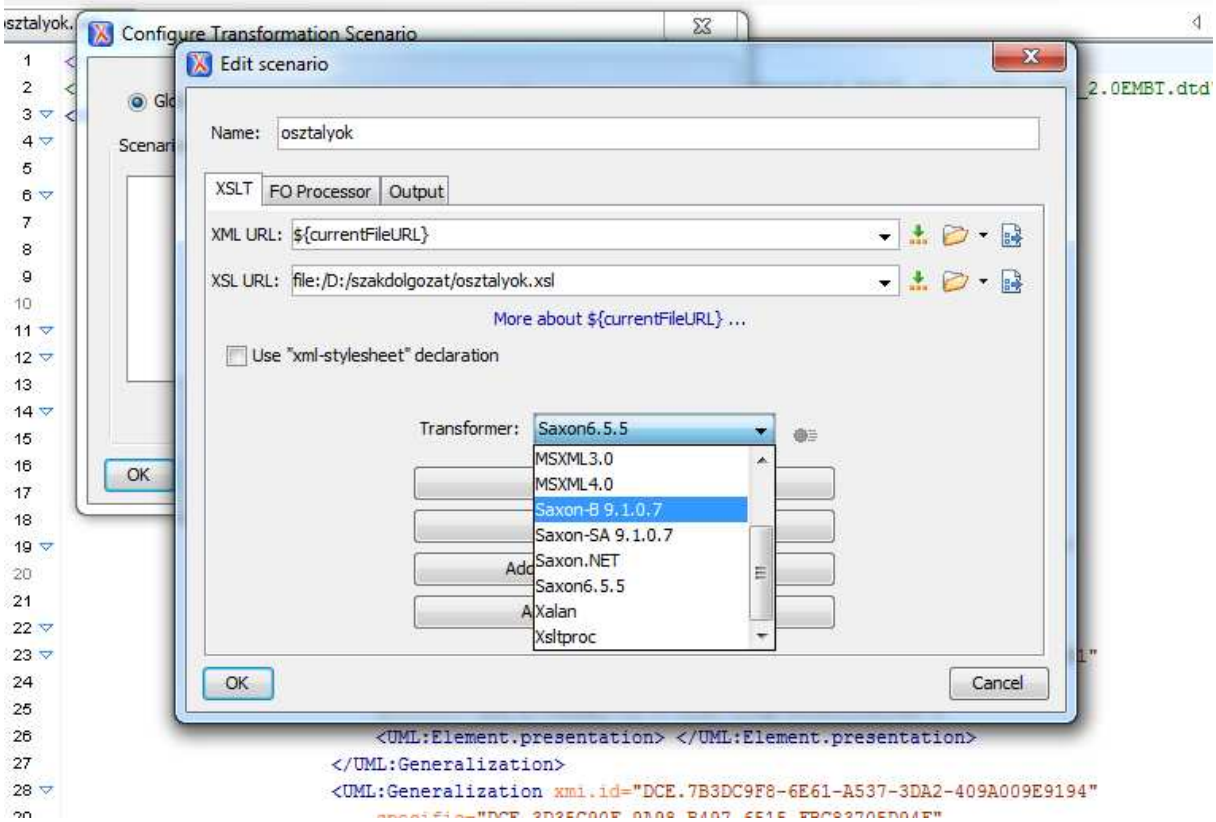
6.4.2 Formázás

Az XSLT fájlok esetén lehetőség van arra, hogy a létrehozni kívánt HTML oldalt formázzuk. Két módszer áll rendelkezésre:

- Stílusdefiníciók helyezhetőek el az XSLT fájl <body></body> részében <style></style> tag-ek között.
- Külső CSS (Cascading Style Sheets) fájl alkalmazásával. Ekkor az XSLT fájl <head></head> részében kell elhelyezni egy linket, ami arra a .css kiterjesztésű fájlra mutat, amely a stílusra vonatkozó információkat tartalmazza.
  Ez a link az osztályok.xsl-ben így néz ki:
  <link href="osztalyok.css" rel="stylesheet" type="text/css"/>
  Ha a .css fájlt így adjuk meg, akkor annak azonos mappában kell lennie azzal a html oldallal, amelyben hivatkozunk rá.

6.5 Az XSLT fájl alkalmazása, az összerendelés

Ahhoz, hogy az osztályok.etd-ből az osztályok.xsl segítségével előálljanak a kívánt HTML oldalak, össze kell rendelnünk őket. A 6.2.2 fejezetben bemutatott Oxygen XML szerkesztőben megadhatjuk azokat a beállításokat, amelyek szükségesek.



7. ábra – Beállítások az Oxygen XML Editor-ban

Az .etd fájl transzformációs beállításainál meg kell adni a használni kívánt XSLT fájl URL-jét. A feldolgozók közül alapértelmezett a Saxon 6.5.5 (Maga a SAXON csomag egy az XML dokumentumok feldolgozására szolgáló eszközök gyűjteménye.) Ez a feldolgozó csak az XSLT 1.0 verzióját támogatja, mivel az osztályok.xsl 2.0 verziójú, így ez nem megfelelő. A XSLT processzorok közül a Saxon-B 9.0.1.7 ismeri az XSLT 2.0-ban bevezetett, általam használt elemeket.

A beállítások után, a  (*Apply Transformation)* gomb hatására abban a mappában, ahol az .etd fájlunk található, létrejön egy *output1* nevű mappa, amelyben benne lesznek az .etd fájl alapján az .xsl által létrehozott .html kiterjesztésű fájlok.

6.6 A HTML váz és a CSS fájl

Az osztálydiagramban (3. ábra – Osztálydiagram) szereplő osztályokkal megegyező nevű, és az osztály-alosztály kapcsolatokkal megegyező linkelésű HTML oldalak formázása külső CSS fájllal lett megoldva. Ahhoz hogy az .xsl fájlban megadott <link href="osztalyok.css" rel="stylesheet" type="text/css"/> hivatkozást a HTML oldal alkalmazza, az osztályok.css-t ugyanabban a mappában kell elhelyezni ahol a forrásfájlok találhatóak. A külső CSS fájl alkalmazásának előnye, hogy több lap vagy akár egy teljes webhely stílusait egy helyen lehet tárolni, így gyorsan és könnyen frissíthető.
A CSS egyszerű szintaxissal rendelkezik, csak néhány angol nyelvű kulcsszót használ a stílusok tulajdonságaihoz. A stíluslap maga a stílust leíró szabályok sora. Minden szabályhoz tartozik egy szelektor és egy deklarációs szakasz. Ez utóbbi kapcsos zárójelek között pontosvesszővel elválasztott deklarációkat tartalmaz. A deklarációk formája a következő: a tulajdonság neve, egy kettőspont, majd az adott tulajdonság értéke.

```
div.vissza {
    font-size: x-large;
    color: #000000;
}
```

kódrészlet az osztalyok.css fájlból

A fenti kódrészlet a betűméretet állítja be nagyra és a szöveg színét feketére. Az a HTML oldal, amely alkalmazza az osztályok.css-t és tartalmaz olyan DIV-et melynek class attribútuma *vissza,* megkapja az adott formázást.

6.7 Az AJAX alkalmazása

A legnyilvánvalóbb ok az AJAX használatára a felhasználói élmény fokozása. Az AJAX-ot használó oldalak viselkedése sokkal inkább hasonlít a desktop-os alkalmazásokhoz, mint a tipikus weboldalakhoz. Amikor egy linkre kattintás hatására a teljes weboldal újratöltődik, az sokszor időigényes művelet. Az AJAX-ot használó oldalak ehelyett képesek rá, hogy csak az oldal szükséges részét frissítsék, így gyorsabb reagálást biztosítanak a felhasználói interakciókra.

Az AJAX képes interaktívan kommunikálni egy XML fájllal. Ezt alkalmaztam arra, hogy a generált HTML oldal a rajta megjelenő információkat egy XML fájlból fejtse ki AJAX segítségével.

```
1  <html xmlns:UML="omg.org/UML/1.4">
2     <head>
3        <title>Favorites</title>
4        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5        <script src="select.js"></script>
6        <link href="osztalyok.css" rel="stylesheet" type="text/css">
7     </head>
8     <body class="main">
9        <h1>r'n'b</h1><br><div class="oldalakra"></div>
10       <form>
11           Select the R'N'B singer:
12           <select name="cds" onchange="showCD(this.value)">
13           <option value="Will Smith">Will Smith</option>
14           <option value="Beyonce Knowles">Beyonce Knowles</option>
15           <option value="Mariah Carey">Mariah Carey</option>
16           <option value="Rihanna">Rihanna</option>
17           <option value="Aaliyah">Aaliyah</option>
18           </select>
19       </form>
20
21       <div id="txtHint" class="txthint"><b>CD info</b></div></br>
22
23       <img class="double-border" src="music.jpg" alt="" />
24
25       <div class="vissza"><a href="javascript:history.go(-1)">Vissza</a></div>
26
27    </body>
28 </html>
```

8. ábra - r'n'b.html kódja

Az 8. ábrán látható az egyik HTML oldal kódja, amely egy JavaScript fájlra mutat. Egy egyszerű HTML form egy drop down box-szal aminek *cds* a neve.

A txtHint id-jű a HTML oldal azon része, ahova majd a webszervertől kapott információ kiírásra kerül. Alapértelmezetten a „CD info" szöveget tartalmazza.

Ha kiválasztunk a legördülő listából egy nevet, akkor lefut a *showCD()* függvény. A függvény lefutása az *onchange* -a Drop down box-ban a kiválasztott érték megváltozik- eseményhez van kötve. Tehát minden egyes alkalommal, amikor megváltoztatjuk a legördülő menüben a kiválasztott nevet, akkor a *showCD()* meghívódik.

9. ábra - r'n'b.html a böngészőben

6.7.1 Az Ajax JavaScript

A *showCD* függvény a megadott *select.js* fájlban található. Paramétere a kiválasztott név, amelyet továbbküld egy XMLHttpRequest objektum segítségével a szervernek. Ezen objektum valósítja meg a kommunikációt a szerver és a kliens között.

A *select.js* tartalmazza még a *stateChanged()* függvényt, amely ha a szerver megtalálta a feltett kérdésre a választ, azaz: kiolvasta a szükséges információt az XML fájlból, akkor a *txtHint* id-jű -be kiírja azt, a HTML oldal újratöltése nélkül.

6.7.2 Az Ajax szerver oldal

A kliens és a szerver közötti kommunikációt a 6.2.3 fejezetben bemutatott IIS webszerver valósítja meg. Az Inetpub/www könyvtárban el kell helyezni az output1 mappát, amely tartalmazza a bővített HTML vázat, a select.js JavaScript fájlt, get.asp ASP fájlt és a catalog.xml fájlt.

A szerver oldali funkciókat a JavaScript segítségével érjük el. A szerver oldalunk egy ASP fájl, aminek get.asp a neve. Ez tartalmaz egy lekérdezést, amellyel az XMLHttpRequest objektummal megkapott értéket megkeresi az XML fájlban (catalog.xml) és a megfelelő információt szintén az XMLHttpRequest objektum segítségével visszaküldi a kliensnek (a böngészőnek), ami kiírja azt a HTML oldalra.

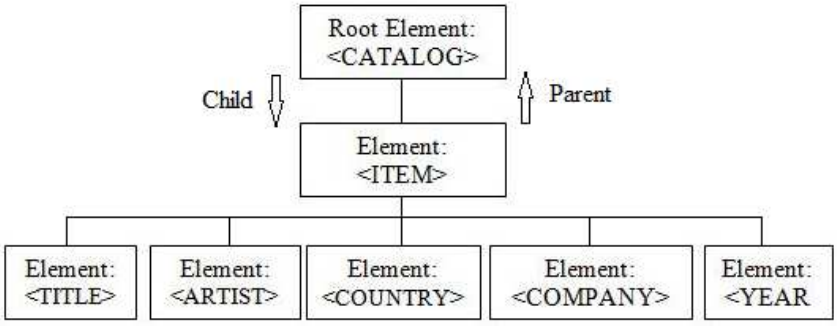
10.ábra – r'n'b.html

6.7.3 Az információtárolásra használt XML fájl

A catalog.xml fájlban található meg, a HTML oldalon a felhasználó által gerjesztett kérdésre a válasz. A fájl felépítése a következő:

```
<CATALOG>
  <ITEM>
    <TITLE>Dangerously in Love</TITLE>
    <ARTIST>Beyonce Knowles</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia Records</COMPANY>
    <YEAR>2003</YEAR>
  </ITEM>
…
</CATALOG>
```

kódrészlet a catalog.xml fájlból



11. ábra - XML fa

Hogyan kerül kiolvasásra az információ? A legördülő listából való választáskor, a kiválasztott név adódik tovább értékként a select.js JavaScript fájl *showCD()* függvényének, amely XMLHttpRequest objektummal átkerül a szerverhez. Az ASP fájl ismeri a *catalog.xml* felépítését és a következő lekérdezést tartalmazza:

```
set nodes=xmlDoc.selectNodes("CATALOG/ITEM[ARTIST='" & q & "']")
```

Ahol a q, a paraméterként megkapott név. Megkeresi melyik az az ITEM, amelyben az ARTIST megegyezik a paraméterrel.

7 Összegzés

Szakdolgozatomban sikerült azt megvalósítani, amit a célkitűzésben megjelöltem: sikerült egy olyan automatizálható eljárást létrehoznom, ami akár más fejlesztőknek is a segítségére lehet. A kiindulási pontként választott osztálydiagram biztosítja a felület átgondoltságát, leírja az oldalak közötti kapcsolatokat, az azok között történő lehetséges navigációt. A dolgozatban bemutatott XSLT fájl biztosítja, hogy az osztálydiagram módosítása után a felület váza egyszerűen újragenerálható marad. A kész vázat a programozó könnyedén képes a kívánt funkciókkal kitölteni. A felületen a felhasználó által választható, hogy miről olvasna bővebb információkat, ezt sikerült dinamikusan, a lapok újratöltése nélkül megoldani.

Az használt technológiákat sikerült alapjaiban megismerni, és az általuk nyújtott számos lehetőség közül néhányat alkalmazni. Persze a fejlesztést sohasem lehet abbahagyni, mindig vannak újabb és újabb ötletek, amiket be lehet építeni egy alkalmazásba. A továbbfejlesztés egyik lépése lehetne, hogy az oldalak alapjául szolgáló adatok nem XML fájlban, hanem adatbázisban kerülnek eltárolásra, ezzel is elősegítve a biztonságosabb információtárolást.

8 Irodalomjegyzék

Bővebb információ:

[1] http://www.omg.org/
[2] http://www.uml.org/
[3] http://www.omg.org/mda/
[4] http://www.w3.org/XML/
[5] http://www.w3.org/TR/xslt
[6] http://www.asp.net/ajax/documentation/
[7] http://www.oxygenxml.com/
[8] http://www.iis.net/getstarted
[9] http://www.w3c.hu/

Bevezetés

http://hasznos.blog.dada.net/post/596064/Az+Internet+története++dióhéjban....

Felhasználói felületek

http://hu.wikipedia.org/wiki/Felhasználói_felület

http://en.wikipedia.org/wiki/User_interface

2004, © Gyula Simon 2005 (magyar verzió)

NetAcademia/Tudástár : Molnár Ágnes, Objektumorientált alkalmazásfejlesztés – Az UML

tudastar.netacademia.net/publikacio/cikk/doc/0309oop2.doc

Rendszerfejlesztés

Ian Sommerville: Software Engineering, 7th edition. Chapter 16 © Ian Sommerville

Ian Sommerville: Szoftverrendszerek fejlesztése Budapest Panem 2007

Object Management Group , Modellvezérelt Architektúra, UML

http://en.wikipedia.org/wiki/Model-driven_architecture

http://agrinf.agr.unideb.hu/if2005/kiadvany/papers/E44.pdf

http://docs.kde.org/stable/en_GB/kdesdk/umbrello/uml-basics.html

tudastar.netacademia.net/publikacio/cikk/doc/0210modell1.doc

XMI, XML, AJAX, XSLT

http://www.omg.org/technology/xml/

http://en.wikipedia.org/wiki/XML
http://hu.wikipedia.org/wiki/XML
http://www.w3schools.com/xml/default.asp
http://hu.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations
http://www.w3.org/Style/XSL/
http://www.w3schools.com/ajax/

9 Függelék

OSZTÁLYOK.ETD

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <!DOCTYPE XMI SYSTEM "C:\Documents and
Settings\norci\.netbeans\6.5rc2\.uml\config\UML_2.0EMBT.dtd"> -->
<XMI xmlns:UML="omg.org/UML/1.4" xmi.version="2.0">
   <XMI.header>
      <XMI.metamodel xmi.name="UML" xmi.version="1.4"/>
      <XMI.documentation>
         <XMI.exporter> Embarcadero's Describe </XMI.exporter>
         <XMI.exporterVersion> 6.0 </XMI.exporterVersion>
      </XMI.documentation>
   </XMI.header>
   <XMI.content>
      <UML:Project xmi.id="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
mode="Design"
         defaultLanguage="UML" projVersion="6.1.4.837" name="Favorites">
         <UML:Element.ownedElement>
            <UML:Class xmi.id="DCE.3D35C90F-9A98-B497-6515-FBC83705D94F"
name="welcome"
               owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B">
               <UML:Element.presentation> </UML:Element.presentation>
               <UML:Classifier.generalization>
                  <UML:Generalization xmi.id="DCE.E73594E1-C31B-D6AF-C2A3-
2118254CD9B1"
                     specific="DCE.3D35C90F-9A98-B497-6515-FBC83705D94F"
                     general="DCE.B7144DE1-CF75-31DC-1F4A-63360106C049">
                     <UML:Element.presentation> </UML:Element.presentation>
                  </UML:Generalization>
                  <UML:Generalization xmi.id="DCE.7B3DC9F8-6E61-A537-3DA2-
409A009E9194"
                     specific="DCE.3D35C90F-9A98-B497-6515-FBC83705D94F"
                     general="DCE.B6B29BAB-3E94-C9D5-C1EF-619887D8CE6F">
                     <UML:Element.presentation> </UML:Element.presentation>
                  </UML:Generalization>
                  <UML:Generalization xmi.id="DCE.771F40B7-F878-EBA6-A263-
6E56972EA051"
                     specific="DCE.3D35C90F-9A98-B497-6515-FBC83705D94F"
                     general="DCE.42CDD2C2-1B94-D23F-25CA-84CE97AF238F">
                     <UML:Element.presentation> </UML:Element.presentation>
                  </UML:Generalization>
               </UML:Classifier.generalization>
            </UML:Class>
            <UML:Class xmi.id="DCE.B7144DE1-CF75-31DC-1F4A-63360106C049"
name="movie"
               owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
               specialization="DCE.E73594E1-C31B-D6AF-C2A3-2118254CD9B1">
```

```
            <UML:Element.presentation> </UML:Element.presentation>
          </UML:Class>
          <UML:Class xmi.id="DCE.B6B29BAB-3E94-C9D5-C1EF-619887D8CE6F"
name="book"
            owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
            specialization="DCE.7B3DC9F8-6E61-A537-3DA2-409A009E9194">
            <UML:Element.presentation> </UML:Element.presentation>
            <UML:Classifier.generalization>
              <UML:Generalization xmi.id="DCE.1498D0F7-5F4D-F69E-EAE0-
C79730CE6ACE"
                specific="DCE.B6B29BAB-3E94-C9D5-C1EF-619887D8CE6F"
                general="DCE.C266A1D3-8619-378E-9775-819F86EB25C1">
                <UML:Element.presentation> </UML:Element.presentation>
              </UML:Generalization>
              <UML:Generalization xmi.id="DCE.0AC89F61-74F9-B325-FD20-
CA10077BCBE0"
                specific="DCE.B6B29BAB-3E94-C9D5-C1EF-619887D8CE6F"
                general="DCE.32F692B0-2A64-6FE0-8325-5B07BCBC28C8">
                <UML:Element.presentation> </UML:Element.presentation>
              </UML:Generalization>
            </UML:Classifier.generalization>
          </UML:Class>
          <UML:Class xmi.id="DCE.42CDD2C2-1B94-D23F-25CA-84CE97AF238F"
name="music"
            owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
            specialization="DCE.771F40B7-F878-EBA6-A263-6E56972EA051">
            <UML:Element.presentation> </UML:Element.presentation>
            <UML:Classifier.generalization>
              <UML:Generalization xmi.id="DCE.E3125A3A-2D5A-C200-9089-
13D7E05EE9A4"
                specific="DCE.42CDD2C2-1B94-D23F-25CA-84CE97AF238F"
                general="DCE.C5FF70F3-FD1B-DFA1-7F09-9BC5B9CDDC84">
                <UML:Element.presentation> </UML:Element.presentation>
              </UML:Generalization>
              <UML:Generalization xmi.id="DCE.72622432-38C6-81FE-FE5C-
1BF260856A89"
                specific="DCE.42CDD2C2-1B94-D23F-25CA-84CE97AF238F"
                general="DCE.FC6B76E3-80D6-8609-C071-2A42CD8D451F">
                <UML:Element.presentation> </UML:Element.presentation>
              </UML:Generalization>
              <UML:Generalization xmi.id="DCE.5822B41B-E59C-F7D4-29A9-
E8B56DE904AF"
                specific="DCE.42CDD2C2-1B94-D23F-25CA-84CE97AF238F"
                general="DCE.127C7AF8-905E-F6E6-2EB7-865267AB195F">
                <UML:Element.presentation> </UML:Element.presentation>
              </UML:Generalization>
            </UML:Classifier.generalization>
          </UML:Class>
```

```
            <UML:Class xmi.id="DCE.C266A1D3-8619-378E-9775-819F86EB25C1"
name="fantasy"
               owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
               specialization="DCE.1498D0F7-5F4D-F69E-EAE0-C79730CE6ACE">
               <UML:Element.presentation> </UML:Element.presentation>
            </UML:Class>
            <UML:Class xmi.id="DCE.32F692B0-2A64-6FE0-8325-5B07BCBC28C8"
name="history"
               owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
               specialization="DCE.0AC89F61-74F9-B325-FD20-CA10077BCBE0">
               <UML:Element.presentation> </UML:Element.presentation>
            </UML:Class>
            <UML:Class xmi.id="DCE.C5FF70F3-FD1B-DFA1-7F09-9BC5B9CDDC84"
               name="r&amp;apos;n&amp;apos;b" owner="DCE.B7E866EC-B0A8-F679-1910-
B691EB63A16B"
               specialization="DCE.E3125A3A-2D5A-C200-9089-13D7E05EE9A4">
               <UML:Element.presentation> </UML:Element.presentation>
            </UML:Class>
            <UML:Class xmi.id="DCE.FC6B76E3-80D6-8609-C071-2A42CD8D451F"
name="rap"
               owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
               specialization="DCE.72622432-38C6-81FE-FE5C-1BF260856A89">
               <UML:Element.presentation> </UML:Element.presentation>
            </UML:Class>
            <UML:Class xmi.id="DCE.127C7AF8-905E-F6E6-2EB7-865267AB195F"
name="hip-hop"
               owner="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B"
               specialization="DCE.5822B41B-E59C-F7D4-29A9-E8B56DE904AF">
               <UML:Element.presentation> </UML:Element.presentation>
            </UML:Class>
         </UML:Element.ownedElement>
         <UML:Package.packageImport>
            <UML:PackageImport xmi.id="DCE.5ADE8BFC-59D2-4551-9966-
80D2A81E6FB2"
               importingPackage="DCE.B7E866EC-B0A8-F679-1910-B691EB63A16B">
               <UML:PackageImport.importedPackage>

                  <UML:Profile xmi.id="DCE.23863F4D-69AB-92FA-1E45-
EC66C9C54BAD"
                     name="Unnamed" isCloned="true"
                     href=".\Favorites.etup#//*[@xmi.id=&amp;apos;DCE.23863F4D-69AB-92FA-
1E45-EC66C9C54BAD&amp;apos;]"
                     isDirty="false"/>
               </UML:PackageImport.importedPackage>
               <UML:Element.ownedElement>
                  <UML:TaggedValue xmi.id="DCE.9A18ACC5-1419-8390-2E17-
2F640BEC3584"
                     owner="DCE.5ADE8BFC-59D2-4551-9966-80D2A81E6FB2"
```

```
name="autoCreated">
                    <UML:TaggedValue.dataValue>true</UML:TaggedValue.dataValue>
                  </UML:TaggedValue>
                </UML:Element.ownedElement>
              </UML:PackageImport>
            </UML:Package.packageImport>
          </UML:Project>
        </XMI.content>
</XMI>
```

OSZTÁLYOK.XSL

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs"
version="2.0"
   xmlns:UML="omg.org/UML/1.4">

   <xsl:output method="text"/>
   <xsl:output method="html" indent="yes" name="html"/>

   <xsl:template match="XMI/XMI.content/UML:Project/UML:Element.ownedElement">
      <xsl:for-each select="//UML:Class">
         <xsl:variable name="filename" select="concat('output1/', @name,'.html ')"/>
         <xsl:value-of select="$filename"/>

         <xsl:result-document href="{$filename}" format="html">
            <html>
               <head>
                  <title>
                     <xsl:value-of select="//UML:Project/@name"/>
                  </title>
                  <link href="osztalyok.css" rel="stylesheet" type="text/css"/>
               </head>
               <body class="main">
                  <h1>
                     <xsl:value-of select="@name"/>
                  </h1>
                  <br/>
                  <xsl:variable name="id">
                     <xsl:value-of select="@xmi.id"/>
                  </xsl:variable>
                  <xsl:variable name="oszt_nev">
                     <xsl:value-of select="@name"/>
                  </xsl:variable>
                  <div class="oldalakra">
                     <xsl:if test="child::UML:Classifier.generalization">
                        <xsl:for-each
                           select="UML:Classifier.generalization/UML:Generalization">
```

```
                        <xsl:variable name="gen">
                           <xsl:value-of select="@general"/>
                        </xsl:variable>

                        <xsl:for-each select="//UML:Class">
                           <xsl:if test="@xmi.id=$gen">
                              <a href="{@name}.html">
                                 <xsl:value-of select="@name"/>
                              </a>
                           </xsl:if>
                        </xsl:for-each>
                        <br/>
                     </xsl:for-each>
                     <br/>
                  </xsl:if>
               </div>
               <div class="vissza">
                  <a href="javascript:history.go(-1)">Vissza</a>
               </div>
            </body>
         </html>
      </xsl:result-document>
   </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```