NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Lovas István**
2015 Hallgató törzskönyvi száma: **T/002145/FI12904/N**

TARTALOMJEGYZÉK

ÁBRAJEGYZÉK

RÖVIDÍTÉS JEGYZÉK

Rövidítés	Mit rövidít	Megjegyzés
AJAX	Asynchronous JavaScript and XML	Aszinkron szerverhívás JavaScript nyelven
API	Application Programming Interface	Alkalmazásprogramozási felület
CASE	Computer-aided Software Engineering	Számítógéppel támogatott szoftvertervezés
CRUD	Create, Read, Update, Delete	A létrehozás, visszaolvasás, módosítás, törlés funkciók együttese.
CSS	Cascading Style Sheets	egymásba ágyazott stíluslapok
DOM	Document Object Model	XML alapú nyelvek objektum leíró nyelve
HOOD	Hierarchic Object-Oriented Design	Metodika
HTML	HyperText Markup Language	hiperszöveges jelölőnyelv
Hub		A jelen dolgozatban SignalR alapú üzenetküldő csomópontot jelöl.
IE	Information Extraction	Információkinyerés
MVC	Model View Controller	Kódszervezési minta
MVVM	Model View ViewModel	Kódszervezési minta
OCL	Object Constraint Language	Egy nyelv az UML modell megszorításainak leírására
OMT	Object-modeling technique	Metodika
ORM	Object-relational mapping	oniektum-relációs leképző
SPA	Single Page Application	Egy oldalas alkalmazás, a különböző aloldalakat jellemzően JavaScript állítja elő
SVG	Scalable Vector Graphics	Skálázható vektorgrafika
SysML	Systems Modeling Language	Rendszermodellező diagramnyelv
tf-idf	term frequency–inverse document frequency	A szó-dokumentum mátrix egy súlyozási módszere
UML	Unified Modeling Language	Egységesített modellező nyelv
W3C	World Wide Web Consortium	
WIP	Work In Progress	Egyidejűleg futó munkafolyamatok száma
XML	Extensible Markup Language	Kiterjeszthető Jelölő Nyelv

1. ábra: Rövidítés jegyzék – táblázat

1 CÉLKITŰZÉSEK

Dolgozatomban kitűzött célom egy olyan eszköz kifejlesztése, ami képes segíteni a szoftverfejlesztés egyes magas absztrakciós tevékenységeit, az eszköz felhasználói által elkészítendő szoftver-rendszer követelményeinek specifikációjával, analízisével, illetve modellezésével kapcsolatos teendőit. A készítendő rendszernek célja továbbá, hogy támogassa a csoportos munkát, lehetőleg úgy, hogy a különböző felhasználók közel valós időben láthassák egymás tevékenységeit a rendszert felhasználva. Végül a rendszer legyen a lehető legkülönbözőbb környezetekből, platformokról elérhető és használható.

A kifejlesztendő rendszer legfontosabb felhasználói célcsoportja az üzleti elemzők, akik a munkájuk során különböző szoftverprojektek követelményeit tárják fel. A követelmény feltárás közben a feltárandó rendszer teljesebb megértése és leírása érdekében modellezik a rendszer felhasználóit, és funkcióit. A kifejlesztendő rendszer ezen üzleti célok elérésében kell, hogy segítsen.

Célom, hogy a fejlesztési folyamat összes alapvető tevékenységét végrehajtsam és a lépések fontosabb eredményeit a dolgozatomban be is mutassam. Így a rendszert specifikálom majd. Megtervezem és implementálom az eszközt. Az elkészült funkciókat tesztelem és a rendszer továbbfejlesztési lehetőségeit is be fogom mutatni.

A fejlesztési folyamat megkezdése előtt, a szakirodalom alapján meg kívánom vizsgálni és a dolgozatomban be is kívánom mutatni a követelménytervezés terminológiáját és a követelményfolyamat legfontosabb teendőit. Összevetem továbbá a követelmény specifikáció hagyományos, illetve az újabban népszerű agilis megközelítéseit.

A szakirodalom alapján meg kívánom vizsgálni, hogy milyen módokon lehet a követelmények szöveges reprezentációjából információt kinyerni és ezeket az információkat milyen módokon lehet a rendszerreprezentáció mélyebb absztrakcióiban felhasználni. Ennek érdekében először be fogom mutatni a szövegbányászat egyes ágait, illetve megvizsgálok olyan módszereket, melyek a szoftverspecifikációk sajátos jellemzői, mint például azok szövegének struktúrája alapján működik.

2 IRODALOMKUTATÁS

2.1 Az irodalomkutatás bevezetése

Az elkövetkezendő szakaszokban a feldolgozott irodalmak alapján bemutatom a szoftverkövetelményeket, a követelménytervezés folyamatát, kitérek a követelmények modellezésére, a szakirány központú szemléletekre, és az agilis követelménykezelésre. Végül a szövegbányászat alapvető koncepcióit mutatom be, és kitérek néhány alternatív, egyszerű, a szoftverkövetelményekkel kapcsolatos információkinyerési megközelítésre.

2.2 A követelménytervezés alapfogalmai

Ebben a szakaszban igyekszem vázolni a követelményekkel kapcsolatos összes fogalmat. Egyes fontosabb fogalmakra a későbbiekben külön fejezetben részletesebben kitérek. Nem foglalkozok ellenben a kritikus rendszerek speciális igényeivel, illetve a formális specifikációk mélyebb elemzésével, mert ezek a dolgozat hatókörén kívül esnek. A szakasz Ion Somerwille könyve [3] alapján készült.

2.2.1 Szoftverkövetelmény

A rendszer követelményei lehetnek a rendszer funkcionalitására vonatkozó elvárások magas és-vagy alacsonyabb szintű, részletesebb megfogalmazása, illetve a rendszerrel szemben támasztott megszorítások. Megmondja, hogy a rendszer a megrendelő problémáit milyen szolgáltatások révén oldja meg, és megoldás közben milyen általános és probléma centrikus megkötésekre kell odafigyelni. Részletesség szerint két csoportját érdemes megkülönböztetni, a felhasználói-, illetve a rendszerkövetelményeket. Ezeket a csoportokat szoktuk vegyíteni is, tehát léteznek a funkcionális felhasználói követelmények, a nem-funkcionális felhasználói követelmények, és a rendszerkövetelmények is lehetnek funkcionálisak, illetve nem-funkcionálisak is. Ezeket az eltérő követelményeket fontos lehet elkülöníteni egymástól, akár jelöléssel, akár úgy, hogy a követelmény dokumentáció külön alfejezeteit alkotják. Megfontolandó viszont az összekapcsolódó, de különböző csoportba sorolt követelmények gyors összevetésére is módot adni, de legalább egy hivatkozást elhelyezni a kapcsolódó követelményre. [3]

2.2.2 Követelmények csoportosításai

Követelmények csoportosítása történhet a megfogalmazásuk mélysége szerint. A gyakorlatban fontos szétválasztani a követelmények azon szintjét, mely a megrendelőkkel, illetve a felhasználókkal történő egyeztetést segítik, ezek a

felhasználói követelmények, és azt a szintet, melyben a rendszer követelményeit részletezzük, ezeket nevezzük rendszerkövetelményeknek. A követelmények egy másik csoportosítása történhet az alapján, hogy az adott követelmény egy a rendszertől elvárt szolgáltatást részletez, azaz funkcionális követelmény, vagy egy a rendszertől, esetleg annak egyes szolgáltatásaitól elvárt tulajdonságot ír le, tehát nem-funkcionális követelmény. [3]

Felhasználói követelmények

Absztrakt módon leírja a rendszertől elvárt szolgáltatásokat, külső viselkedését, és azok működési megszorításait. Közérthető, természetes nyelvű leírás, melyet a rendszer felhasználójának, a megrendelőnek, és ezek megbízottjainak szánnak. Esetenként a könnyebb érthetőség kedvéért kiegészítik magas szintű, vázlatos használati eset diagramokkal. [3]

Rendszerkövetelmények

A rendszer szolgáltatásait, funkcióit, működési feltételeit és megszorításait részletezi. A felhasználói követelményeket fejtik ki bővebben. Érdemes megjegyezni, hogy a rendszerkövetelmények dokumentumát szokták specifikációnak is hívni. Lényeges a specifikáció pontos és precíz megfogalmazása, mert a rendszerspecifikáció gyakran része a szerződésnek. A rendszerspecifikációból lehetetlen kizárni minden a tervezéshez köthető információt.

A rendszerkövetelmények megalapozzák a tervezést, illetve az implementációt, így a szoftver megalkotásában résztvevők a legfőbb olvasói. Ebből következik, hogy, míg a felhasználói követelményeknél elengedhetetlen a természetes nyelv és a külön képzettség nélküli érthetőség, addig a rendszerkövetelmények szintjén, az egyszerű formális szövegek mellett speciális jelöléseket és modelleket is alkalmazhatunk. Ilyenek lehetnek a stilizált, formázott és strukturált természetes nyelv vagy a követelmények grafikus modelljei, mint a részletezett használati esetek vagy akár a matematikai formális nyelvek. [3]

Funkcionális követelmények

Leírja a rendszertől elvárt szolgáltatásokat. Az iménti fejezetekből világosan látható, hogy két szintje lehetséges, a funkcionális felhasználói követelmények magas szintű állítások, amik megfogalmazzák a rendszertől elvárt funkcionális viselkedést, míg a funkcionális rendszerkövetelmények, e funkcionalitás részletezései. [3]

Nemfunkcionális követelmények

Az egész rendszerre vonatkozó tulajdonságok. Olyan eredő rendszertulajdonságokra vonatkozhat, mint a megbízhatóság, a válaszidő vagy a tárfoglalás. Lehet megszorítás

egyes használt technológiákra, eljárásokra, vagy szabványokra vonatkozólag. A teljes rendszerre vagy akár annak egy-egy összetevőjére is vonatkozhatnak. Amíg egy funkcionális követelmény nem megfelelő támogatása a rendszer csak kis részét érinti, addig, ha a rendszer nem teljesíti a nemfunkcionális követelményeket, az akár a teljes rendszert is használhatatlanná teheti. A nemfunkcionális követelmény nem csupán a rendszerre vonatkozhat, de a kifejlesztésének folyamatára is. Megszabhatja tehát a kifejlesztés közben használt metodikát, különböző minőségszabványok betartását írhatja elő, vagy akár megszabhatja a fejlesztés alatt használatos CASE eszközök sorát. Lehetséges, hogy egy adott nemfunkcionális követelmény megléte szükségessé teszi más, funkcionális követelmények felvételét.

A nemfunkcionális követelmények egyik problematikája, hogy a megrendelők gyakran csak általános célokként fogalmazzák meg. Ilyen nehezen mérhető jelzőket használnak, mint gyors, kisméretű, egyszerűen kezelhető, hordozható, megbízható, stb. Ezek helyett célszerű a követelményekben különböző objektív metrikákat használni. Például a gyors jelző kifejezhető inkább a másodpercenkénti tranzakciók számával vagy egy eseményre adott átlagos válaszidővel. [3]

Szakterületi követelmény

A szakterületi, vagy más néven környezeti követelmények, a rendszer alkalmazási területéről származnak. Ezek a követelmények nem éppen felhasználói igények, inkább a szoftver alkalmazásának szakterületéből adódó funkcionalitások, vagy megszorítások.

Ezek teljes értékű funkcionális, vagy nemfunkcionális követelmények, de mind a nyelvezetük, mind fogalomrendszerük az adott szakterülethez igazodik. Előírhatja, hogy az adott feladatot hogyan kell végrehajtani, hogy az illeszkedjen az adott szakterület bevett gyakorlatához. Lehet akár egy szabvány, de lehet akár egy képlet is, amivel az adott szakterületen számolnak.

Ezen követelmények nagy jelentőségűek. Az adott szakterület képviselőinek egyértelműek, azonban a rendszer tervezőinek némiképp idegenek lehetnek a nyelvezetük és a mögöttes implicit háttértudás hiányában. Ezt a problémát projectszótár használatával lehet némiképp orvosolni. [3]

Szükséges és javasolt követelmények

A javaslat nem mindig követelmény, lehet, hogy csupán egy igény a rendszerre vonatkozólag, esetleg egy teoretikus elképzelés, ami jelenleg nem döntő fontosságú a rendszer szempontjából. Érdemes lehet ezeket is rögzíteni, hisz nyerhetünk belőlük újabb jó ötleteket vagy az is lehet, hogy egy átpriorizálás közben felértékelődik, és szükségessé nyilváníthatják.

A javasolt követelmények nyelvezetében érdemes ilyen kifejezéseket használni, mint: „jó lenne ha”, vagy „érdemes lenne”, „javasolt lenne”, „javallott” stb.

Míg a szükséges követelmények a „kell”, a „szükséges”, a „fontos”, a „kívánt” az „elvárt”, és hasonló szavak használata különíti el az előző csoporttól. [1] [5]

2.2.3 Kulcsfigura

A kulcsfigurának nevezzük a rendszerrel kapcsolatba kerülő végfelhasználókat, és az összes olyan egyént, akire a szoftverrendszer bármilyen hatást gyakorol. Ilyen hatások lehetnek például a szoftver beüzemelése közben előforduló fennakadások hatásai, vagy akár a megrendelő cégen belüli személyek közti politikai erőviszonyok megváltozása, az újonnan bevezetett rendszer hatására. [3]

2.2.4 Verifikáció és Validáció

Ezt a két lépést együtt szokták V&V-nek is rövidíteni. Míg a verifikáció azt ellenőrzi, hogy a szoftvert jól, azaz a specifikációknak megfelelően készítettük-e el, addig a validáció azt ellenőrzi, hogy tényleg a megfelelő, a felhasználó által óhajtott, értékes terméket alkottuk meg. [3]

2.2.5 Fogalomszótár

A követelmények között, vagy a különböző modelleken megjelenhetnek olyan szavak, melyeket érdemes lehet definiálni. Ez több célt is szolgálhat, növeli a szövegek precizitását, megóvhat az egyes félreértésektől, kiküszöböli a kétértelműségeket. A szakterületi kifejezések leírása segítheti a fogalmak megértését, és ez által segítheti a hatékonyabb fejlesztést. A rendszerben használt rövidítéseket is érdemes lehet ide felvenni. [5]

2.3 A követelménytervezés folyamata

Ebben a folyamatban megértjük és definiáljuk a rendszer által biztosítandó szolgáltatásokat, illetve a fejlesztési és az üzemeltetési megszorításokat. A folyamat végeredménye a követelménydokumentum, mely rendszerint külön tárolja a felhasználói és a rendszerkövetelményeket.

A folyamatban elemezzük a rendszer fontosságát, majd felderítjük, elemezzük, dokumentáljuk és ellenőrizzük a szoftverkövetelményeket. Ennek megfelelően a folyamat négy nagy tevékenységre bontható. Ezek a részfolyamatok a megvalósíthatósági tanulmány elkészítése, a követelmények feltárása és elemzése, a követelmények validálása, illetve a követelmények kezelése és követése. [3]

A **megvalósíthatósági tanulmány elkészítése** során megvizsgálják és becslést adnak arról, hogy a rendszerrel kapcsolatos elvárások kielégíthetőek-e az adott szoftveres és hardveres technológiák segítségével. Eldöntik, hogy a rendszer költséghatékony-e az adott üzleti szempontokat figyelembe véve. Megvizsgálja továbbá, hogy a költségvetési megszorítások mellett kivitelezhető-e a rendszer. Lehetőség szerint minél gyorsabb, és olcsóbb folyamatnak kell lennie. A tanulmány elkészítése során kiszámításra kerül egy ROI (Return Of Investment) érték, mely megadja, hogy milyen mértékben és mikor térül meg az elkészítendő rendszer, illetve mekkora annak az üzleti haszna. A megvalósíthatósági tanulmány információt biztosít a rendszer elkészítésének költséghatékonyságáról. A tanulmány elkészítésének végeztével döntés születik a fejlesztés folytatásáról. A részfolyamat kimenő dokumentuma a megvalósíthatósági jelentés. [3]

A követelmények feltárása és elemzése:

A folyamat során a potenciális felhasználókkal és megrendelőkkel történő megbeszélések, és egyeztetések során, illetve az esetlegesen már működő rendszerek és folyamatok megfigyelése által az elemzők feltérképezik és megértik a készítendő szoftver követelményeit. Ez a folyamat magában foglalhatja egyes rendszermodellek, illetve prototípusok elkészítését, melyek elősegíthetik a követelmények pontosabb megértését.

A követelmény feltárást megnehezítheti, hogy a rendszerrel kapcsolatos érintett kulcsfigurák pontos személye nem ismert, vagy azok nem ismerik, vagy nem képesek pontosan kifejezni azt, hogy mit várnak el a rendszertől. Továbbá az is előfordulhat, hogy a különböző kulcsfiguráknak eltérő vagy akár egymáséinak ellentmondó igényei vannak. A követelmények elemzése és priorizálása során figyelembe kell venni a rendszert befolyásoló üzleti és gazdasági környezetet.

A követelmény feltárási és elemzési folyamat felderítési-, osztályozási-, szervezési-, priorizálási- és dokumentálási lépései a folyamat során ciklikusan követik egymást. Így a már felderített követelmények folyamatosan fejlődnek, és új követelmények bukkanhatnak fel a folyamat során.

A követelmények felderítése során segítségünkre lehetnek a különböző hasonló rendszerek megfigyelése, a rendszerrel kapcsolatba kerülő másik rendszerek megfigyelése, a kulcsfigurákkal folytatott interjúkból leszűrt tanulságok, prototípusok elkészítése és azoknak a kulcsfigurákkal történő elemzése.

Az interjú módszere segíthet számos követelmény és igény felderítésében és a rendszer későbbi felhasználóinak alaposabb megismerésében, de önmagában nem képes a teljes rendszer összes szükséges követelményét felfedni

A követelmény feltárás másik eszköze a forgatókönyvek vagy más szóval szcenáriók készítése, mely megkönnyítheti a kulcsfigurákkal való precíz kommunikációt, mivel azok ilyenkor a konkrét problémára, magára az üzleti folyamatra tudnak koncentrálni és azzal kapcsolatban véleményt formálni. A forgatókönyvek különböző eszközökkel és különböző formalitási szinteken készíthetők. [3]

Követelmény specifikációs folyamat:

Az elemzési tevékenység során összegyűjtött információk egységes dokumentummá történő szerkesztésének folyamata. Ez a szoftverspecifikáció megalkotásának folyamata, itt készítjük el és tartjuk karban a szoftverkövetelmények dokumentumát. [3]

2.4 Szoftverkövetelmények dokumentuma

A szoftverkövetelmények dokumentuma, amit szoktak szoftverkövetelmény specifikációnak is hívni, az a dokumentum, mely a követelmény specifikáció folyamata során jön létre. A specifikáció a követelmények egy szabványos formába való szedésének és leírásának folyamata. Maga a dokumentum tartalmazza a felhasználói-, illetve a rendszer követelményeket. Szöveges dokumentum, így a megrendelő külön előképzettség nélkül is olvashatja, de a kötött szerkezete elősegíti a lehető legteljesebb információreprezentálást.

A funkcionális és a nemfunkcionális követelményeket célszerű megkülönböztetni a dokumentumon belül. Ez lehet elszeparálás, ilyenkor a külön elhelyezkedő követelmények között nehezebb megtalálni az esetleges összefüggéseket, viszont az egyes követelmények nem mosódnak egybe, és jól elkülöníthetőek a funkcionális, illetve a nemfunkcionális megfontolások. A megkülönböztetés történhet esetleg más vizuális jellemzőkkel is.

A dokumentum tartalmára vonatkozólag az IEEE/ANSI 830-1998-as szabvány nyújthat támpontot. [3]

2.5 Követelmények kezelése

A követelmények kezelése napjaink gyorsan változó világában igen fontos, ugyanis a rendszerrel kapcsolatos elvárások az idő előrehaladtával folyamatosan változhatnak. Megváltozhat az adott funkcionalitást igénylő üzleti folyamat, egyes relatív minőséget befolyásoló elvárások, vagy akár maga a megrendelő vállalat is. Ezzel a változással úgy tarthatjuk a lépést, ha az alkalmazással szembeni követelményeket és ezután magát a szoftverrendszert is a megváltozott érdekekhez szabjuk. A követelmények között vannak olyanok, amik nem, vagy csak igen lassan változnak, mint a megrendelő alapvető tevékenységével, vagy a szakterület alapjaival kapcsolatosak, ezeket hívjuk

tartós követelményeknek, vannak ezen kívül gyakrabban módosuló, átmeneti követelmények is.

A követelmények kezelése gyakran igen sok erőforrást felemésztő és költséges feladat, ezért érdemes a követelménytervezés kezdeti szakaszában elhatározni a kívánt szintjét és megtervezni a menetét. A kezelés kisszámú követelmény esetén még elvégezhető kézi módszerekkel, de egy bizonyos mennyiség után elkerülhetetlen egy CASE eszköz használata.

A követelménykezelést megkönnyítendő, érdemes a követelményekhez egy, azokat a teljes rendszer szintjén azonosítani képes, egyedi azonosítót rendelni. Az azonosítót felhasználhatjuk a rendszer további elemeivel, illetve más dokumentumokkal való összekapcsolásra. [3]

2.6 Követelmény validáció

A követelmények valószerűségét, konzisztenciáját és teljességét ellenőrző tevékenységet hívjuk validálásnak. Azért van szükség a követelmények validálására, hogy meggyőződjünk arról, hogy tényleg azt a rendszert készítjük-e el, amire a megrendelőnek szüksége van. A szoftver hibáinak javítási költsége annál alacsonyabb, minél előbb észrevesszük azt, tehát, ha a követelményekben van hiba, célszerű azt, ha lehet még a követelménytervezés folyamatában, a validálás során észrevenni. A követelményekben vétett hiba kihatással van az összes további fejlesztési lépésre és akár tévútra is vihetik az egész szoftverprojectet. [3]

2.7 A követelmények fontossága

A követelmények meghatározása, elemzése és karbantartása a megvalósítandó komplex szoftverrendszerek életciklusában egy kritikus fontosságú feladat, hisz ez a szakasz segít megérteni, hogy a megrendelő mit is vár el a készítendő rendszertől. A feltárt követelmények alapján döntünk arról, hogy a rendszer megvalósítható-e és, hogy megközelítőleg mennyibe fog kerülni a megrendelőnek. A szoftver komplexitása is becsülhető általa. Általában a szoftver specifikáció része a szerződésnek.

Hagyományosan a specifikáció köti össze a felhasználó igényeit és a fejlesztőket, így a kommunikációs szerepe is jelentős. Amennyiben a rendszer megrendelői mi magunk vagyunk, vagy a kommunikáció igen jó és gyakori a megrendelővel, illetve egyes metodikákat alkalmazva, a követelmények kifejtése a fejlesztés során több részletben, iteratívan történik. [3]

2.8 A természetes nyelv problémái

A természetesen nyelven írt dokumentumok, mint amilyen lehet a felhasználói követelmények is, számos problémával rendelkeznek. Az egyik ilyen probléma az egyértelműség hiánya. Olykor nehéz a nyelvet pontosan használni, egy dolgot többféleképpen is leírhatunk, és a szavainknak is lehet számos jelentése. A szövegeink emellett terjengőssé is válhatnak, ezzel megnehezítve a lényeges információk kiszűrését. A folyó szövegben összemosódhatnak az egyes követelmények, illetve a különböző típusú, funkcionális és nemfunkcionális követelmények keveredhetnek így nehezebb ezeket elválasztani egymástól. [3]

2.8.1 Ajánlások a természetes nyelv problémáinak kiküszöbölésére

A követelmények megfogalmazásakor érdemes lehet minden követelményhez egy egyszerű magyarázatot fűzni, ami kifejti, hogy miért került be a megjegyzés, mi a szerepe annak. Ez nem csak érthetőbbé teszi az adott követelményt, de változásakor is segítséget nyújthat.

Érdemes egy szabványos követelményformátum elkésztése, ami megadja, milyen formai és tartalmi szerkezettel adjuk meg az egyes követelményeket. Egyes ajánlásokban például a követelményeket kártyákra vették fel, minden egyes ilyen kártya egy követelményt tartalmaz, illetve tartalmazza az adott követelmény magyarázatát, más követelményekkel való kapcsolatát, azoktól való függését, illetve a követelmény forrását, azaz azt a személyt, akitől a követelmény ered. Így könnyen megtalálható az, akivel a követelmény változásakor érdemes lehet konzultálni.

A követelmény kulcsfontosságú részeit érdemes lehet kiemelni, félkövér, dőlt, vagy egyéb szövegformázási eszközökkel, ezzel elősegítve a követelmény későbbi olvasását, és a fontosabb információk hangsúlyozását. [3]

A Mellékletek fejezet Követelmény formátum sablonok részénél konkrét példákat is bemutatok a követelmények strukturált formába öntésére.

2.9 Az üzlet és a követelmények modellezése UML segítségével

Az üzleti modellt a megrendelő és a rendszer kifejlesztésével megbízottak üzleti elemzői közösen alkotják meg. Az üzlet egyes részeinek ábrázolására számos nem UML alapú módszer terjedt el, mint például a szervezeti diagram, mely a szervezeteken belüli alá- és fölérendeltséget, a vezetőségi hierarchiát modellezi, vagy a szervezeti folyamatmodellek, melyek üzleti feladatok végrehajtásához szükséges tevékenységek folyamatát fejezi ki. Vannak olyan üzleti modellek, melyeken a piacépítést, bevételtermelést vagy az üzlet növelését lehet megtervezni. A különböző üzleti modellek az üzlet különböző aspektusait hivatottak mutatni.

Az informatikai rendszerek üzleti modellezésekor egy bevett eszköz az UML használati eset diagramja, amit ezen a modellezési szinten szoktak üzleti feladatdiagramnak is nevezni. Itt derítjük fel az üzleti szereplőket, az üzleti feladatokat és ezek kapcsolatait. Az üzleti szereplőkről és a feladatokról érdemes szöveges leírást készítenünk. A feladat leírása tartalmazhatja a feladat definícióját, a fő célját és, hogy miért szükséges a rendszer, illetve az adott szereplők számára. Ezt a feladat küldetését áttekintő leírást célszerű közérthetően megfogalmazni. Mindemellett célszerű a feladathoz megadni egy szöveges forgatókönyvet, vagy más néven szöveges szcenáriót, mely a feladat végrehajtásának lépéseit pontokba szedve, félig formális szövegként ábrázolja. Az üzleti feladatmodell egy másik gyakori diagramtípusa a tevékenységdiagram, mely a szcenáriók megadásának egy a szöveges forgatókönyveknél formalizáltabb módja.

A követelmény-feltárás során ezekből a diagramokból indulunk ki, illetve ezeket részletezzük. A finomítás során jelennek meg új diagram elemek, illetve az általánosítás, a kiterjesztés és a beszúrás kapcsolatok a használati eset diagram elemei között. Míg az üzleti feladatdiagramon az üzleti célok jelennek meg használati esetként, addig itt már jelöljük a rendszer által biztosított összes funkciót. A cél és a funkciók különbségét úgy lehetne szemléltetni, hogy a felhasználó a rendszert nem keresi fel azért, hogy bejelentkezzen, ami lehet egy rendszerfunkció, de lehet az a célja, hogy a termékek között böngésszen. Az üzleti elemzésmodell, mely egy sztereotípiákkal ellátott osztály diagram, már alkalmazható az üzlet és az üzleti folyamatok során előkerülő összes szereplő, eszköz, és lépés a rendszer szempontjából szükséges virtuális lenyomatának modellezésére. A rendszerben szereplő dinamizmusok modellezésére a kommunikációs-, a szekvencia-, illetve az állapot diagram nyújtanak eszközt. A követelmény tervezés során a használati eseteket, illetve rokon funkcionalitásokat már gyakran elkezdjük csomagokba rendszerezni, ezek kapcsolatainak jelölésére kiváló eszköz az UML csomag diagramja. [1]

2.10 A követelmények modellezésére használt diagram eszközök

2.10.1 Használati-eset diagram

A használati eset vagy más néven Use-Case diagram a rendszer felhasználóinak a szemszögéből tekintve ábrázolja a rendszer funkcióit, és céljait. A fejlesztendő szoftverrendszerben megjelenő követelmények áttekinthető ábrázolásának az egyik elterjedt eszköze. Az ábrázolás középpontjában a rendszer külső felhasználói és az általuk végezhető üzleti tevékenységek, szcenáriók állnak.

Szereplőknek vagy aktoroknak hívjuk azokat a felhasználói köröket, melyek használni kívánják a rendszerünket. Az ilyen aktorok általában a valóságban létező felhasználói csoportokat vagy szerepköröket jelölnek, és gyakran megegyeznek a megrendelő szervezet egyes munkahelyi beosztásaival. Szereplőként szoktuk továbbá jelölni a

fontosabb kapcsolódó külső rendszereket, illetve esetenként az olyan külső eseményeket, melyekre a rendszer reagál. A rendszer felhasználói a diagramon gyakran pálcika emberként jelennek meg, de számos eszköz módot ad a megjelenés testre szabására, ezzel javítva a diagram kifejező erejét. A szereplők a modellben <<aktor>> sztereotípiájú elemek.

A szereplők által végezhető feladatokat, üzleti célokat és üzleti tevékenységeket nevezzük használati esetnek, vagy az angol terminológiát átvéve Use-Case-nek. A diagramon megjelenő használati esetek többsége a kifejlesztendő rendszer későbbi elvárt szolgáltatása, azaz a rendszer kifele mutatott kapcsolódási pontja. Emellett megjelenhetnek más, a rendszer vagy az üzlet szempontjából fontos külső folyamatok is, külső használati esetek formájában. Minden használati eset teljes forgatókönyvvel, azaz szcenárióval kell, hogy rendelkezzen, ami megadja, hogy a szolgáltatás milyen lépésekből áll. A használati esetek ovális alakzatként jelennek meg. A rendszer felelősségi körébe tartozó használati eseteket érdemes, kerettel elválasztani a rendszert használó külső szereplőktől, és az esetlegesen megjelenő külső feladatoktól. [5]

2.10.2 Tevékenység diagram

A tevékenység diagram, amit neveznek aktivitás diagramnak is, a rendszer időben lezajló változásainak a szemléltetésére szolgáló egyik eszköz. A használatával igyekszünk a rendszerben megjelenő üzleti munkafolyamatokat, illetve a rendszer tevékenységeinek lépéssorát, grafikusan modellezni. Gyakran használják egy-egy használati eset kifejtésére. [5]

2.10.3 Sztereotípiákkal ellátott elemzési osztálydiagram

Az elemzésdiagram arra hivatott, hogy magas absztrakción mutassa a rendszerben megjelenő osztályokat, és a köztük fennálló kapcsolatokat. Ezen a szinten csak az osztályok nevei szerepelnek, és nincsenek feltüntetve az állapotokat tárolni képes attribútumok se a műveleteket végző konkrét metódusok. Az üzleti elemzésdiagram egy sztereotípiákkal ellátott elemzési osztály diagram, mely segítségével részletesen elemezhetjük a szereplők, és a rendszerben megjelenő további elemek statikus kapcsolatait. Ezeken a diagramokon már a fejlesztők szemszögéből modellezzük az üzletet megvalósító rendszert.

A gyakorlatban, ezen a modellezési szinten háromféle sztereotípiával látjuk el az elemeket. Az ilyen elemek lehetnek határoló-, irányító- vagy entitásosztályok. Ezek a diagramokon általában megjelenésükbe is jól elhatárolódnak.

A határoló osztályok hivatottak reprezentálni a felhasználói- vagy más rendszerekkel való összeköttetést biztosító interfészeket, az ilyen osztályokat a <<boundery>> sztereotípia jelzi. A rendszerben feldolgozási és irányító szerepet betöltő osztályok a

<<controller>> sztereotípiával jelölt kontroller osztályok. A harmadik osztálytípus, mely leginkább adattároló szerepet tölt be az <<entity>> sztereotípiájú entitás. [5]

2.10.4 Szekvencia diagram

A sorrend diagram a rendszer viselkedését írja le, méghozzá úgy, hogy a rendszer elemei között fellépő kölcsönhatások időbeli viszonyait állítja a modellezés középpontjába. Jól szemlélteti, hogy a modellezni kívánt rendszerviselkedésben a résztvevő objektumpéldányok mikor jönnek létre, mikor végeznek műveletet, milyen üzenetváltásokkal kommunikálnak egymással.

A diagram a feladatok sorrendjét, és időbeliségét nagyszerűen képes ábrázolni, de az elágazások, illetve a ciklikusságok jellemzésére, használható aldiagramok, OCL (Object Control Language) és megjegyzések átláthatatlanná és nehézkesen használhatóvá tehetik. Ezek szemléltetésére másik eszközt érdemes választani, például a tevékenység diagramokat.

A szekvencia diagramokon továbbá csak közelítőleg szemléltethető a műveletek vagy az üzenetek időigénye, a tervezés szakaszban egy-egy elem pontosabb időbeli állapotváltozásait szemléltethetjük időzítés diagrammal.

A szekvenciadiagramok akkor használhatóak hatékonyan, ha az adott tevékenységsorrend viszonylag kevés elem közti sűrű kommunikáció révén megy végbe. [5]

2.10.5 Együttműködési diagram

A szekvencia mellett egy másik a rendszerben megjelenő interakciókat mutató diagramtípus az együttműködési vagy más néven kommunikációs diagram. Itt viszont az időbeliség helyett hangsúlyosabb az objektumok szerveződése és a kapcsolataik. Az üzenetváltások hasonló célt szolgálnak, mint a szekvencia diagramoknál, de egymásutániságuk itt halványabban, egyszerű számozásként jelenik meg.

Amennyiben sok elem vesz részt az adott tevékenységben, de ezek között viszonylag kevés üzenetváltás zajlik, akkor e diagramtípus a szekvencia diagramnál praktikusabb szemléltető eszköznek bizonyulhat. [5]

2.10.6 Állapotdiagram

A tevékenység diagram mellett, az állapot-átmenet diagram egy másik eszköz a rendszer időbeli változásainak a szemléltetésére, de az aktivitásokkal szemben itt sokkal inkább a rendszerben külső események hatására bekövetkező állapotváltozások állnak a modellezés középpontjában. Az állapot-átmenet diagramok nem az objektum orientált világból származnak, de jól illeszkednek az OO szemlélethez is. [5]

2.10.7 Csomagdiagram

Az UML modellünkben szereplő, különböző összetartozó elemek és funkcionalitások együtt kezelésére alkalmasak a csomagok. Ezek a csomagok magas szinten használati esetek csoportosítására hivatottak, de amennyiben osztályok csoportosítására használjuk ezeket, akkor a későbbi megvalósítás során tényleges névtérként vagy csomagként jelenhetnek meg, amennyiben erre az adott programozási környezet módot ad.

A csomagok között leggyakrabban használt kapcsolati típus a függőség kapcsolat, mely azt fejezi ki, hogy az egyik csomag működéséhez felhasználja a másik csomagot, tehát függ tőle. Ezen a diagramon értelmezett az úszósávok használata, amit többnyire a csomagok közti rétegződés (angolul layer) szemléltetésére használunk. [5]

2.11 Az agilis követelménykezelés és modellezés

Az agilis követelményekkel kapcsolatos szemlélet szerint a hagyományos IEEE 830 szabványban lefektetett, és ahhoz hasonló megfontolások alapján megírt követelménydokumentumok egyik problémája, hogy azt tárolják, hogy a rendszernek mit kell végrehajtania, és nem azt, hogy a felhasználónak mi a célja, aminek az elérésében a rendszer segíti őt. A terjedelmes több száz oldalnyi formális specifikációk, megnehezítik a teljes kép áttekintését és gyakran túl sok részletet próbálnak meg feltárni. [7]

A használati esetek már inkább a felhasználó céljait figyelembe véve készülnek, de a használati esetek is egy-egy nagyobb rendszerfolyamatot szemléltetnek, és a hozzájuk tartozó, a felhasználó és a rendszer kommunikációs sorát szemléltető fő és mellék forgatókönyvek a továbbiakban is a kelleténél túl sok részletet fednek fel, illetve a feladatok priorizálására sem ad külön módot. Mivel a használati esetek egy folyamatot írnak le, ezért a megváltozásuk gyakran nagy kihatással járhat a rendszerre nézve. Mindemellett, a vázlatos használati eset diagram napjainkban is gyakran használt eszköz akár az agilis fejlesztések során is, mert egy jó vizualizációs eszköze lehet a felhasználói követelményeknek.

Ezeket kiváltandó napjainkban egyre inkább terjednek el és veszik át a helyet az egyszerű szöveges felhasználói sztorik. [6][7][9][10]

2.11.1 Felhasználói sztorik, mint a használati esetek agilis megközelítése

A felhasználói sztori (User Story) a használati esethez hasonlóan a felhasználó szemszögéből közelíti meg a rendszert, de nem egy folyamatot, hanem egy sokkal kisebb részt, a rendszer egy műveletét írja le.

A felhasználói sztori a felhasználó nyelvezetében, mondat formájában írja le az adott felhasználó rendszerrel kapcsolatos célját. Néhány szempontból hasonló a használati

esetekhez, de leírásuk mindig informális és a felhasználó által könnyen értelmezhető. Az ilyen felhasználói sztorik annyira rövidek és tömörek, hogy többnyire egy kártyán vagy egy felragasztható jegyzetlapon is elférnek. A sztorik kisméretű, a felhasználó számára értéket képviselő funkcionalitást írnak le. Mindig tartózkodnak a részletek említésétől, így nem vezetik se a felhasználó se a fejlesztő képzeletét. Fontos továbbá, hogy lehessen róluk beszélgetni, segítsék a párbeszédet a fejlesztő és a felhasználó között. Lényeges szempont, hogy a felhasználó is képes legyen meghatározni ezeknek a relatív fontosságát. Az ilyen szöveges leírások elkészítése és karbantartása külön előképzettség nélkül is könnyen elvégezhető. Míg a használati esetek létrehozását és karbantartását, bonyolultságuk miatt gyakran a fejlesztőknek kellett végezniük, addig a sztorikat már az üzleti oldal is nagyobb magabiztossággal elkészítheti.

A felhasználói sztorik kis méretének köszönhetően könnyebb velük áttekinteni a teljes rendszer összes értékes célját. Ezek a sztorik szoktak megjelenni a különböző backlogokban is, melyek a projektben jelenlévő összes hátralevő feladatot fontosságuk sorrendjében tárolják. A kis méret további előnye, hogy a megváltoztatásuk és karbantartásuk is egyszerűbb. Amikor egy csapat az adott sztorit elkezdi elkészíteni, tehát amikor szükséges, természetesen a sztorit is részletezhetik.

A User Stories segítségével való modellezés során először felderítjük a rendszer fő céljait, ezeket a nagy horderejű átfogó sztorikat epikusoknak (epics) hívjuk, és a későbbiek során ezeket bontjuk fel részletesebb alsztorikra. Fontos, hogy a sztorik függetlenek legyenek egymástól és értéket képviseljenek. A méretük kezelhető legyen, tehát soha ne legyenek akkorák, hogy megnehezítsék a komplexitásuk és a fontosságuk megbecslését vagy a későbbi tervezésüket. [6][7][9][10]

2.12 A szövegbányászatról általánosságában

A szövegbányászattal kapcsolatos fejezetek megírásához Tikk Domonkos által szerkesztett Szövegbányászat című könyv [4] nyújtotta az alapot.

Az emberek, már a kezdeti, ősi civilizációkban is, a szóbeli mellett, jellemzően írásbeli szövegek segítségével tárolták, és adták át egymásnak az ismereteiket. Napjainkban a rögzített tudásanyagainknak jelentős hányada egyszerű szöveges dokumentumokban található. Ezt a feltevést támasztják alá, többek között a Merill Lynch elemzései is, melyek becslése szerint az üzleti információk körülbelül 85%-a található strukturálatlan, illetve gyengén strukturált szövegekben. Az általunk kezelt szövegek növekvő arányban digitálisan tárolt dokumentumok. Így talán nem meglepő, hogy a szövegek, és főképp a bennük lévő információk kezelésének hatékonyabbá tétele napjaink egyre fontosabbá váló informatikai tevékenysége. Az egyik tudományág, mely ezzel foglalkozik, a szövegbányászat.

A szövegbányászatot definiálhatjuk úgy, mint szöveges adatokon végzett feldolgozási és elemzési tevékenység, melynek célja a dokumentumban rejtett információk feltárása, azonosítása, és elemzése. A szövegbányászat interdiszciplináris szakterület, mely olyan informatikai eszközök mellett, mint a gépi tanulás és a hatékony algoritmusok, a matematika és a nyelvészet eszközeit is felhasználja.

A szövegbányászat két nagy alaptípusa a keresés és a rendszerezés. A keresésnél kiválasztjuk azokat a dokumentumokat, ahol egy adott keresőkifejezés előfordul, míg a rendszerezésnél valamilyen kategóriákba vagy előre nem definiált csoportokba soroljuk azokat. A szövegbányászat főbb feladattípusai is ilyen jellegű feladatokat, vagy ezek kombinációját hajtják végre céljaik elérése során.

Ezek a főbb feladattípusok a kereséstámogatás és információ-visszakeresés, az információkinyerés, az osztályozás, a csoportosítás, az összegzéskészítés, a kivonatolás, a válaszkereső rendszerek, a szövegelemzés, és a napjainkban egyre inkább tért nyerő webes tartalomkeresés. Ezek közül csak a dolgozat szempontjából leginkább hangsúlyos információkinyerést fogom mélyebben részletezni.

2.13 Szövegbányászati előfeldolgozás

Számos szövegbányászati feladat megoldható már létező adatbányászati eszközökkel és algoritmusokkal. Ehhez a szöveges adatokat úgy kell transzformálni, olyan alakra kell hozni, hogy ezek a bejáratott eljárások lehetőleg hatékonyan képesek legyenek működni rajtuk. Ezt a transzformációs lépést szokták előfeldolgozásnak is nevezni. A folyamat végeredménye a dokumentumot reprezentáló modell. Az előfeldolgozás egységesítési, formalizációs és normalizációs feladatokat is tartalmaz. [4]

2.14 Szövegbányászati modellalkotás

2.14.1 Egy ismertebb dokumentum reprezentációs model bemutatása

A legelterjedtebb modellek jellemzően valamilyen numerikus objektumok. Ez számos előnnyel jár. Az egyik előny a kisebb tárolási méret. Ugyanis ha a szavakat karakterenként letároljuk, és jellemzően egy-két bájtos egy karakter, akkor nagyobb helyet vennénk igénybe, mint a numerikus tárolásnál, ahol szavanként egy darab 2 vagy 4 bájtos számmal számolunk. Mindemellett, a számok használatának van egy másik jelentős előnye, méghozzá az, hogy matematikai műveleteket-, és transzformációkat hajthatunk végre az ilyen modelleken. Ráadásul a modellben a matematikai eszközökkel való munka elősegíti a dokumentumok hatékony kezelését.

Azt, hogy ténylegesen milyen modellt és adatábrázolást használunk, befolyásolja a megoldandó feladat típusa. Keresés jellegű feladatoknál egy megfelelő szóelőfordulás

táblázat is nagy szolgálatot tehet, míg a rendezés jellegű feladatoknál összetettebb dokumentum összehasonlító módszerekre van szükség.

A modellalkotásnál használt három nagy matematikai elméleti megközelítés a halmazelméleti, az algebrai, illetve a valószínűségelmélet alapú. A halmazelmélet alapú modellek jó szolgálatot tehetnek az egyes keresőrendszerekben, hisz kereséskor minden dokumentumra fennáll, hogy része az eredményhalmaznak vagy sem. Az algebrai modellben a dokumentumokat olyan algebrai objektumokként reprezentáljuk, mint a vektor vagy a mátrix. Ezeket algebrai műveletekkel össze is hasonlíthatjuk, ezért ezek már használhatók rendszerezési feladatok megoldásánál is. A legelterjedtebb megvalósítása a vektortér modell és annak változatai. A valószínűségi modellben maguk a dokumentumok valószínűségi események által reprezentáltak, míg a kapcsolataik feltételes valószínűségi becslések eredményei. [4]

A vektortér modell

A vektortér modellben hatékonyan meg lehet határozni a dokumentumok távolságát, illetve hasonlóságát. A szövegbányászatnál gyakran élünk különböző intuitív heurisztikákkal, melyek meghatározzák az ezeket felhasználó eszközök felhasználhatósági körét és korlátait. A vektortér modellnél azt jelentjük ki intuitív módon, hogy azokat a dokumentumokat tekintjük hasonlónak, melyek szókészlete átfedi egymást, és ennek a hasonlóságnak a mértéke arányos az átfedés mértékével. A modell egy sokdimenziós vektortérben, vektorokkal reprezentálja a dokumentumokat. A vektortér egyes dimenzióit a dokumentumgyűjtemény, más néven korpusz, egyedi szavai adják. Tehát egy-egy dokumentum a szavaiból álló vektor, abban a vektortérben, ahol az egyes szavak a teret kifeszítő vektorok. A dokumentumgyűjteményt egy szó-dokumentum mátrixszal reprezentáljuk. Az egyedi szavak összessége a szótár vagy más néven lexikon. [4]

2.14.2 A szó-dokumentum mátrix jellemző súlyozási sémái

A legegyszerűbb módszer, ami csak a szó dokumentumbeli esetleges meglétét jelöli, a bináris reprezentáció, ahol is a mátrix egy adott dokumentumot reprezentáló oszlopvektorában egy adott szóhoz tartozó sorban nullát írunk akkor, ha az adott szó nem szerepel a dokumentumban és egyet, ha igen, és nem számít az, hogy hányszor; ez az információ elvész.

A leggyakrabban használt tf-idf (term frequency and inverse document frequency) súlyozást úgy kapjuk, hogy a vektortérmodell szavakat reprezentáló tengelyeit az adott szavak idf által megadott relevanciájával arányosan súlyozzuk.

Tehát a szó-dokumentum mátrix adott d_{ik} súlya kiszámítható a $d_{ik} = fki * idf(tk)$ képlettel.

A tf-idf súlyozás értéke magas lesz a nagy megkülönböztető képességű, adott dokumentumra gyakori, de a korpuszra ritka szavaknál. Alacsonyabb lesz a korpuszban gyakoribb vagy az adott dokumentumban ritkább szavaknál, és elhanyagolhatóan alacsony, akár zérus az olyan szavaknál melyek az egész korpuszban gyakran fordulnak elő. [4]

2.15 Információkinyerés

Az információkinyerésnél (Information Extraction - IE) a fő célunk nagy mennyiségű szövegből kigyűjteni a legfontosabb információkat. Tesszük ezt olyan formában, hogy azt később akár egy relációs adatbázisba is beírhatjuk. Tehát a strukturálatlan adatokat kívánjuk valamilyen struktúrában összefoglalni. Az információkinyerés napjaink meghatározó szövegbányászati kutatási iránya, hisz kiválóan alkalmas lehet nagy mennyiségű emberi munka kiváltására. Az adatok strukturált formába öntésével segíti a folyamatosan növekvő mennyiségű információinkat könnyebben kezelhető és jobban áttekinthető, jobban ellenőrizhető és feldolgozhatóvá tenni. Egyik dinamikusan fejlődő altípusa a nyelvközi információkinyerés (Cross-Language IE), melynél az adatokat több különböző nyelvű szövegekből is összeszedjük és táblázatba öntjük, majd elég csak a táblázatfejléceket lefordítani a kívánt nyelvre.

Az információ-kinyerés nagymértékben feladatfüggő megoldásokat kíván, mert többnyire csak előre rögzített típusú elemeket vagyunk képesek a szövegekből hatékonyan kinyerni. Fontos lehet ismernünk a felhasználási szakirány egyes jellemzőit ahhoz, hogy kideríthessük, hogy az adott feladat szempontjából mik a leginkább fontos attribútumok, amiket a szövegből ki szeretnénk gyűjteni, és azokat milyen módon és formában lehet célszerű a felhasználók számára prezentálni. A megoldásunk továbbá függeni fog az alkalmazási terület jellemző korpuszától is. [4]

2.16 További szövegelemzési megfontolások

A szövegek nyelvtani elemzését és annak a szoftverek tervezésben való használatáról először Russell J. Abbott 1983-ban publikált Program design by informal English descriptions című művében tesz említést. Módszerének lényege, hogy a probléma szöveges megfogalmazását alapul véve annak az egyes főneveiből határozza meg a rendszer objektumosztályait, és azok attribútumait, a szöveg igéiből pedig a rendszertől elvárt szolgáltatások, illetve funkciók származhatnak. Abbott szerint az angol szöveg analízise során felfedezett tulajdonnevek konkrét objektumokra, a köznevek osztályokra, a cselekvést kifejező igék metódusokra, a létigés szerkezetek generalizációra, a „has an” jellegű szerkezetek aggregációra utalhatnak. [5][8]

Az ilyen jellegű szöveg analízist az európai hadászati és légi iparban is használatos HOOD metodológia is alkalmazza. Booch is Abbott munkájából indult ki és fejlesztette azt tovább saját módszerének a megalkotása során. Az OMT módszer kezdeti

verzióiban is, a fejlesztés első lépéseként, szövegelemzéssel határozták meg a rendszer főbb osztályait, majd a későbbiekben az OMT-II egy megelőző lépésként a használati esetek használatát ajánlotta a kiinduló szövegek létrehozásának megkönnyítésére. [3]

A módszer egyik hibája, hogy automatikus használata nagy mennyiségű fals osztályt eredményez. Ezért a Rational Unified Process már a szövegben előforduló szakterületi fogalmak jelöli ki fő osztályoknak. Ez a módszer is számos felesleges osztályt jelöl ki, ellenben nem foglalkozik olyan kérdésekkel, mint, hogy az adott szereplőknek legyen osztály megfelelőjük a rendszerben, és melyeknek ne. [5]

Robert C. Martin Tiszta kód című művében, amikor a helyes és beszédes elnevezéséről értekezik, utal arra, hogy az osztályok nevei mindig főnevek, vagy főnévi szerkezetek, de semmiképpen sem igék. Kerülendőnek tartja az olyan általános osztályneveket, mint az Adat, a Menedzser vagy az Info, ezeknek a kifejező ereje igen csekély, félreértésekre adhat alapot, hogy gyakran környezetfüggő a jelentésük, ezért legfeljebb egy másik főnévvel együtt alkothatnak jól használható osztálynevet. A specializált osztálynevek megalkotására pedig az ősosztály főneve elé írt melléknevek használatát ajánlja. A tagfüggvények neveinek pedig igék vagy igei kifejezések használatát javasolja. A melléknevek jelenléte, konstansokra vagy az egymás utáni melléknevek enumeráció jellegű adatszerkezetre utalhatnak.

Robert C. Martin a névválasztásnál a megoldástartomány neveit részesíti előnyben, de kiemeli, hogy ha nem létezik ilyen, akkor a feladattartomány, azaz a szakirány kifejezései is hasznosak lehetnek, hiszen az ilyen szavak használatakor a szakirány szakértőivel is érdemes lehet konzultálni, ha a fejlesztés során problémába ütközünk. [2]

2.17 Az irodalomkutatás eredménye

2.17.1 A követelményelemzést támogató eszköz főbb tulajdonságai.

Az irodalomkutatásban megemlítettem számos eljárást a követelmények leírására, modellezésére. Az elkésztendő eszköz a követelmények felvitelére ezek közül, a napjainkban divatos felhasználói sztori (User Story) sablont nyújtja majd, melynek eredeti részei: a sztorit végrehajtó felhasználó, felhasználói cél, a cél eléréséből származó felhasználói üzleti haszon. Ezt kiegészítem egy kötelezően kitöltendő név mezővel, egy opcionálisan megadható leírás mezővel, és az analízist megkönnyítendő opcionális fontosság és komplexitás mezőkkel.

A követelménymodellezés támogatására az eszköz az UML használati eset diagramját biztosítja.

2.17.2 A specifikációs- szöveg és modell közti híd megteremtése

Véleményem szerint, a szövegek nyelvi elemzése, kiegészítve különböző szakiránnyal kapcsolatos szótárak használatával és a kisebb megkülönböztető képességű szavak kiszűrésével, egy jó eszköz lehet a rendszer főbb felhasználóinak, funkcióinak és osztályainak, illetve azok kapcsolatainak feltérképezésében. Ellenben ezek olyan szövegbányászati és szakirány specifikus eszközöket igényelnek, melyek túlmutatnak e dolgozat határain. Így az eszköz ezen a téren úgy kívánja segíteni a felhasználót, hogy a felhasználói történet sablonokban felvett új felhasználói szerepköröket, a modellezés résznél felkínálja majd használatra.

2.17.3 Egy modern CASE eszköz főbb tulajdonságai.

Véleményem szerint a jövő CASE eszközeinek a csoportos munka támogatására kell törekedniük, illetve arra, hogy a szoftverfejlesztés minél nagyobb területét lefedjék. Úgy tegyék mindezt, hogy egymással a lehető legnagyobb összhangban működnek. Emellett a továbbiakban is törekedniük kell a munka minél teljesebb, minél hatékonyabb, a lehető legkevesebb emberi beavatkozást igénylő segítésére.

Ezekből az elkészítendő szoftver arra vállalkozik, hogy minél inkább támogassa a kollaborációt és könnyű legyen használni. Ellenben nem célja, hogy a teljes fejlesztést végigkísérje, se az, hogy más eszközökkel együtt működjön. Ezek reális továbbfejlesztési céloknak tekinthetőek.

3 MÁR LÉTEZŐ HASONLÓ JELLEGŰ PROGRAMOK VIZSGÁLATA

A következőkben két elterjedt modellező eszközt vizsgálok meg. A vizsgálat során elsősorban azt figyelem meg, hogy az adott szoftver miképpen támogatja a követelmény folyamat egyes lépéseit, a modellezést, illetve a csoportos munkavégzést.

3.1 Sparx Enterprise Architect

A Sparx cég Enterprose Architect (EA) nevű eszköze kimagasló UML támogatással rendelkezik és számos programozási nyelven készült kódbázissal képes szinkronban tartani a modellt. A vizsgálat során a termék 11.1-es verzióját használtam.

3.1.1 A követelményfolyamat támogatása

A rendszer lehetőséget nyújt a követelmények modellelemként való felvételére, és azok egymáshoz való kapcsolására, illetve azok készültségi fokának, illetve komplexitásának megadására. A követelmények felvételének egy másik eszköze a használati esetek, melyet a rendszer támogat. A használati esetekhez leírást és szcenáriót is megadhatunk. A szcenáriók megadásához használhatunk természetes nyelvet. A rendszerben az adott használati esetekhez felvehetünk továbbá aldiagramokat, többek között aktivációs-, szekvencia-, kommunikációs- és állapot átmenet diagramot. Az eszköz módot ad a felhasználói felület prototípusának megrajzolására és a követelményekhez való kapcsolására.

3.1.2 A modellezés támogatása

A rendszer kiemelkedő tulajdonságokkal rendelkezik a modellezés terén, ugyanis számos rendszermodellező eszközt támogat, mint a SysML és az UML 2.5 szabvány szerinti modellezés.

3.1.3 A csoportos munka támogatása

Az EA a projecteket külön fájlkiterjesztésű (.eap) fájlokban tárolja. Lehetőség van arra, hogy ezeket a fájlokat, külső verziókezelő rendszer segítségével ossza meg a csapat egymással. Ez csak kis teamek esetében jelenthet megoldást, és nehézkes az ütközések feloldása, amennyiben többen egyszerre szerkesztették az adott projectfájlt. A szoftvert továbbá be lehet úgy konfigurálni, hogy a project adatait ne a saját fájljába, hanem egy külső adatbázis kezelő rendszerben tárolja, ezzel lehetőséget adva a csoportos diagramkészítésre.

3.2 Visual Paradigm

Egy másik elterjedten használt modellező eszköz a Visual Paradigm (VP), az EA-nál kevesebb modelltípust és programozási környezetet támogat, de a modellezés támogatásához egy jóval ergonomikusabb eszköz. A modellelemek felvételére a hagyományos eszköztáras fogd és vidd módszer használható. Kész elem jobb felső sarkában lévő nyílra kattintva is létrehozhatunk új elemeket.

3.2.1 A követelményfolyamat támogatása

Az EA-hoz hasonlóan itt is lehetőség van, a követelmények diagramelemként való felvételére. Itt egy alapsablon és néhány kiegészítő kapcsolat is adott, amik a követelményelemzéshez segíthetnek. A követelményeket a rendszer alap szinten ellenőrzi is a követelmény elemek helyességét, például szól, ha nincs az adott elem csomagba téve, vagy a szövege nem szerepel a project szótárában, esetleg, ha az adott követelmény nem kapcsolódik a modell egyetlen más eleméhez sem. Ez az eszköz is támogatja a használati esetek módszerét, de a szcenáriók leírására az EA által támogatottak mellett felhasználói sztorikat is használhatunk, amiket priorizálhatunk, illetve sprintekhez is rendelhetünk. A rendszer lehetőséget ad továbbá az Abbotéhoz hasonló szöveges analízishez, mely során egy egyszerű szöveges fájlban jelölhetünk ki részeket és címkézhetjük fel azokat szereplőnek, használati esetnek, osztálynak, vagy egyszerű projectszótárbeli elemnek.

3.2.2 A modellezés támogatása

A diagramok megrajzolására számos ötletes eszközt ajánl, a hagyományos eszköztárról húzd és vidd módszer mellett, az eszköztár elemére duplán kattintva tudunk ugyanabból az elemtípusból egymás után többet is felvenni. Amennyiben egy kész diagramelemre kattintunk, az elem körül számos elem jelenik meg, amik az elemből kihúzva az eddigi diagramelemhez kapcsolt, a kiválasztott elemnek megfelelő elemet hoznak létre. Ez némileg kényelmesebb megoldás lehet az EA jobb felső nyilacskájánál. A modellelemeket összekötő nyilak elhelyezése is némileg több módot ad, mint az Enterprise Architect.

3.2.3 A csoportos munka támogatása

A csoportos munka támogatására a termékhez járó VPository nevű felhőszolgáltatást nyújtja a cég, melyre verziókezelő rendszerekhez hasonlóan lehet feltölteni és leszinkronizálni a projectet. A konkurens változtatások kezelését is megoldja a rendszer. Mód van egy ehhez hasonló szerver beüzemelésére is, amennyiben a felhőszolgáltatást nem tartjuk megfelelőnek.

3.3 Az elkészítendő rendszer a két létező rendszer tükrében

Az Enterprise Architect egy kiváló modellező eszköz, de inkább a modellezés alsóbb szintjeinek a kóddal való szinkronizálásában emelkedik ki. (Például van saját Visual Studio és Eclipse beépülő modulja is, ami az egyszerű kódgenerálás és visszaolvasáson túl további funkciókat is nyújt). A Visual Paradigm ezeken a pontokon némileg alul marad, hisz csak a Java és a C++ nyelvet támogatja, de a diagramok elkészítése némileg komfortosabb vele, továbbá a csoportmunkára is ad egyszerűbb megoldást. Mindkét szoftver támogatja a modellből történő dokumentációgenerálást, de az ellenkező irányt már kevésbé támogatják. Ezek a szoftverek valamilyen módon a szoftverprojectek megvalósításának számos szintjét támogatják.

Az általam létrehozandó szoftver módot kíván adni arra, hogy a követelményeket szövegként is meg lehessen alkotni. Célja továbbá, hogy egy reszponzív webes alkalmazásként számos platformról, és eszközről kezelhető legyen. Mindemellett megoldást kíván nyújtani arra, hogy a felhasználók folyamatosan, közel valós időben láthassák egymás munkáját, így az eszköz jóval kényelmesebb és produktívabb kollaborációt tehetne lehetővé. Ugyanakkor az elkészítendő eszköz a szoftverfejlesztés egy szűkebb részére, a követelményekkel való magas absztrakciós munkára kíván csak megoldásokat nyújtani.

4 A RENDSZERREL SZEMBEN TÁMASZTOTT KÖVETELMÉNYEK

4.1 Felhasználói szintű funkcionális követelmények.

Ebben a szakaszban kifejtem a rendszerrel szemben támasztott funkcionális követelményeket.

1.0 Projektek kezelése

A rendszernek módot kell biztosítania arra, hogy a felhasználó projekteket kezeljen a rendszerben.

Magyarázat: A projektek különítik el a felhasználó különböző jellegű munkáit. A projekt tartalmazza a hozzá tartozó összes követelményt, diagramot és modell elemet.

1.1 Új projekt készítése

A rendszer módot kell, hogy biztosítson arra, hogy a felhasználó projektet vegyen fel a rendszerbe.

Megkötés: A rendszer, ne engedje olyan projekt felvételét, aminek nem adtak meg nevet.

1.2 Meglévő projekt szerkesztése

A rendszer módot kell, hogy biztosítson arra, hogy a felhasználó már létező projektet szerkesszen.

Megkötés: A rendszer, ne engedje, hogy a projekt nevét úgy módosítsák, hogy az ne tartalmazzon karaktert.

1.3 Projekt törlése

A rendszer módot kell, hogy biztosítson arra, hogy a felhasználó projektet töröljön.

Megkötés: A rendszer, ne engedje azon projektek törlését, amiknek van tartalma.

1.4 Projekt megnyitása

A rendszer módot kell, hogy biztosítson arra, hogy a felhasználó megnyisson egy projektet, és ez után láthassa az adott projekt tartalmát.

Megkötés: A rendszer csak azon felhasználóinak küldjön eseményt egy projekt változásairól, akik megnyitották azt.

Megkötés: Ha a felhasználó úgy kíván megnyitni egy projektet, hogy az adott ablakban van már nyitott projektje, a rendszer előbb zárja be az előzőleg megnyitott projektet

1.5 Projekt bezárása

A rendszer módot kell, hogy biztosítson arra, hogy a felhasználó bezárhasson egy már megnyitott projektet, és ez után láthassa az adott projekt tartalmát.

Megjegyzés: A rendszer a felhasználónak a továbbiakban ne küldjön értesítést az adott projekt változásairól.

2.0 Projektek kezelése

A rendszernek módot kell biztosítania arra, hogy a felhasználó projekteket kezeljen a rendszerben.

Magyarázat: A projektek különítik el a felhasználó különböző jellegű munkáit. A projekt tartalmazza a hozzá tartozó összes követelményt, diagramot és modell elemet.

1.1 Új projekt készítése

A rendszer módot kell, hogy biztosítson arra, hogy a felhasználó projektet vegyen fel a rendszerbe.

Megkötés: A rendszer, ne engedje olyan projekt felvételét, aminek nem adtak meg nevet.

1.2 Meglévő projekt szerkesztése

A rendszer módot kell, hogy biztosítson arra, hogy a felhasználó már létező projektet szerkesszen.

Megkötés: A rendszer, ne engedje, hogy a projekt nevét úgy módosítsák, hogy az ne tartalmazzon karaktert.

2.0 Követelmények kezelése

A rendszernek módot kell biztosítania arra, hogy a felhasználó követelményeket vegyen fel a rendszerbe.

Megjegyzés01: A rendszer által támogatott követelmény sablon a Felhasználói Sztori sablon.

Megjegyzés02: A rendszer a CRUD műveleteket a projektével megegyező módon támogatja a követelményeknél is.

Megjegyzés03: Amennyiben a felhasználó a szereplőhöz a rendszerben még nem létező nevet ad meg, a rendszer automatikusan vegyen fel úgy szereplőt a megadott néven.

Megkötés01: A követelmény nevére a projekt nevével megegyező megszorítások érvvényesek.

Megkötés02: A követelményhez kell, hogy tartozzon szereplő, aki végrehajtja a sztorit.

3.0 Diagramok kezelése

A rendszernek módot kell biztosítania arra, hogy a felhasználó diagramokat kezeljen a rendszerben

Megjegyzés01: A rendszer által támogatott követelmény sablon a Felhasználói Sztori sablon.

Megjegyzés02: A rendszer a CRUD műveleteket a projektével megegyező módon támogatja a diagramoknál is.

Megjegyzés03: A rendszer a megnyitás illetve a bezárás műveleteket a projektével megegyező módon támogatja a diagramoknál is.

4.0 Modellezés

A rendszernek módot kell biztosítania arra, hogy a felhasználó a követelményeket modellezze

Megjegyzés01: A rendszer által támogatott követelménymodellezési eljárás az egyszerű használati eset diagram.

*Megjegyzés01: A rendszer támogatja a **szereplők** illetve a **használati esetek** diagramra való felvételét.*

4.2 Felhasználói szintű nemfunkcionális követelmények.

Ebben a szakaszban megadom a rendszer egyes nemfunkcionális követelményeit, és felvázolom a rendszer határait és megkötéseit.

- A megvalósítandó rendszer egy követelmény specifikációs és követelmény analízist támogató intelligens, elosztott eszköz.
- Az eszköz kell, hogy rendelkezzen webes felhasználói felülettel, mely lehetőleg minél több platformon helyesen jelenik meg.

- A felhasználó által rendszerrel végzett munkát, minél előbb, akár a munkavégzés pillanatában, közel valós időben láthassák a rendszer további felhasználói.
- A rendszernek módot kell biztosítania a további bővítésének megkönnyítésére. Erre mind a kliens oldali, mind a szerver oldali tervezésnél gondolt kell fordítani.

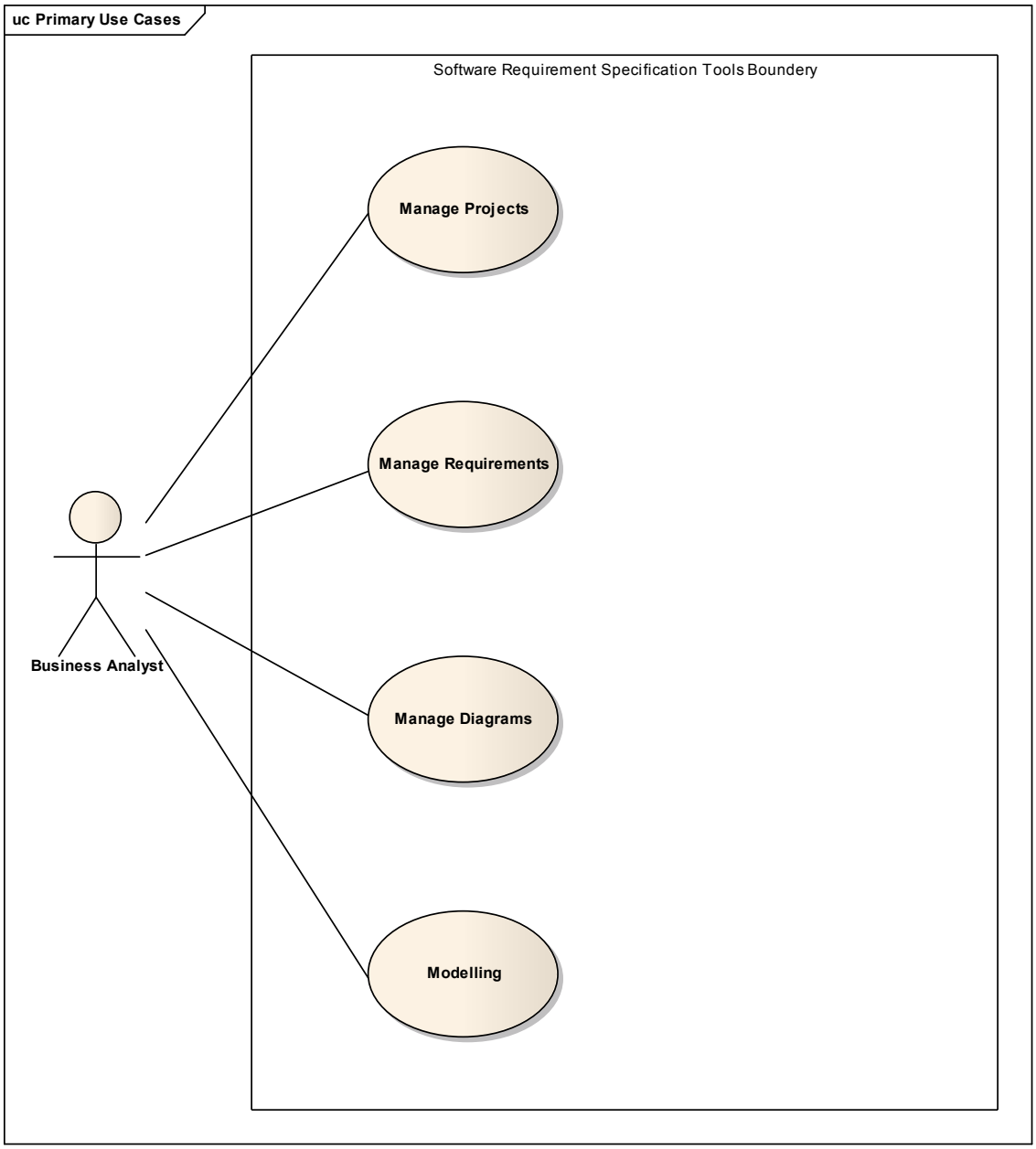
4.3 A rendszer határai

A rendszer egy megvalósíthatósági példa, mely azt kívánja bizonyítani, hogy a jelenlegi webes eszközökkel egy ilyen komplexitású CASE eszköz megvalósítható.

A rendszer jelen verziójában nem támogatja a felhasználói hitelesítést sem a felhasználói szerepköröket, ezekre a legtöbb platform kínál kész megoldásokat, ezért ez a dolgozat hatókörén kívül esik. A rendszer azt feltételezi, hogy a bejelentkezés már megtörtént és a kommunikáció biztonságosságát egy másik réteg biztosítja. Természetesen az autentikáció és authorizáció egy esetleges későbbi továbbfejlesztés részét kell, hogy képezze.

A rendszer felhasználóját az UML diagramokon egy üzleti elemző (**Business Analyst**) szerepkörrel modellezem.

4.4 A rendszer főbb felhasználói funkcióinak áttekintő használati eset nézete



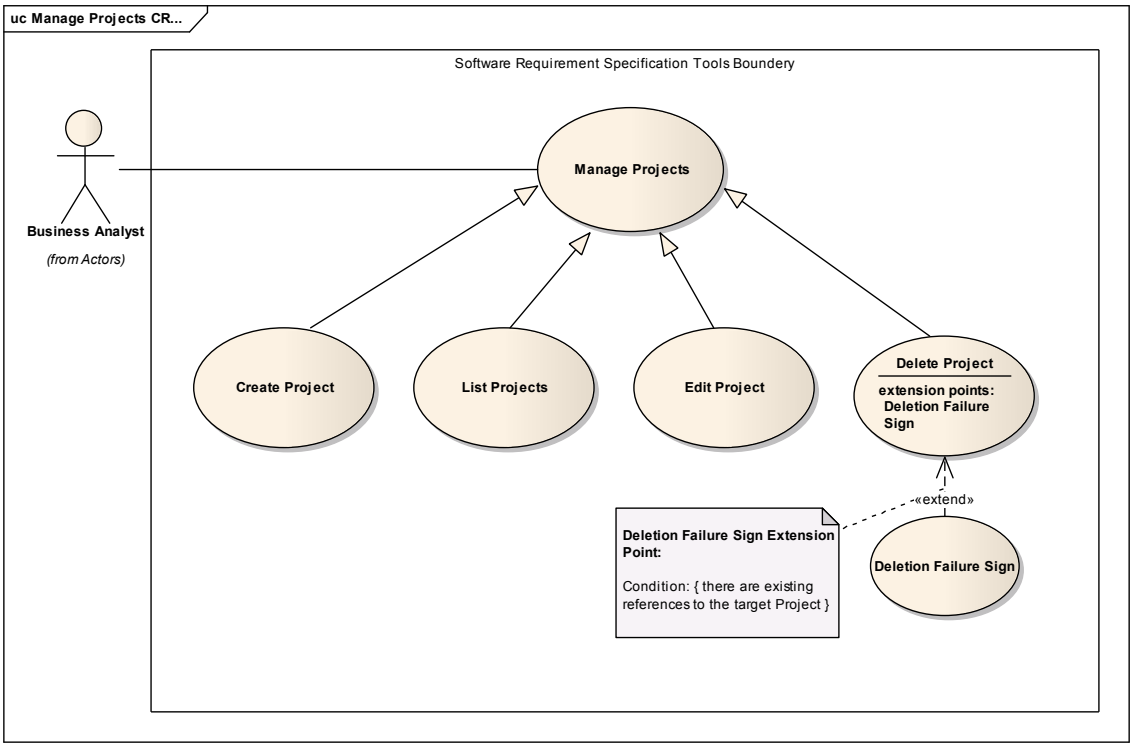
2. ábra: A felhasználói funkciókat áttekintő használati eset diagram.

Rendszer felhasználójának négy nagy feladatköre lehet, amik sorra Projektek kezelése (Manage Projects), Követelmények kezelése (Manage Requirements), Diagramok kezelése (Manage Diagrams), Modellezés (Modelling).

4.5 A részletezett rendszerfunkciók használati esetei

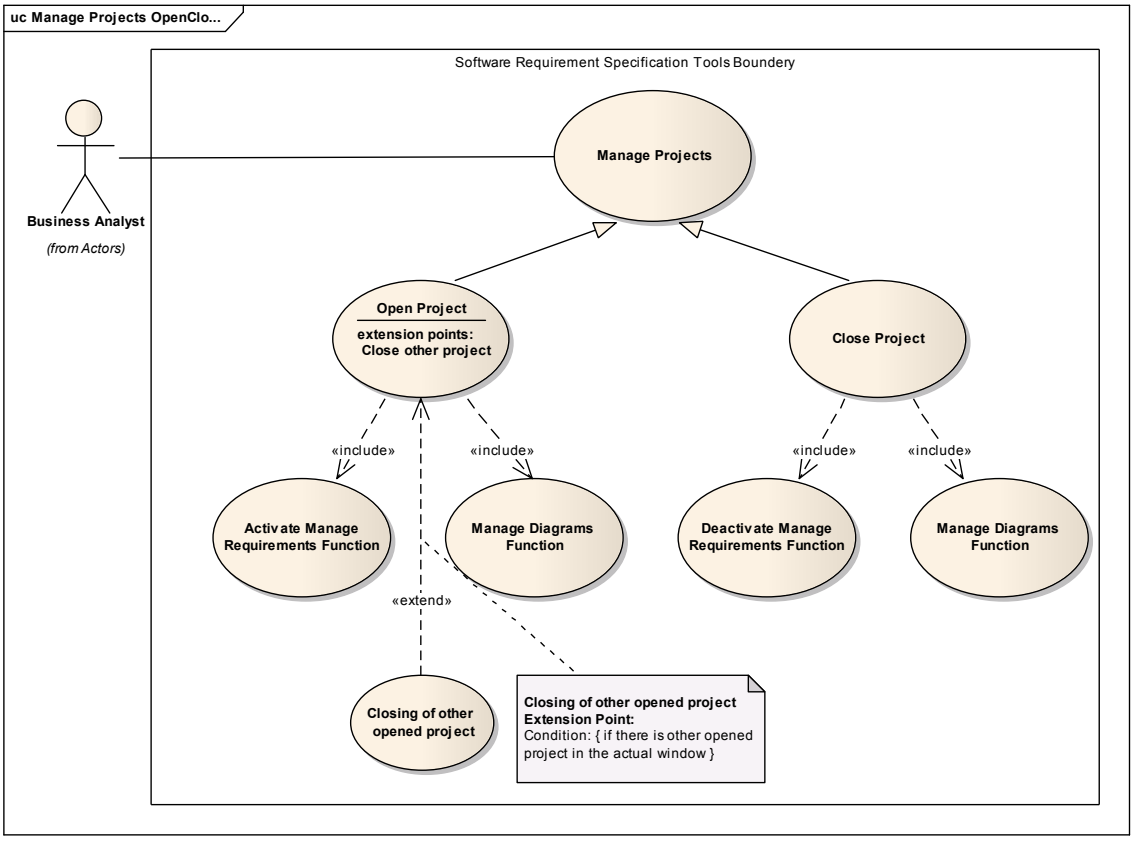
A fő kezelési funkciók általában elemek felvételét, listázását, módosítását, és törlését (CRUD) tartalmazzák, azzal a megkötéssel, hogy téves törlések megakadályozása végett a rendszer megakadályozza a törlést, amennyiben törlendő elem tartalmaz kapcsolódó elemeket.

4.5.1 Projektek kezelése



3. ábra: Projektek kezelése funkció - használati eset diagram (CRUD)

A projekt kezelése során az üzleti elemző felvehet új projektet, böngészheti a már meglévő projekteket, szerkeszthet már meglévő projekteket, és törölhet projektet. A rendszer ellenőrzi a project törölhetőségét, és hibajelzést ad, amennyiben az nem törölhető.

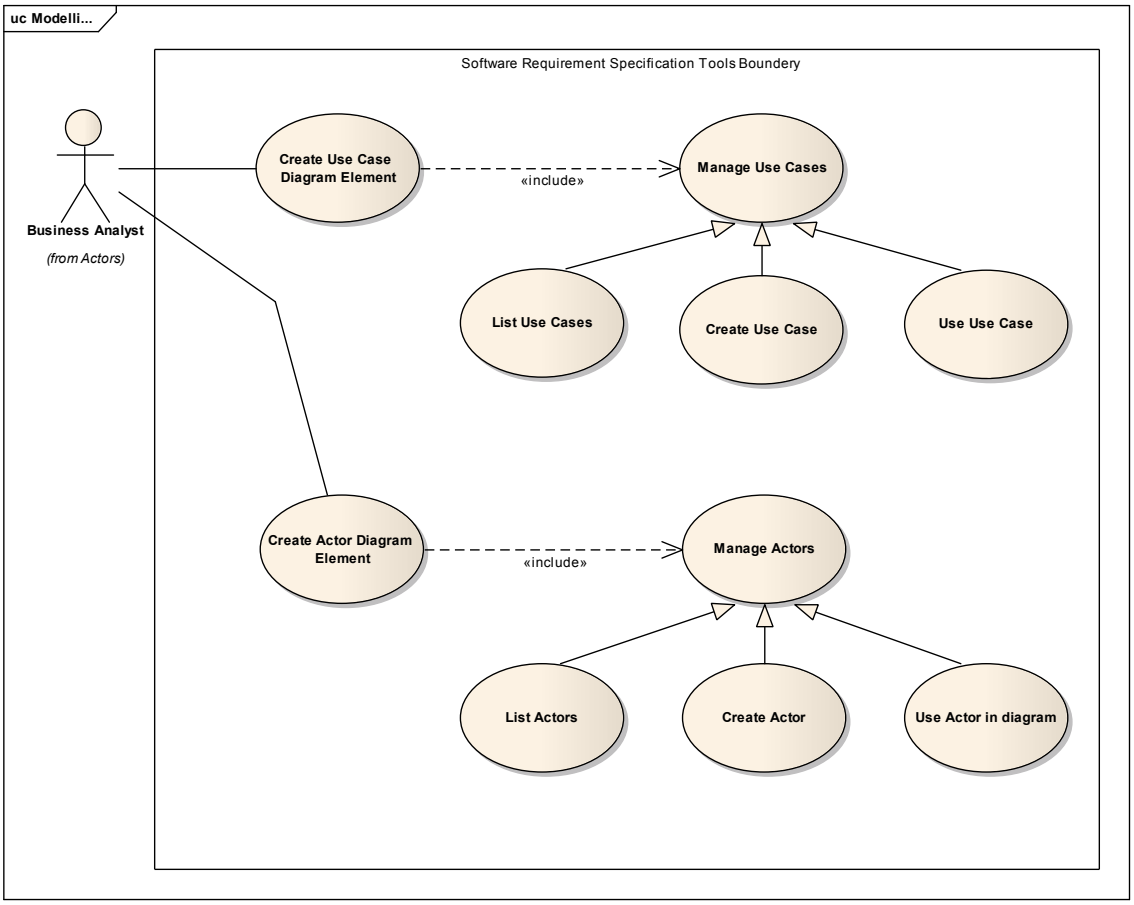


4. ábra: Projektek kezelése funkció - használati eset diagram (Megnyitás/Bezárás)

A projektek kezelése közt szerepel továbbá a projekt megnyitása és bezárása. A Követelmények kezelése és a Diagramok kezelése funkciók csak nyitott projekt mellett válnak elérhetővé, és csak a megnyitott project elemeire vonatkoznak. A felhasználó egy ablakban csak egy projektet nyithat meg, de annak nincs akadálya, hogy másik ablakban másik projektet nyisson meg. Amikor a felhasználó új projektet kíván megnyitni, a rendszer gondoskodik az esetlegesen előzőleg megnyitott projektek bezárásáról.

A Diagramok kezelése a projektkezeléssel nagymértékben analóg, de a diagram megnyitásakor a Modellezés funkció kerül aktiválásra.

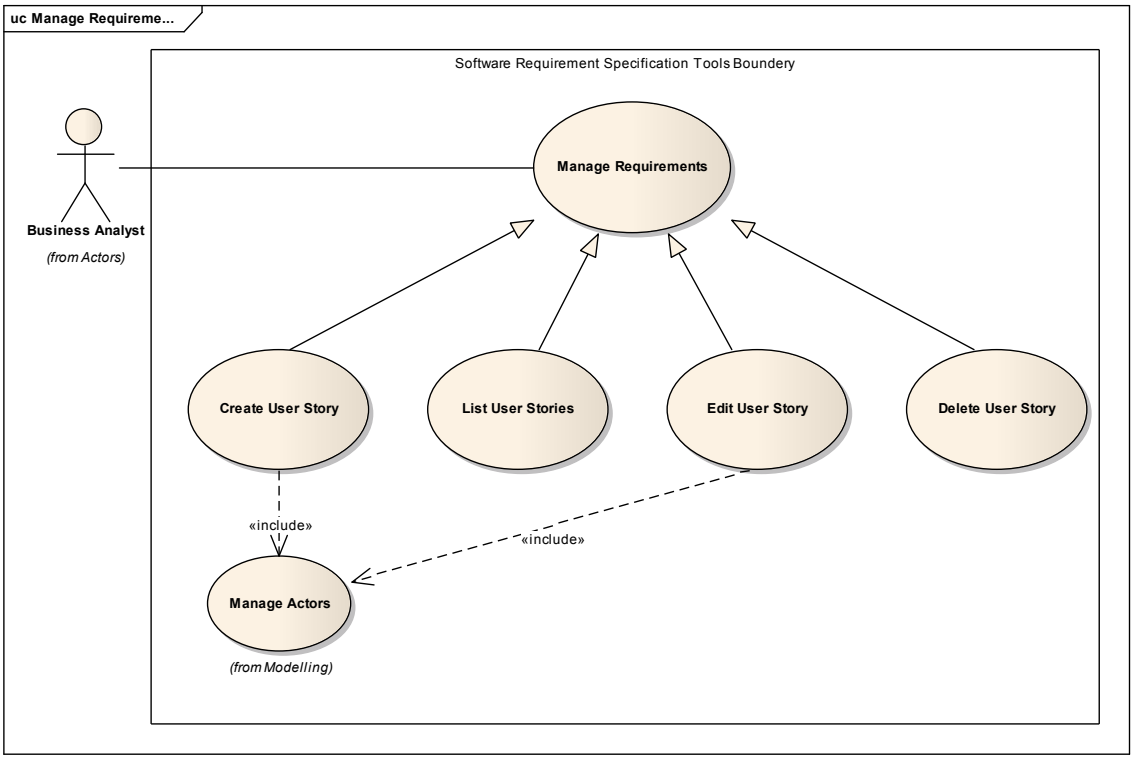
4.5.2 Modellezés



5. ábra: Modellezés funkció - használati eset diagram

Amikor az üzleti elemző a diagramra új használati esetet kíván felvinni, akkor eldöntheti, hogy már egy kész használati esetet visz fel a diagramra, vagy egy újat készít, amit a rendszer rögtön fel is vesz a diagramra. A szereplők felvitele is ehhez hasonló módon történik.

4.5.3 Követelmények kezelése



6. ábra: Követelmények kezelése funkció - használati eset diagram

Az üzleti elemző a követelmények kezelése részben felvehet új felhasználói sztorit, listázhatja az eddig felvitt sztorikat, szerkeszthet és törölhet is egy már meglévő sztorit.

Az felvitel és a módosítás közben ki kell választani egy felhasználói szerepkört, amire vonatkozik a sztori, itt vagy egy már meglévő szerepkört választ, vagy ha még nem létezőt vesz fel a sztorihoz, a rendszer automatikusan felveszi az új szerepkört is.

5 A RENDSZER KIFEJLESZTÉSE KÖZBEN ALKALMAZOTT MÓDSZERTANOK

A manapság legelterjedtebb életciklus modellek mind iteratív jellegű, ciklikus folyamatok. Ezekre általában igaz, hogy egyszerre a rendszer egy részét veszik, és ezen végigviszik a fejlesztés lehetőleg összes lépését, majd az elkészült produktumot a felhasználónak meg is mutatják véleményezésre.

A napjainkban divatos Agilis elveket követve igyekeztem a rendszert kisebb, a felhasználó számára értékeket hordozó egységekre bontani, ezeket súlyozni, és fontosságuk, illetve kockázatuk szerinti sorrendben elkészíteni. A project követésére a Kanban módszert választottam. A fehér táblám (white board) egy részét kijelöltem kanban táblának, és ezt a részt Backlog, To-Do, Doing, Done oszlopokra osztottam. Felmértem a hátra lévő magas szintű feladatokat (epikusokat), és felírtam ezeket öntapadós cetlikre, mindet felragasztottam a Backlog oszlopra, ami a hátralévő feladatokat reprezentálta. Majd fontosság és kockázat szerint az első hármat betettem a To-Do oszlopba, mely olyan elemeket tartalmaz, melyekkel minél hamarabb célszerű foglalkozni. A Doing oszlopba azt tettem, amikkel foglalkoztam. A To-Do és a Doing oszlopban egyszerre 3-3 feladat lehet (WIP – Work In Progress). Amikor egy-egy feladat a Doing oszlopba került, rendszerint felbontottam kisebb részfeladatokra, és újrarendeztem a táblát. Így áramoltak a feladatok a project alatt a kész állapot felé.

A fejlesztés kezdetén megvizsgáltam, hogy az egyes feladatok megoldására milyen létező technológiák elérhetőek. Ezeket összevetettem, és kiválasztottam közülük azt, amelyik az adott körülmények között a legjobb választásnak tűnt. Ezek közül a technológiák közül, azokat, amelyek újak voltak számomra, nagy kockázatú elemként kezeltem, és igyekeztem a fejlesztés kezdeti szakaszaira, legalább egy prototípust készíteni velük, illetve elolvastam velük kapcsolatban több fejlesztési útmutatót. Ez segített kiszűrni az esetlegesen mégsem alkalmas technológiákat, illetve így a felhasznált technológiák sajátosságait figyelembe véve könnyebb volt később tervezni velük.

A tervezés során már igyekeztem a teljes rendszer sajátosságait is figyelembe venni, és alkalmazni a szakmában elterjedt praktikákat és mintákat. A kezdeti tervet később újratervezéssel finomítottam.

6 A RENDSZER KIALAKÍTÁSA SORÁN FELHASZNÁLT TECHNOLÓGIÁK ÉS JELLEMZÉSÜK

A követelményeket véleményem szerint egy webes alkalmazással lehet leginkább lefedni. A weboldalak számos eszközről könnyen elérhetőek, így a megoldás kliens oldala platform független. Így a tervezés kezdeti szakaszában eldöntöttem, hogy kliens szerver architektúrát használok majd. Ahol az üzleti logika tetemes részéért a vastag kliens felel majd.

6.1 Szerver oldali technológiák

6.1.1 ASP.NET

Az ASP.NET a Microsoft webes keretrendszere, mely tökéletesen alkalmas akár nagyvállalati szintű webes alkalmazások létrehozására.

Egyik előnye, hogy meglehetősen jó eszközkészlettel támogatott. A Visual Studio egy komfortos környezetet biztosít a webes alkalmazások fejlesztésére. A .NET platformra jellemző, hogy a fejlesztés legtöbb szakaszára léteznek bevett szokások, és a Microsoft által ajánlott technológiák. Mindemellett egy egyszerű eszköz is biztosított, hogy külső féltől származó csomagokat (NuGet) egyszerűen installáljunk a programunkba.

Konkurens technológiák, mint a Java platform (Java EE vagy Spring MVC), a RUBY vagy a mostanában egyre népszerűbb JavaScript alapú Node.Js eszközkészletében, a fejlesztői támogatásában, vagy a platform kiforrottságában véleményem szerint elmaradnak az ASP.NET környezettől.

Választásomat mindezek mellett az is befolyásolta, hogy az ASP.NET környezetben van a legnagyobb szakmai tapasztalatom, így ez bír a legkisebb kockázattal a fejlesztésre nézve.

6.2 A kliens oldali megjelenítés kezelése

6.2.1 HTML5 és CSS

A webes kliensek megjelenését HTML és CSS webes (W3C) szabványok felhasználásával alkottam meg.

A felbontástól minél inkább független, úgymond reszponzív webes kliensek készítését támogató Twitter Bootstrap webes kezelőfelület keretrendszert (front-end framework) is igénybe vettem.

6.2.2 Rajzolás egy HTML oldalon

A webes rajzolásra három technológiát vizsgáltam meg. Elsőként a CSS3 képességeit vizsgáltam meg. A CSS3 segítségével HTML elemek transzformálásával lehet rajzolni. Előnye, hogy így a rajz tisztán HTML elemekből áll, és a DOM (Document Object Model) elemek JavaScript-ből egyszerűen elérhetőek. Hátránya viszont, hogy a komplexebb alakzatok megalkotására nem, vagy csak nagyon nehezen alkalmazható.

A másik, modern böngészők által támogatott technológia, a HTML5 Canvas, mely kis terhet ró a böngészőre, így alkalmas gyorsan mozgó animációk megrajzolására, de hátránya, hogy a kirajzolt elemek nem objektumok, nincsenek beépített eseményeik. Ezt külső keretrendszer (pl: paper.js) vagy saját kezűleg kell megírni.

A harmadik megvizsgált technológia az SVG (Scalable Vector Graphics) mely egy XML (Extensible Markup Language) alapú leírónyelv. A leggyakrabban használt formákra, mint az ovális, a téglalap, vagy az egyszerű vonal tartalmaz külön elemet, így a HTML és CSS3 kombinációjánál kényelmesebb a használata, ellenben rendelkezik saját DOM-mal és kiválthatnak eseményeket is. A Canvas-nál némiképp nagyobb terhet ró a böngészőre, de a diagramokon jellemzően nem megy végbe egyszerre annyi gyors mozgás, hogy ez egy mai rendszeren különösebb gondot okozzon, így végül az SVG technológiát választottam.

6.3 A kliens oldali dinamika kezelése

Napjaink webes alkalmazásainak jellemző kliens oldali programozási nyelve a JavaScript, amit a legtöbb korszerű böngésző natív módon támogat. A beépülő modulokat igénylő nyelvek, mint az ActionScript (Flash), illetve a Java (applet) egyre inkább háttérbe szorul.

A JavaScript gyengén típusos mivolta miatt nehezen alkalmazható nagyméretű projektekbe. Ezért a Microsoft elkészítette a TypeScriptet, ami egy a Java a C# nyelvekhez hasonló alapú OOP eszközökkel, és szintaktikával rendelkező nyelv. Ilyen eszközök például a modul az osztály, az interfész, a láthatósági szintet állító kulcsszavak. A nyelv nagy erőssége az, hogy visszafelé kompatibilis a JavaScript felé, tehát ami működő JavaScript kód az működő TypeScript kód is egyben, a TypeScript annotációkból a fejlesztői környezet plusz információt nyer, illetve a TypeScript fordító kiszűri az esetleges hibákat és JavaScript kódot állít elő a helyes TypeScript kódból. A nyelv jelenleg nyílt forráskódú és a Microsoft mellett a Google is támogatja a fejlesztését. A szoftver elkészítése során a kliens oldalon nagymértékben kihasználtam a TypeScript előnyeit.

A JavaScript kód szervezésére az AngularJs-t választottam. Ez napjainkban talán leginkább elterjedt SPA (Single Page Applicaion) keretrendszer, mely egy úgynevezett

MV* keretrendszer, ami azt jelenti, hogy használható MVC illetve MVVM jellegű szervezésre is. Előnye, hogy a DOM manipuláció felelősségét leveszi a fejlesztő válláról. A HTML nézet és a JavaScript háttérobjektumok között a kapcsolatot adatkötésekkel adhatjuk meg. Az adatkötést deklaratív módon, a HTML elem attribútumában jelezhetjük. A keretrendszer lehetőséget ad a HTML nyelv mintegy kibővítésére, saját címkék létrehozásával. Ezeket a címkéket hívják direktíváknak, amiknek a legfőbb jelentősége, hogy egy funkciót és a hozzá tartozó megjelenést lehet egységbe zárni, és HTML elemként újra felhasználni. A keretrendszer továbbá lehetőséget ad arra, hogy funkcióink egy részét, a kontrollerből szolgáltatásként kiemeljük. Az ilyen szolgáltatások (service) függőség beinjektálás (Dependency Injection) segítségével a rendszer több pontján újra felhasználhatóak. A tervezéskor érdemes azt is figyelembe venni, hogy az ilyen szervizek az egyke tervezési-mintát követik.

6.4 Kommunikációs technológiák

A hagyományos http kérelem válasz alapú kommunikációban megszokott, hogy csak a kliens szólíthatja meg a szervert. Amennyiben az a cél, hogy a kliens a változásokról közel valós időben értesüljön az esetleges változásokról az eddigi modellben különböző taktikákat kellett kialakítani (például gyakori AJAX kérelmek az esetleges változás felmérésére)

A HTML5 megjelenésével egy új technika a WebShocket került kifejlesztésre, mely egy full-duplex kapcsolati csatorna a szerver és a kliens között. Napjainkban a technológia támogatása egyre javul, és a modern böngészők támogatják, de a szerver oldali támogatása sokszor nem megoldott (Például csak IIS 8.0 és annál újabb szerver).

A SignalR egy olyan keretrendszer mely arra hivatott, hogy elfedje a kommunikáció tényleges megvalósítását, amennyiben él a WebShocket kapcsolat, akkor azt használja, vagy automatikusan átáll egy olyan lehető legjobb kommunikációs formára, ami mind a szerver, mind a kliens által támogatott. Mindemellett további előny, hogy egy további absztrakciós rétegben (HUB) egyszerűbbé teszi a kommunikációt azzal, hogy kliens csonkokat biztosít a kliens oldal felől történő kommunikációra, és a szerver irányából kezdeményezhető kéréseket is megkönnyíti. SignalR az ASP.NET platform szerves része, így a választott környezetben az adott problémára jelenleg a legjobb megoldást nyújtja.

7 RENDSZERTERV

Ebben a részben tárgyalom a legfőbb tervezési döntéseimet, és azok okát. Elsőként a megoldás könyvtárszerkezetének jelentősebb részeit tekintem át, majd rátérek a szerkezet és a dinamizmus felvázolására. Végül rátérek a részletes tervekre és azok magyarázatára.

7.1 Az elkészült szoftver könyvtárstruktúrája.

Már a tervezés első szakaszában utánanéztem, hogy hogyan érdemes szervezni egy ASP.NET alkalmazást.

A megoldás (Solution) neve SoftwareRequirementsTool, mely két projektet tartalmaz, Az egyik egy osztálykönyvtár típusú projekt, aminek a neve SoftwareRequirementsTool ez hivatott szerver oldali háttérlogika tárolására. A másik projekt a Visual Studio ASP.NET Web API sablonjára épül és a neve: SoftwareRequirementsTool.Web.

A SoftwareRequirementsTool gyökerében csak könyvtárak szerepelnek. Az üzleti logika tárolására hivatott Business. A Data könyvtár az adathozzáférési réteg osztályait tárolja. A Services mappán belül kell elhelyezni a külső szolgáltatásokhoz való kapcsolatot biztosító alkalmazás részeket. Végül az Utilities, ahol a különböző segédkomponensek találhatóak. Jelenleg az üzleti logika legnagyobb hányada a vastag kliensen található és külső szolgáltatást sem veszek igénybe, így a Data, illetve az Utilities könyvtárat leszámítva a többi helyőrzőként szerepel.

Az ilyen szervezés előnye, hogy míg kisméretű, nincs minden könyvtárnak külön osztálykönyvtára így a megoldás lehetőleg gyorsan fordul. Később, ha mondjuk az adatelérési réteget kiszerveznénk külön SoftwareRequirementsTool.Data nevű projectbe, akkor a benne szereplő osztályok teljes neve (névtere) nem változna. Véleményem szerint ez a megközelítés a méret növekedésével is jól skálázódna.

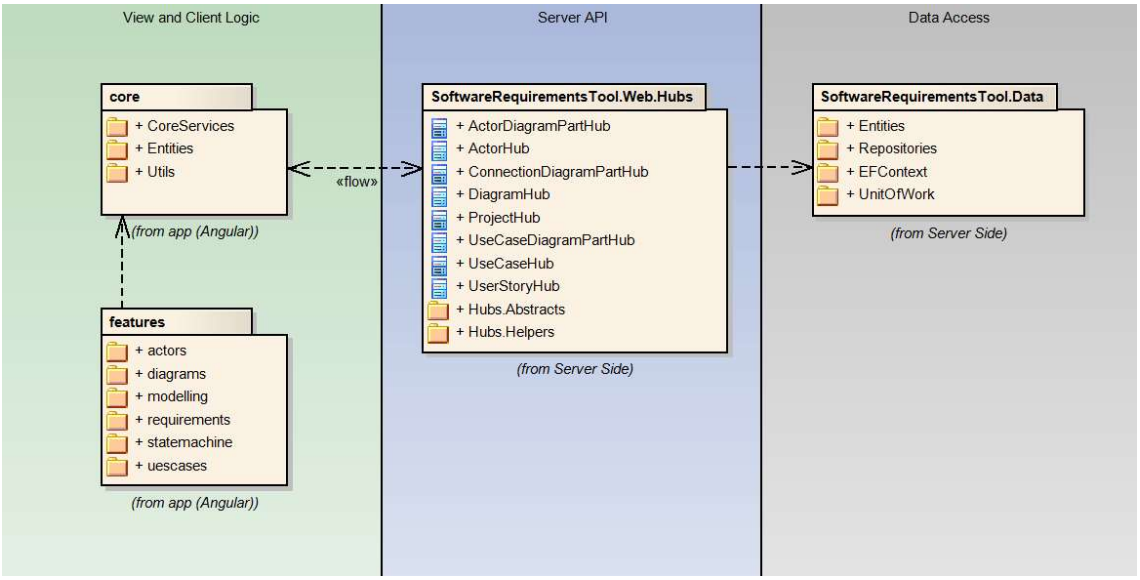
A SoftwareRequirementsTool.Web project alapvető felépítése tükrözi az ASP.NET konvenciókat, de azt kiegészítettem további két könyvtárral, a SignalR Hub-okat tartalmazó Hubs mappával, illetve az általam írt kliens oldali alkalmazáslogikát és felhasználói felület részeket tartalmazó app mappával.

Az app mappa alapvető az AngularJs közösség által bevett gyakorlatként használt mappastruktúrákat adott feladathoz szabva készítettem. A blocks alkönyvtár az alkalmazás szerte és az alkalmazások között is megosztható, újrafelhasználható és az adott feladattól független segédkódokat tárolja. A features mappa tárolja a rendszer egyes funkcióit külön-külön almappánként egységbe zárva. A layout mappa tárolja azokat a megjelenési elemeket, amiket több aloldalon is használok. A services mappa olyan alkalmazásfüggő AngularJs kódokat tartalmaz, melyet több funkcióban is

felhasználók. A widgets olyan direktívákat tartalmaz, melyet alkalmazás szerte több helyen is újra fel lehet használni. A cores mappa tartalmazza a teljes egészében TypeScript nyelven írt üzleti logikát és kliens oldali entitás modellt.

A könyvtárak tényleges tartalmát majd a későbbiekben, a megvalósítás részleteinél tárgyalom.

7.2 Szerkezet áttekintő nézet



7. ábra: Szerkezet áttekintő - rétegdiagram

A rendszert négy nagyobb alrendszerre bontottam, melyek mind további almodulokat tartalmaznak.

A kliens oldali kód teljes egészében a SoftwareRequirementsTool.Web projektben található. Ezen belül is az app mappa core almappájában található a szerverrel történő kommunikációt biztosító CoreServices modul, ami fizikailag az api illetve az instantiation almapákban helyezkedik el. A core alrendszer további moduljai az entitásokat magában foglaló Entities modul, illetve az Utils modul, ami különböző segéd osztályokat tartalmaz, többek között típusvizsgálatokra, tömbkezelés elősegítésére, illetve az objektumokkal végzett munka elősegítésére.

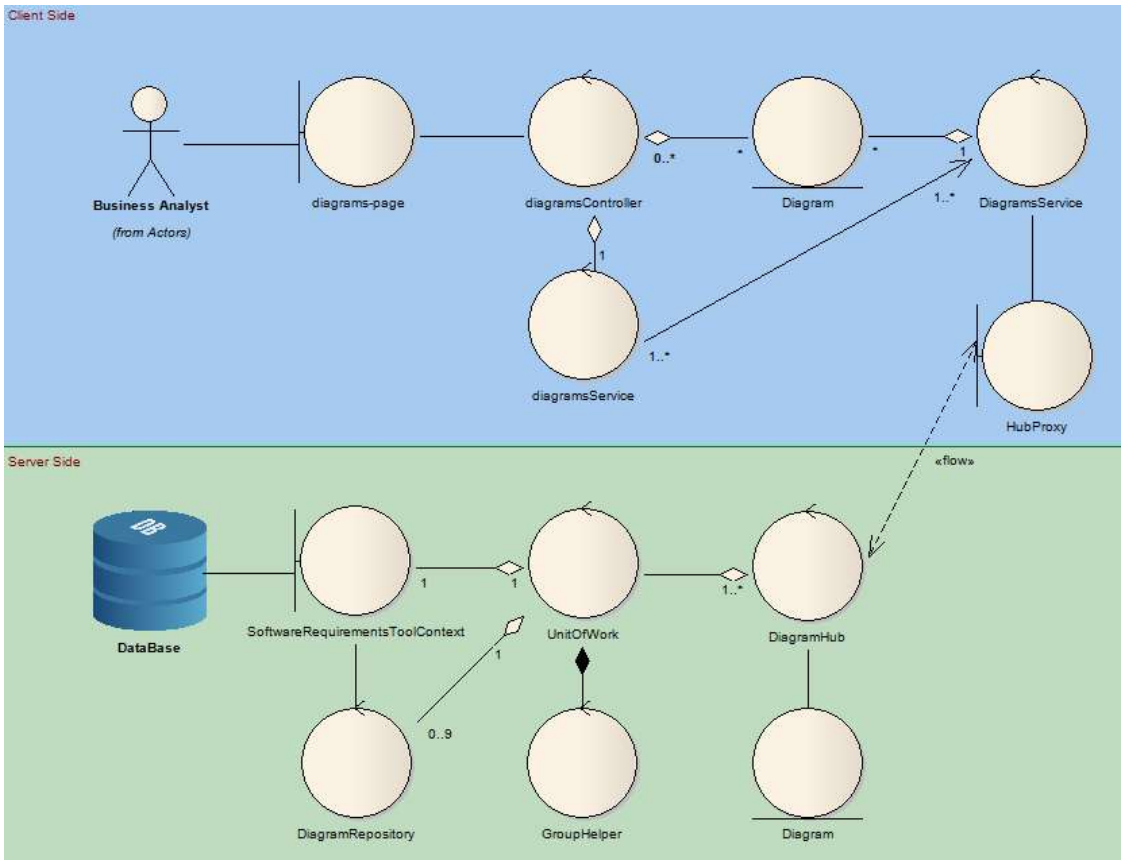
További fontos alrendszernek tekintem az app mappa features almappájában található kódbázist, mely a kliens oldali üzleti logikát tarzalmazza, rendszerfunkciónként külön almappában.

A Hubs alrendszer is a SoftwareRequirementsTool.Web projekt része, de ez már a szerver oldali kommunikációs felületet adja. A céljának elérése érdekében a SignalR keretrendszer szolgáltatásait is igénybe veszi.

A SoftwareRequirementsTool.Data névtér felelős a szerver oldali adatmodell kialakításáért, illetve az adatbázis és az adatmodell szinkronban tartásáért. Az adat elérési réteg az Entity Framework objektum-relációs leképző (ORM) technológiát használja.

7.3 Új diagram felvétele elemzésdiagram

A kliens szerver kommunikációt elősegítő szerkezet és a rendszerfelépítés bemutatását az egyik ténylegesen megvalósított funkción kersztűl vezetem be. Ez az „új diagram felvétele” funkció.



8. ábra: Szerkezet áttekintő - elemzésdiagram

Az Üzleti elemző egy HTML oldalon kereszül kommunikál a rendszerrel. A diagrams-page oldal és az AngularJs diagramsController modelje kétirányú adatkötés segítségével tartja a kapcsolatot. A diagramsController minden oldalnavigációnál újra inicializálódik, és több oldalhoz is köthető. Tartalmaz egy diagramsService nevű szolgáltatást, ami alkalmazás szerte egyke (Singleton) így az első beinjektálás után az alkalmazás futása alatt életben marad. A kontroller referenciát tárol a rendszerben tárolt diagramokra. A DiagramService egy TypeScriptben írt szolgáltatás, a szerveller történő kommunikáció felelőse. A diagramService egyes feladatait delegálja a DiagramService-nek. A HubProxy felület azt a hub prokszit szimbolizálja, amit a SignalR a Hub alapján

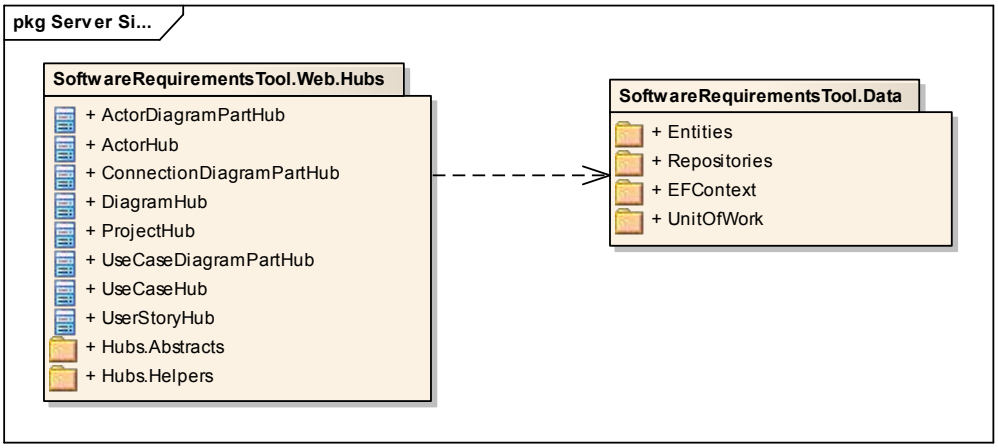
generál, és jelen példában a DiagramHub osztállyal tart fent két irányú kommunikációs kapcsolatot.

A szerver oldalon a kéréseket a Hub osztály fogadja, mely referenciát tárol a Diagramra. A hub kapcsolatban áll egy UnitOfWork-el, mely több Repository-t fog össze. A DiagramRepository adja ki a tényleges parancsot a SoftwareRequirementsToolContext-nek. Az adatbázis tényleges elérése a SoftwareRequirementsToolContext felelőssége.

7.4 A szerver oldal modellje

Ebben a szakaszban a szerver oldali kódbázis moduljait tekintem át csomag diagramok segítségével. A mellékletek „osztályhierarchiák” része alatt pedig elhelyeztem a legfontosabb kapcsolódó osztályhierarchiákat.

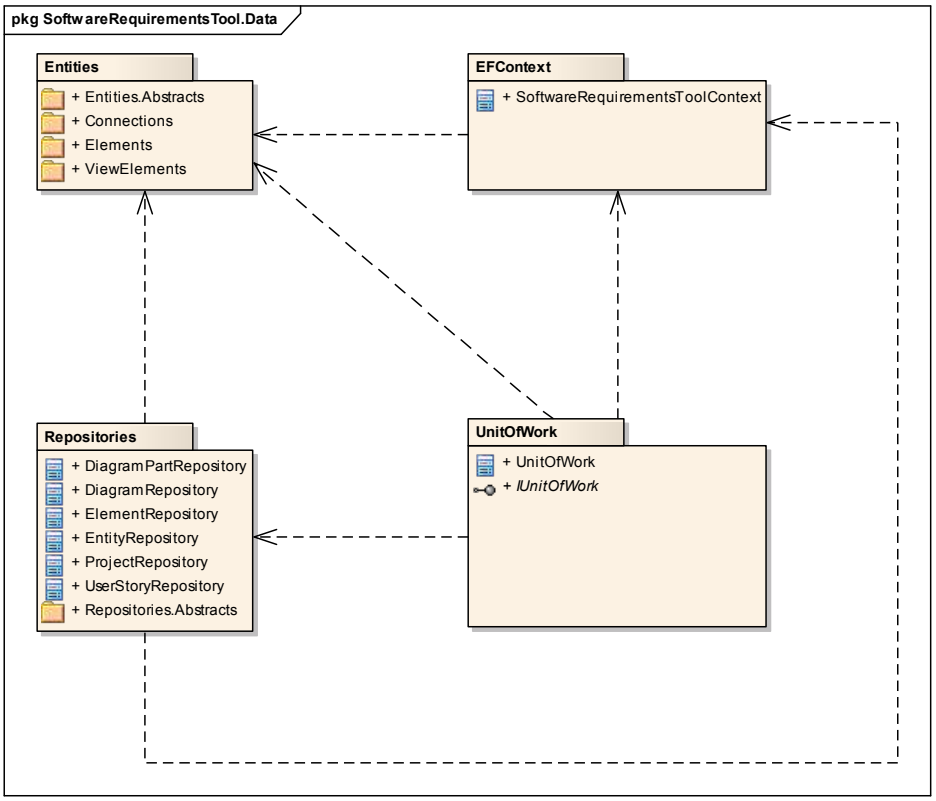
7.4.1 A szerver oldal moduláttekintő nézete



9. ábra: Szerver oldal áttekintése - csomagdiagram

A kliens szerver kommunikációért felelős Hub-ok nagymértékben építkeznek a Data névtér megoldásaira. Egyfelől a Data névtérből származnak a kommunikáció során felhasznált entitások, illetve az adatbázis elérését is a Data névtér konkrét Repository, illetve UnitOfWork megvalósításai adják.

7.4.2 Az adatelérési réteg részletesebb áttekintése



10. ábra: Adatelérési réteg - csomagdiagram

A Data.Entities névtér alatt találhatóak a rendszer szakirányú adatainak tárolásáért felelős osztályok. Ezekből épül fel az entitás modell.

A Data.EFContext névtérben található SoftwareRequirementsToolContext nevű osztály az Entity Framework DbContext osztályának konkretizálása, és azért felelős, hogy az adatbázist és az entitás modell adatait szinkronban tartsa.

Az adatelérés absztrakciójaként használtam a Repository tervezési mintát. Ez a minta arra szolgál, hogy egy adott entitáshoz tartozó adatbázis műveleteket egy helyen tárolja, így csökkentve a kódismétlés mértékét. Mindemellett elfedi az adatbázis elérés konkrét részleteit, így az e szint alatt található adatelérési rétegek egyszerűbben lecserélhetőek. A minta generikus változatát használtam így az alapvető műveleteket egy generikus ősosztály látja el, amit a konkrét leszármazottak típusparamétereznek fel, illetve adnak hozzá további szolgáltatásokat.

A UnitOfWork tervezési minta célja, hogy több Repository-t fogjon össze. Ez a tároló osztály gondoskodik a konkrét Repository példányok létrehozásáról. Ennek az előnye, hogy a konkrét Repository-k létrehozása csak igény esetén történik. A UnitOfWork gondoskodik továbbá arról, hogy a Repository-k közös Context-en osztozzanak. Ennek

az a legfontosabb hozadéka, több különböző entitáson végzett művelet vonható egy tranzakció alá.

7.5 A rendszert alkotó osztályhierarchiák jellemzése

Ebben a szakaszban részletezem a tervezett osztály hierarchiákat. A fontosabb diagramok a Mellékletek rész „Fontosabb osztályhierarchiákat leíró diagramok” alfejezetében találhatóak meg.

A rendszerben található adatok tárolására az Entities modul hivatott. Az Abstracts almoduljában találhatóak az entitáshierarchia gyökerét adó interfészek, melyekre az összes többi entitás osztálya. Az összes entitás származik az AbsEntity osztálytól. Ez az absztrakt osztály, csak egy Id és egy TypeName tulajdonságot biztosít. Az Id arra hivatott, hogy az adatbázis kezelésnél egységesen lehessen hivatkozni és kezelni az entitásokat. A TypeName arra hivatott, hogy a kliens oldal egyszerűen meg tudja állapítani az adott osztály eredeti típusát. Az Elements almodul tárolja a követelmény és modell elemeket, míg a ViewElements almodul a diagramelemeket tárolja. A diagramelemek tárolnak egy elemet. A diagram egy olyan elem mely diagram elemeket tárolhat, így a diagram a későbbiekben részdiagramokat is tárolhat (összetétel tervezési minta). A Connection Connection osztálya két entitást kapcsol össze.

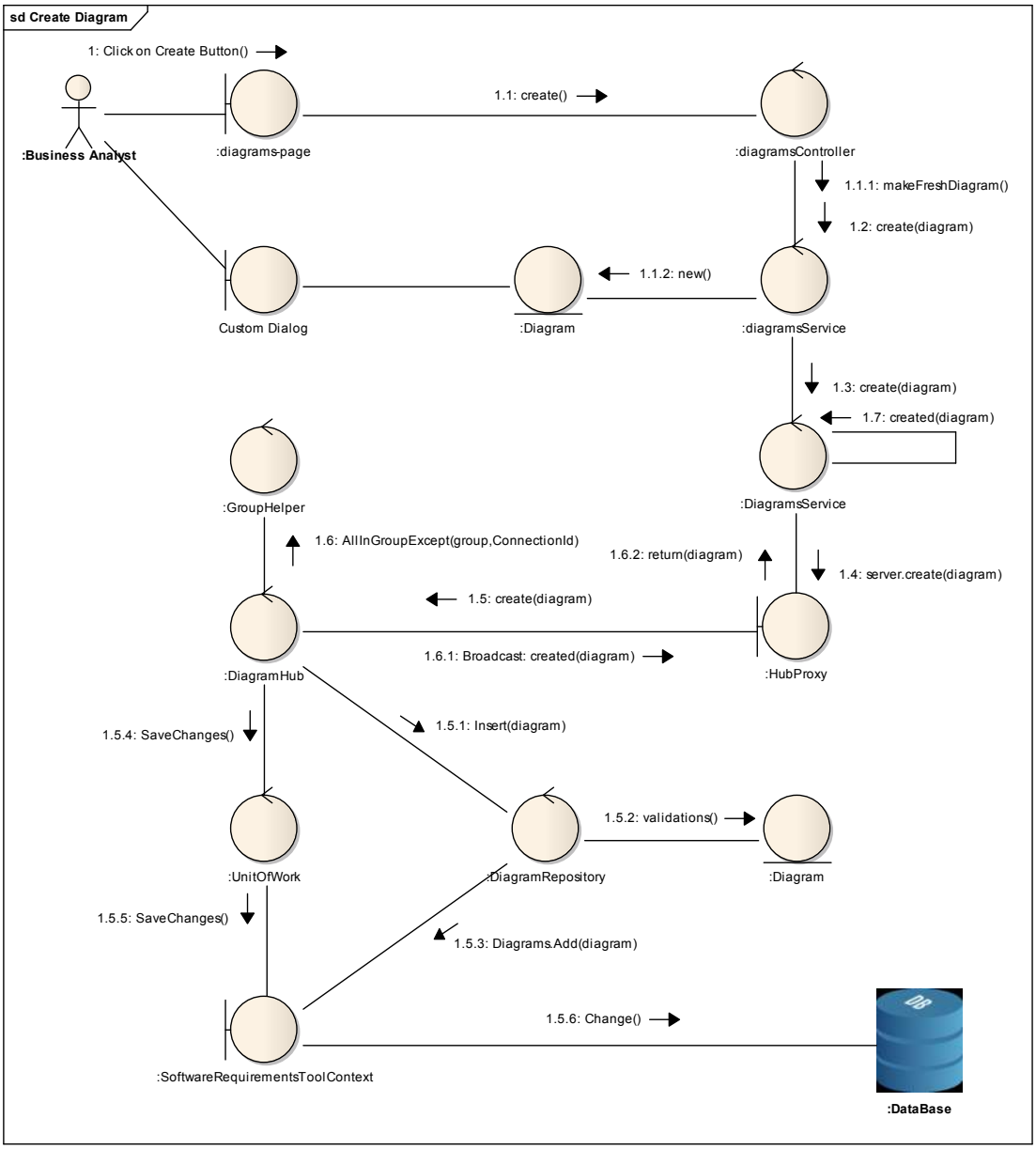
A Szerkezet áttekintő nézeten bemutatott modulok közül, a features modul kivételéve az összes többi modul hierarchiája némiképp tükrözi az Entities modul hierarchiáját, hisz mind valamilyen módon, ezeken az entitásokon dolgoznak, és ahogy egy ősentitás kiemel közös tulajdonságokat, úgy emelnek ki közös funkcionalitást a hozzá tartozó kontroller jellegű osztályok. A kontroller jellegű osztályhierarchiák (HUB, Repository, CoreServices) közös tulajdonsága, hogy a funkcionalitást elvégző osztályok a hierarchia legtetején találhatóak, és különböző horgokat adnak a leszármazottaiknak a kiterjesztésükre.

Erre kiváló példa az AbsCrudHub<T>, mely egy olyan generikus absztrakt ősosztály, mely a leszármazott típusaitól, vár egy típusparamétert, egy konkrét Repository-t és külső függőségként vár egy UnitOfWork-öt. A publikus interfésze absztrakt, melyet a leszármazottak az átadott típusparaméter függvényében megvalósítanak. Mindemellett meghatároz minden ilyen metódushoz egy hasonló szignatúrájú, de protected elérésű segédmetódust (pl: Create(T entity) - Create(T entity, string groupName)) melyeket a leszármazottak felparaméterezhetnek. A Create(T entity, string groupName) ráadásul egy példa a sablon függvény tervezési mintára, ugyanis a metódus meghatároz egy folyamatlépés sorrendet. Minden folyamatlépést egy olyan metódus meghívásával végez el, amit a leszármazott típusok felülírhatnak, ezzel megváltoztatva azt az egy lépést.

7.6 A rendszer dinamikájának elemzése

7.6.1 Új diagram felvétele áttekintő kommunikációs diagram

A kliens szerver kommunikáció dinamikájának a bemutatását az egyik ténylegesen megvalósított funkción kersztűl vezetem be. Ez az „új diagram felvétele” funkció. (A valós kommunikációban további elemek is részt vesznek, de ezeket a diagram és az általa leírt folyamat áttekinthetősége érdekében nem jelölök)



11. ábra: Rendszer dinamika - kommunikációs diagram

1. A felhasználó a Create New gombra kattint a felületen.
    1.1 Az oldalhoz kötött diagramController create() függvénye aktiválódik.
        1.1.1 A kontroller meghívja a diagramService makeFreshDiagram fuggvényét
        1.1.2 A diagramService létrehoz egy új Diagram példányt, amit a felhasználó egy dialógus ablak segítségével kitölthet.
    1.2 A kontroller meghívja a diagramService create függvényét, a frissen létrehozott diagrammal.
    1.3 A diagramService meghívja a DiagramServiceInstance példány create(diagram) függvényét.
    1.4 A DiagramService a HubProxy kliens csonk szerver irányába kommunikáló objektumának meghívja a create(diagram) függvényét.
    1.5 A HubProxy továbbítja az üzenetet a DiagramHub felé.
        1.5.1 A DiagramHub a DiagramRepository Insert(diagram) metódusát hívja
        1.5.2 A DiagramRepository példány ellenőrzi a kapott Diagram entitás integritását.
        1.5.3 A helyes Diagramot a SoftwareRequirementsToolContext Diagrams nevű kollekció tulajdonságához adja.
        1.5.4 A DiagramHub meghívja a UnitOfWork SaveChanges metódusát.
        1.5.5 A UnitOfWork a SoftwareRequirementsToolContext segítségével elindítja a mentési folyamatot.
        1.5.6 A SoftwareRequirementsToolContext az általa tárolt modell változásait SQL nyelvre transzformálja és végrehajtatja az adatbázissal.
    1.6 A DiagramHub a GroupHelper példány segítségével megszerzi azon kliensek listáját, akiket értesítenie kell, arról, hogy létrejött egy új Diagram.
        1.6.1 A DiagramHub értesíti a szükséges klienseket (az adott kliens HubProxy-ján keresztül meghívja a kliens DiagramService created(diagram) függvényét)
    1.7 Az eredeti hívó kliens DiagramService megkapja a létrehozott diagramot, mint visszatérési értékét,
    1.8 A DiagramService Meghívja a saját created(diagram) függvényét, ezzel beszúrva a diagrams tömbjébe azt. Ezután visszajelez a meghívójának (Callback).

8 AZ IMPLEMENTÁCIÓ RÉSZLETEI

Ebben a szakaszban vázolom a fejlesztés, megelőző szakaszait, a fejlesztés menetét és a fejlesztés közben tapasztalt fontosabb kihívásokat, illetve a rájuk adott megoldást.

8.1 A fejlesztői környezet leírása

8.1.1 Visual Studio

A fejlesztés során a Visual Studio 2013 Ultimate változatát használtam. A környezetet különböző kiegészítésekkel még inkább az adott fejlesztési feladat hatékony végzésére optimalizáltam.

Az egyik ilyen kiegészítés a webes projektek fejlesztését elősegítő web essentials. Legfőbb előnye, hogy kapcsolódni tud egyszerre akár több böngészőhöz is, és egy kiadható paranccsal (ctrl+alt+enter) frissíteni képes az összes böngészőt. További előnye, hogy javítja a Visual Studio webes technológiákhoz társított kódkiegészítőjét.

Az AngularJs fejlesztést nagymértékben segítette, hogy feltelepítettem a SideWaffle kiegészítőjét, mely számos sablont tartalmazott, így megkímélt az ismétlődő kódok örökös begépelésétől.

A Resharper a JetBrains cég egy méltán híres, de sajnos fizetős Visual Studio kiegészítése. Az eszköz harminc napos próbaverzióval rendelkezik. Mind a szerver oldalon, mind a kliens oldalon nagymértékben feljavítja a Visual Studio kód kiegészítési és kód navigációs képességeit. A kód javítására tett javaslataiból sokszor egy gyakorlott .NET fejlesztő is okulhat. Nagy előnye továbbá, hogy a viszonylag új TypeScript-hez is tartalmaz hasonló funkciókat. A JavaScript kód elemzése révén figyelmeztet egyes lehetséges hibákra, illetve olyan részekre, ahol felesleges, kód található. A lehetséges JavaScript hibák jelzése véleményem szerint számos alkalommal, hosszas és kimerítő hibajavítási procedúrát előzött meg.

8.1.2 Böngészők

Chrome: A fejlesztés során leggyakrabban ezt a böngészőt használtam. Az F12-vel előhozható menüje kényelmes HTML+CSS+JavaScript kezelést biztosított. A böngészőhöz elérhető a Batarang nevű kiegészítő, mely segítségével elérhetjük az AngularJs kontrollerek kontextusát ($scope), ezzel mind a hibajavítást, mind a fejlesztést nagymértékben elősegítve. A kontextus megfigyelésére egy másik kiegészítő az ng-inspector.

Internet Explorer: Az Internet Explorer nagy előnye, hogy ez a böngésző működik legszorosabban együtt a fejlesztői környezettel, képes a megállni azoknál a töréspontoknál, amit Visual Studio-ban helyezek el a JavaScript kódban. Így, amikor

hosszú és szerteágazó JavaScript kódrészeket kellett átnéznem, a Visual Studio komfortos környezetét választottam az Internet Explorer segítségével.

Firefox: A Firefox és a firebug kiegészítője széles körben elismert webes fejlesztői környezet, mely talán az egyik legjobb böngészőbe épülő fejlesztői modul. A rendszert a Chrome-nál ritkábban használtam, mert az összességében jobb fejlesztői élményt nyújtott AngularJs mellett.

Fontos megjegyezni, hogy a böngészők gyorsító tárazását érdemes a fejlesztés alatt kikapcsolni.

8.2 SignalR

A Hub-ok is hierarchiába vannak szervezve. Az alrendszer osztály diagramja a melléklet „Fontosabb osztályhierarchiákat leíró diagramok” része alatt megtalálható.

A hierarchia elkészítése közben akkor jött elő egy probléma, amikor az első ProjektHub-ból a funkcionalitást kiemeltem a generikus ős osztályba. A rendszer nem volt képes HubProx-kat generálni. Utánajártam, és a generikus osztályokat mindenképpen absztraktnak kell jelölni, ugyanis a rendszer nem tud generikus proxy-t generálni.

A második probléma a csoportos üzenetküldésben volt, ugyanis az üzenetek egyes esetekben nem mentek át. A SignalR beépített csoport kezelése nem mindig működött, és erre nem sikerült csak megkerülő megoldást találnom. Elkészítettem egy saját GroupHelper osztályt, mely egy szál biztos késői betöltésű egyke mintát valósít meg a .NET keretrendszer Lazy<T> osztálya segítségével.

A Hub-ok a konkrét UnitOfWork-ot a Ninject Függőség befecskendezés segítségével kapják meg. Ennek a felkonfigurálása a Startup.cs fájlban történik meg.

8.3 TypeScript modulok

A TypeScript-ben írt kódbázis szinte teljes egészében az app/core mappa valamelyik almappájában található.

A TypeScript-ből generált, vagy kézzel létrehozott JavaScript kódokat nem egyenként hivatkozom le a HTML oldalon, hanem az ASP.NET Bundle szolgáltatása segítségével, teljes könyvtárakban szereplő összes JavaScript-et belinkelem. Itt találkoztam az első problémába a TypeScript-tel kapcsolatban, ugyanis a Bundle egyenként, abc sorrendben linkeli be a szkript fájlokat, így előfordulhat, hogy egy osztály ősosztálya, még nem töltődött le, és ez problémákat okozhat. A fordító bekonfigurálható úgy is, hogy csak egy fájlba fordítsa az összes JavaScript-et, de én inkább azt választottam, hogy külön

abstracts és concrete alkönyvtárakba szervezem az ős illetve a konkrét osztályokat, és igyekszem minél kevesebb külön fájlt használni.

A TypeScript kódok közül kiemelném a CoreServices modul BaseSignalRService ősosztályát, ami a SignalR hub-okkal történő kommunikációért felelős ősosztály. A Hub kommunikáció mindig aszinkron módon történik. Az osztály egyes esetekben visszahívó függvény (Callback) segítségével jelez vissza a meghívójának. A BaseSignalRPromisedService leszármazottja már JQuery ígéreteket (Promise) használ a visszahívásra, ami egy sokkal elegánsabb felületet ad, és lehetőség van később több ilyen ígéret teljesülésének megvárására, így elkerülve a visszahívási pokol (callback hell) nevű jelenséget. Azért döntöttem a JQuery Promise könyvtára mellett, mert a SignalR is ezt használja, így a Core modul is függ a JQuery-től. Ennek a megoldásnak az a hátulütője, hogy az AngularJs $q Promise könyvtára alapból nem kompatibilis a JQuery ígéretekkel, de a $q.when() függvény segítségével egyszerűen átkonvertálható a JQuery ígérete $q-s Promise-okká. A CoreServices modul osztályainak áttekintő osztály diagramja a melléklet „Fontosabb osztályhierarchiákat leíró diagramok” része alatt megtalálható.

Az Entities modul EntityFactory osztálya képes előállítani egy szervertől kapott egyszerű JavaScript objektumot átalakítani, egy megfelelőbb konkrét típussá. Ehhez az Utils modul DynamycTypeHelper osztályának stringToFunction nevű statikus függvénye segít. A C# entitások TypeName tulajdonságukban közlik a típusukat. A stringToFunction ez alapján a név alapján keres létező típust a window globális változóban. (vagy a this-ben, ha nem böngészőben fut a kód) Amennyiben nincs konkrét egyezés, az EntityFactory még keres egyezést. Például, ha a TypeName tartalmazza a View részszót, akkor a visszaadott típus BaseView típusú lesz.

Megítélésem szerint a TypeScript egy nagyon ígéretes technológia, ami nagymértékben leegyszerűsítheti nagyméretű JavaScript kliensek létrehozását.

8.4 SVG rajzolás

A fejlesztés kezdeti szakaszában döntenem kellett arról, hogy milyen módon kívánom kirajzolni a diagramok elemeit, megvizsgáltam néhány opciót, a CSS3 transzformációkat hamar elvetettem, mert nehezen alkalmazható és nem praktikus a jelen a feladatra. A másik módszer a HTML5 Canvas, sok feladatra alkalmazható, de önmagában ugyancsak nem praktikus a jelen feladathoz. Megvizsgáltam egy Canvas-ra épülő keretrendszert, a Fabric.js-t, ami elsőre impozánsnak tűnt, de maga a keretrendszer rugalmatlan, így bonyolultabb feladatokra (későbbi diagramok) nem lett volna alkalmas.

Végül az SVG mellett döntöttem, amiről tudtam, hogy sok hasonló problémára, már sikeresen alkalmazták.

Létezik egy jó minőségű JavaScript-es SVG alapú diagram könyvtár, a JointJs, de a könyvtár Backbone MVC-t használ és az AngularJs integrálása túl mutatott volna a jelen dolgozat határain.

Végül úgy döntöttem, hogy AngularJs direktívákat fogok készíteni.

8.5 AngularJs

A fejlesztést az AngularJs megismerésével folytattam, ugyanis a keretrendszer nagyban befolyásolhatja, hogy hol és milyen módon helyezkedik el az alkalmazás forráskódja. A keretrendszert az alkalmazás számos pontján használtam. Itt csak a jelentősebb, vagy kevésbé ismert felhasználásokra térek ki.

8.5.1 Diagram elemek direktívái

Az AngularJS képes kétirányú adatkötést biztosít, a HTML oldal elemei és az elkészített kontrollerek/direktíva között. A kötés a $scope, kontextus objektumon keresztül zajlik. Ez az adatkötés SVG elemeknél is működik. Például az alábbi kód az svg elem x attribútumát összeköti a scope.diagramElement.X tulajdonsággal.

<svg ng-attr-x="{{diagramElement.X}}"> …

A sikeres adatkötés után az elemből újrafelhasználható direktívát szerettem volna készíteni. El is készítettem a useCaseDir.js-t és a hozzá csatolt useCaseDir.svg-t

A direktíva a modelling-page.html oldalon.

<use-case diagram-element="part"> </use-case>

A megjelenés így viszont nem történt meg. Két probléma gátolta a megjelenést, az első, hogy be kellett állítani a direktívában, hogy SVG alapú sablont használok (templateNamespace: "svg"). A másik probléma az volt, hogy a direktívák úgy működnek, hogy meghagyják a DOM-ban komponens elemét és a belsejébe generálják a sablont, így jelen esetben az SVG sablon belekerült a usecase nyitó és záró címke közé, ez a HTML értelmezőknek nem okoz gondot, de ezt a saját címkét egy rajzlapnak használt SVG címke foglalta magában, és ez az értelmező nem fogadja el a számára ismeretlen címkéket. Erre a problémára megoldást adott, hogy beállítottam a direktívában, hogy cserélje le az elemet a tartalmával (replace: true)

Az elem fogd és vidd mozgatására a D3.js könyvtárat használtam. Ami egy külön lekérdező rendszerrel működik, ezért az AngularJs által visszaadott JQuery objektumot át kellett transzformálni, ami szerencsére egy egyszerű tömbbé alakítással megoldható volt. Ez látható a BaseView osztályban: d3.selectAll(domElement.toArray()).call(drag);

8.5.2 Késleltetett AngularJs betöltés

Az AngularJs, amennyiben a HTML tartalmazza bármelyik elemén az ng-app attribútum-direktívát, rögtön elindul. Ezt el kellett halasztani addig, amíg a SignalR kapcsolat fel nem épül.

A megoldás az volt, hogy a HTML-ből kitöröltem az ng-app attribútumot, és az app.js fájlban, miután a BaseSignalRService visszajelez, hogy elkészült a kapcsolat, meghívódik az initAngular függvény, mely a DOM betöltődése után meghívja az angular.bootstrap függvényt, ami elindítja az AngularJs további folyamatait.

9 AZ ELKÉSZÜLT RENDSZER BEMUTATÁSA

Az itt bemutatott aloldalakhoz képernyő képeket is készítettem, melyek a melléklet „Képernyő képek” részében találhatóak.

9.1 A projektek aloldal

A projektek aloldalon keresztül lehet a projektekkel kapcsolatos funkciókat ellátni. Az aloldal funkcionalitásának megteremtésében a web projekt app/features/projects mappában található HTML fájl és JavaScript kódok felelnek. A projektek megjelenítésére az app/widgets mappában található entity-jumbotron direktíva felel, amit újrafelhasználok a diagramok aloldalon is.

A diagramok aloldal a projektek aloldalhoz nagymértékben hasonlít, így nem térek ki rá külön alfejezetben, a mellékletek „Képernyő képek” fejezetében látható, a diagramok oldalról készült képen egy olyan dialógus ablak látható, melyet a rendszerben számos ponton használok. Amint a képen is látható, az ablak végzi el a neki átadott elem kliens oldali validációját.

9.2 A követelmények aloldal

A követelmények aloldalon keresztül lehet a követelményekkel kapcsolatos funkciókat ellátni. A követelmények felvitelére a rendszer sztori módszerét támogatja. A Mellékletek rész Követelmény formátum sablonok között megtalálható a User Story sablon. A sablon által kijelölt mezőkön túl az általam kialakított sablon, tartalmaz egy név és egy leírás mezőt, illetve a követelmény analizálásánál felhasználható fontosság és komplexitás mezőket.

Egy-egy követelmény felvételét, módosítását és törlését az app/features/requirements mappa szkriptjei végzik. A userstory-jumbotron direktíva a CRUD műveletek koordinálásának a felelősségét részben átveszi a kontrollertől.

9.3 A modellezés aloldal

A modellezés aloldalon keresztül lehet a megnyitott diagramhoz elemeket felvenni. Az bal oldali eszköztár segítségével lehet a tőle jobbra elhelyezkedő rajzoló felületre felvenni szereplőket, használati eseteket és asszociációs vonalakat. A rendszer felajánlja a már létező aktorok és használati esetek a diagramra való felvételét, de módot ad új elemek létrehozására is.

A feladat végrehajtásában számos modul részt vesz és maga a feladat is kiterjedt. Amennyiben új elemet is fel kell vennünk, akkor három aszinkron hívást kell sorosan egymás után végrehajtanunk, mielőtt a diagramra helyezhetnénk az új diagramelemet.

Először felnyitunk egy dialógus ablakot, mely csak a felhasználótól érkezett megerősítő gomb esemény után oldja fel az ígéretet. Ez után, elküldjük a szervernek az újonnan létrehozott, elemet. Majd ha a szerver ezzel végzett, egy újabb kérést küldünk a szervernek, ami már az adott diagramrész elemet veszi fel, mely hivatkozik az előzőleg felvett elemre. Miután erről is érkezett visszajelzés a szervertől, felvehetjük az új diagramelemet a diagramra.

A másik bonyolult részfeladat a diagramelem megjelenítése a diagramon. Magát a megjelenést az elemhez tartozó direktíva tartalmazza, a megjelenés logikáját pedig a hozzá tartozó BaseView típus adja. Például a UseCase elem megjelenését a use-case direktíva adja. A megjelenítéssel kapcsolatos felhasználói eseményfigyelésének, és a megjelenített oldal adatatárolásána a UseCaseView a felelőse. Mivel a felhasználói eseményekre való reagálás itt nem az AngularJs felügyeletével zajlik, az események bekövetkeztekor a scope.$evalAsync() függvénnyel kiváltom azt, hogy az AngularJs frissítse a direktívákat, ezzel frissítve a megjelenő képet is.

A kapcsolatok szerverre történő szinkronizálását nem technikai okok miatt nem oldottam meg. Ugyanis a szerver oldalon a kapcsolatokat tároló osztály absztrakt ős típusokat használ navigációs tulajdonságnak, ami se a Json.NET keretrendszer se az Entity Framework nem kezel megfelelően. A probléma megoldása nem oldható meg a rendszer alapos újratervezése nélkül. A kliens oldalon a kapcsolatok ábrázolása úgy történik, hogy a kapcsolat a két diagram elem középpontját köti össze. A kapcsolatokat rajzolom ki először, és ilyenkor az összes többi elem fölébük kerül, így összetett objektumoknál is esztétikus az asszociációs egyenes megjelenése, hátránya viszont, hogy a további kapcsolati nyíl típusok bevezethetőségére nem ad megoldást

9.4 Az elkészült rendszer elérése

A rendszer forráskódja elérhető a DVD mellékleten, illetve a github forrásmegosztó honlapon keresztül a https://github.com/lovi88/SoftwareRequirementsTool címen.

A működő projekt elérhető a Microsoft Azure felhőben a http://softwarerequirements.azurewebsites.net címen.

10 TESZTELÉS

10.1 Kézzel végzett tesztelés

A rendszer egy JavaScript vastagkliens alkalmazás. Így mindenképpen célszerű a kezelőfelületen keresztül tesztelni. Ráadásul egyes tesztesetek egyszerre több böngésző együttes figyelését igénylik, ezért a rendszer manuális tesztelése mellett döntöttem.

10.1.1 Hibák súlyossága szerint kialakított rendszer

- Blokkoló hiba
  - Olyan hiba, amire más funkciók is épülnek, így a hiba több helyen is kihat, több funkció működését, illetve tesztelését is gátolja.
  - Mielőbbi beavatkozást igényel.
- Kritikus hiba
  - Fontos funkció működését csorbító hiba.
  - A hiba minél előbb, lehetőleg a következő verzióra történő kijavítása.
- Normál hiba
  - A hiba egy mellékes funkcióban következik be, vagy a felhasználói élményt nagymértékben rontja.
  - Kijavítása lehetőleg néhány verzión belül történjen meg.
- Elhanyagolható hiba:
  - Olyan hiba, ami a felhasználói élményt csak kis mértékben befolyásolja. Ide értendők például az esztétikai hibák.
  - Megoldása esetlegesen egy későbbi változatban megtörténhet.

10.1.2 Egy-böngészős tesztesetek:

Teszteset neve	Bemenő adat / Tevékenység	Elvárt viselkedés	Eredmény	Hiba súlya
Új Projekt készítése	Helyes adat	Sikeres felvétel	Siker	___
Új Projekt készítése	Üresen hagyott név mező	Felugró figyelmeztető buborék	Siker	___
Új Projekt készítése	Csak fehér karaktereket tartalmazó név mező	Felugró figyelmeztető buborék	Kudarc (de más hibajelzés van)	Elhanya-golható hiba
Projekt megnyitása	Kattintás egy projekt open gombján	Megjelenő Requirements és Diagrams menüpontok	Siker	___
Törölhető projekt törlése	Egy olyan projekt törlése, aminek	A projekt eltűnik, és oldalújratöltéskor sem jelenik meg újra	Siker	___

	nincsenek elemei			
Nem törölhető projekt törlése	Egy olyan projekt törlése, melynek vannak elemei	A projekt nem tűnik el és a rendszer esztétikus hibavisszajelzést ad	Kudarc (a rendszer a törlést nem engedi, de a hibajelzés nem megfelelő)	Normál hiba

12. ábra Egy-böngészős tesztesetek - táblázat

10.1.3 Több-böngészős tesztesetek

1. Két böngésző aktív, különböző projekt van megnyitva

Új diagram felvétele	Új diagram felvétele az egyik böngészőben	A másik böngészőben nincs változás	Sikeres	___
Új követelmény felvétele	Új követelmény felvétele az egyik böngészőben	A másik böngészőben nincs változás	Sikeres	___

13. ábra: Több-böngészős tesztesetek 1. - táblázat

2. Két böngésző aktív, ugyan az a projekt van megnyitva

Új diagram felvétele	Új diagram felvétele az egyik böngészőben	A másik böngészőben is megjelenik a változás	Sikeres	___
Új követelmény felvétele	Új követelmény felvétele az egyik böngészőben	A másik böngészőben is megjelenik a változás	Sikeres	___

14. ábra: Több-böngészős tesztesetek 2. - táblázat

3. Modellezés, a két böngészőben ugyan az a diagram van megnyitva

Új használati eset felvétele	Új használati eset felvétele	Mindkét böngészőben megjelenik az új elem	Sikeres	__
Használati eset mozgatása a modellezési területen	Használati eset mozgatása	Mindkét böngészőben hasonlóan mozog az alakzat	A másik böngészőben akadozó a mozgás	Normál hiba
Asszociációs kapcsolat felvétele két diagram elem között	Két használati eset összekötése	Mindkét böngészőben megjelenik az új elem	Csak a kiinduló böngészőben jelenik meg az alakzat	Kritikus hiba

15. ábra: Több-böngészős tesztesetek 3. - táblázat

10.2 A tesztelés kimenete

10.2.1 Asszociációs kapcsolat felvétele két diagram elem között

A diagramok közti asszociációs kapcsolat nem menthető. A hiba azért áll fent, mert a szerver oldalt nem hívjuk meg. Ez csak egy fix arra nézve, hogy az Entity Framework nem tudta a kapcsolat (Entities.Connections.Connection) absztrakt navigációs tulajdonságait feloldani.

Hiba mértéke: A hiba funkcionalitást csökkentő, kritikus hiba.

10.2.2 Használati eset mozgatása a modellezési területen

A hiba azért áll fent, mert minden második változásnál szólunk a szervernek, és az menti a változásokat az adatbázisba is. Vizsgálati céllal kiiktattam az adatbázisba való mentést, és a böngészők közti szinkron folyamatos lett.

Hiba mértéke: A hiba felhasználói élményt csökkentő, normál hiba.

Megoldási javaslat: Az elemek mozgatásakor csak elemfrissítési üzenet menjen a többi böngészőhöz, és az elem lerakásakor történjen tényleges adatbázismentés.

10.2.3 Nem törölhető projekt törlése

A hiba bekövetkeztekor, egy kivétel szövegét jelenítjük meg egy felugró ablakban, majd újratöltjük az oldalt, mert a kliens már előre törölte az elemet.

Hiba mértéke: A hiba felhasználói élményt csökkentő, normál hiba.

Megoldási javaslat: A törlési rész átalakítása, a kliens várja meg a szerver oldal visszajelzését a törlés sikerességéről, a szerver pedig ne küldjön ki kivétel szöveget.

11 TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

11.1 A rendszer technikai továbbfejlesztése

- A rendszer felépítése, a SignalR által használt Json.NET és az Entity Framework megkötései miatt, sokat veszített a rugalmasságából. A rendszerfelépítés megváltoztatása vagy a keretrendszerek hátrányainak kiküszöbölése javasolt a további funkciók beépítésének megteremtése érdekében.
- JointJs diagram keretrendszer integrálása.

11.2 További rendszerfunkciók kifejlesztése

Követelmény feltáráshoz kapcsolódó lehetősségek:

- A megrendelők által a webes felületen kiküldhető űrlapok biztosítása a felhasználói vélemények összegyűjtése.
- Új követelmény típusok támogatása
- Új követelmény sablonok támogatása, és lehetőség biztosítása a sablonok személyre szabására.
- Követelmények közti kapcsolatok felvételek biztosítása.
- Követelmények integritásának, ellentmondás mentességének a biztosítása.
- A követelmény szövegek formázásának biztosítása.

Modellezéssel kapcsolatos lehetősségek:

- Új diagram típusok támogatása
- Követelmények alapján történő diagramgenerálás
    - Szövegbányászati eszközökkel történő információ kinyerés.

Nemfunkcionális javítási lehetősségek:

- A kezelő felület egységesítése, ergonómiai javítása.

Egyéb lehetősségek:

- A tesztelés fejezetben megtalált programhibák javítása
- Az az oldal reszponzív megjelenésének további javítása.
    - Az aloldalak nem minden felbontásban jelennek meg megfelelően.
- Felhasználó kezelés támogatása
- A diagramok és a kapcsolódó követelmények közti navigáció.
- A projekt elemei közti kapcsolatok vizualizálása.

12 ÖSSZEFOGLALÓ

A dolgozatom célja, hogy egy olyan eszközt készítsek el, mely támogatja a követelménytervezés egyes magas absztrakciós tevékenységeit, és ezt úgy teszi, hogy a rendszert használók a lehető legkülönbözőbb platformokról tudjanak az eszközzel közösen dolgozni.

A dolgozatom első felében a szakirodalom alapján bemutattam a követelménytervezés hagyományos és újabb módszereit.

Megvizsgáltam a szakirodalom alapján, hogy milyen módszerekkel lehet a követelmények szöveges reprezentációjából, információt kinyerni, és azt milyen módokon lehet felhasználni a követelmények modell reprezentációjának előállítására.

Megvizsgáltam két a piacon lévő modellezéssel foglalkozó terméket, és felvázoltam, hogy ezekhez képest a dolgozat keretei közt elkészülő szoftver milyen főbb eltéréseket mutat.

Feltártam és specifikáltam az eszközzel szemben támasztott főbb funkcionális és nemfunkcionális követelményeket. Bemutattam továbbá a rendszer határait.

Megvizsgáltam, hogy mely technológiák a leginkább alkalmasak a jelenlegi szoftverfejlesztési projekt elkészítésére, majd e technológiák figyelembe vételével megterveztem a rendszert.

Bemutattam az implementáció fontosabb szakaszait, illetve azt, hogy milyen nagyobb kihívásokra milyen megoldást adtam.

A rendszer tesztelése és a tesztelési eredmények kiértékelése után felvázoltam néhány továbbfejlesztési lehetőségét a rendszernek

13 SUMMARY

The main goal of my thesis is to create a tool which can be used in the higher abstractions of the requirements design process. In addition to that the tool must be accessible from most of the platforms and it must support collaborative team work.

In the first part of my thesis I showed the main concepts of the requirements design and its more and less traditional approaches.

After that I have examined the ways for information retrieval from the textual representation of the requirements and using that information to generate the model representation of the requirements.

I have created the review of two existing products of the market which are for modelling and I drew up the main differences from the software of my thesis.

I have explored and specified the main functional and non-functional requirements of the tool. And I have shown the boundaries of the system.

I have examined that what are the best technologies to use in my software development project and I designed the system architecture considering of these technologies.

I have shown the main phases of the implementation process and my solutions for the main problems during the development.

After testing the system and evaluating the test results I have sketched some ways of the further development of the system.

14 IRODALOMJEGYZÉK

14.1 Felhasznált könyvek

[1] Maksimchmuk, R. A., Naiburg, E. J.: UML földi halandóknak *Kiskapu Kft.*,2006

[2] Martin, R. C.: Tiszta kód. *Kiskapu Kft.*, 2010

[3] Sommerville, I.: Szoftverrendszerek fejlesztése. *Panem Kiadó Kft.*, 2007

[4] Tikk D.: Szövegbányászat. *Typotex Kiadó Kft.*, 2007

[5] Vég, Cs.: Alkalmazásfejlesztés a Unified Modelling Language szabványos jelöléseivel. *Logos 2000 Bt.*, 1999

14.2 Felhasznált honlapok

[6] Ambler, S. W.: UML 2 Use Case Diagrams: An Agile Introduction (http://www.agilemodeling.com/artifacts/useCaseDiagram.htm),
utoljára megtekintve: 2015-05-13.

[7] Cohn, M.: Advantages of User Stories for Requirements (http://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requirements), utoljára megtekintve: 2015-05-13.

[8] Due, R. T.: Abbot Textual Analysis

(http://www.informit.com/articles/article.aspx?p=29043&seqNum=9),
utoljára megtekintve: 2015-05-13.

[9] Ponomareff, D.: Agile stories, estimating and planning

(http://www.slideshare.net/dimka5/agile-stories-estimating-and-planning),
utoljára megtekintve: 2015-05-13.

[10] Stellman, A.: Requirements 101: User Stories vs. Use Cases

(http://www.stellman-greene.com/2009/05/03/requirements-101-user-stories-vs-use-cases/),
utoljára megtekintve: 2015-05-13.

MELLÉKLETEK

Követelmény formátum sablonok

A sablonok leírásánál kapcsos zárójelek közé helyeztem a kötelezően kitöltendő részeket, és szögletes zárójelek közé helyeztem az opcionális részeket.

Felhasználói követelmény sablon:

{SORSZÁM (pl:2.3.1)} {Követelmény neve - nagyobb és félkövér betűformátum}.

{A követelmény rövid leírása, a lényeges részek félkövérrel kiemelve}

[Magyarázat: - tömör magyarázata az adott funkciónak, dőlt betűs szedésben]

[Szerző: - a követelmény készítőjének neve, elérhetősége]

User Story sablon:

A felhasználói sztorik leírása alapvetően lehet kötetlen, de napjainkban elterjedt sablonja az:

Én, mint {**felhasználói szerepkör}** képes vagyok [**egy tevékenységre]**, mert ezzel elérhetem az [**üzleti értéket képviselő célomat]**

Ugyanez angolul:

As a **{role}** I can [**activity/ feature]** so that **[benefit /business value]**

Ez a sablon azért olyan közkedvelt, mert külön részt szentel a „Miért” kérdésnek, így sokkal tisztább és érthetőbb a fejlesztésben résztvevő különböző személyeknek az adott sztori fontossága. Komplexebb több felhasználói szerepkört érintő leírásánál ajánlott áttérni az egyes szám harmadik személyre.

Rendszerkövetelmény sablon

{A szoftver neve félkövérrel szedve}

Funkció	A funkció neve / néhány szavas leírása
Leírás	A funkció néhány mondatos leírása
Bemenet	A rendszerfunkcióhoz szükséges bemenő adatok
Forrás	A be és a származási helyük leírása.
Kimenet	A szolgáltatás eredménye
Cél	A szolgáltatás célja
Művelet	A végrehajtandó műveletek leírása
Előfeltétel	A funkció meghívásának előfeltétele
Utófeltétel	A funkció eredményeként előállt utófeltételek
Mellékhatás	A funkció környezetére gyakorolt hatása

16. ábra: Rendszerkövetelmény sablon - táblázat

Formális használati eset szcenárió sablon.

{A folyamatleírások lépéseinek listába szedett szöveges leírása}

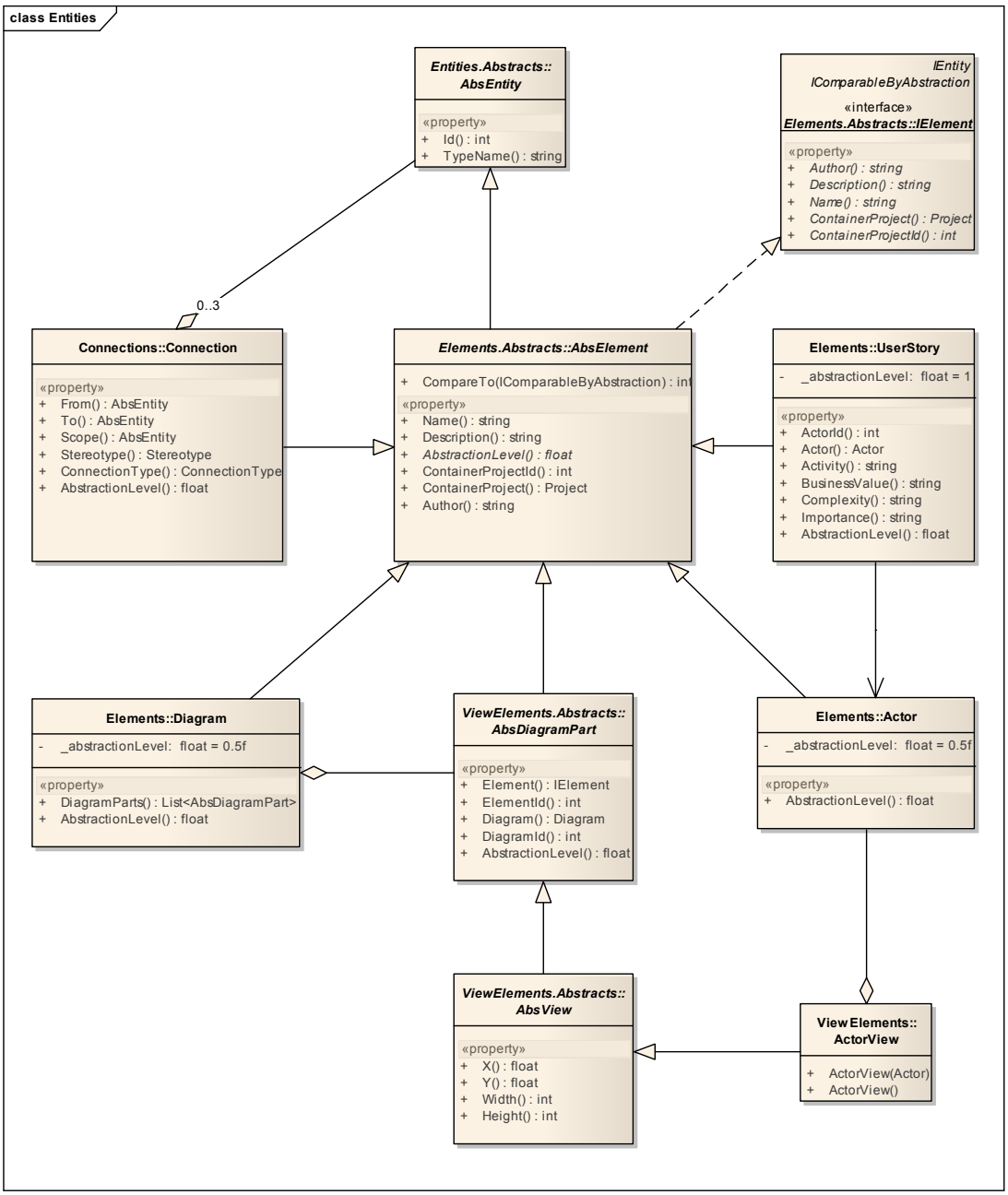
Name	Az adott használati eset neve.
Summary	Egy rövid összefoglaló a használati eset céljáról.
Rationale	A használati eset és a használatának a folyamat egyes körülményeinek kifejtése.
Users	A használati esetben résztvevő szereplők felsorolása
Preconditions	A használati eset előfeltételei
Basic Course of Events	A folyamat helyes ágának leírása, számozott lépéspontokba szedve.
Alternative Paths	Az egyes kibővítési pontokhoz tartozó folyamatirányok kifejtése.
Postconditions	A folyamat végső kilépési feltételei.

17. ábra: Formális használati eset szcenárió sablon

Fontosabb osztályhierarchiákat leíró diagramok

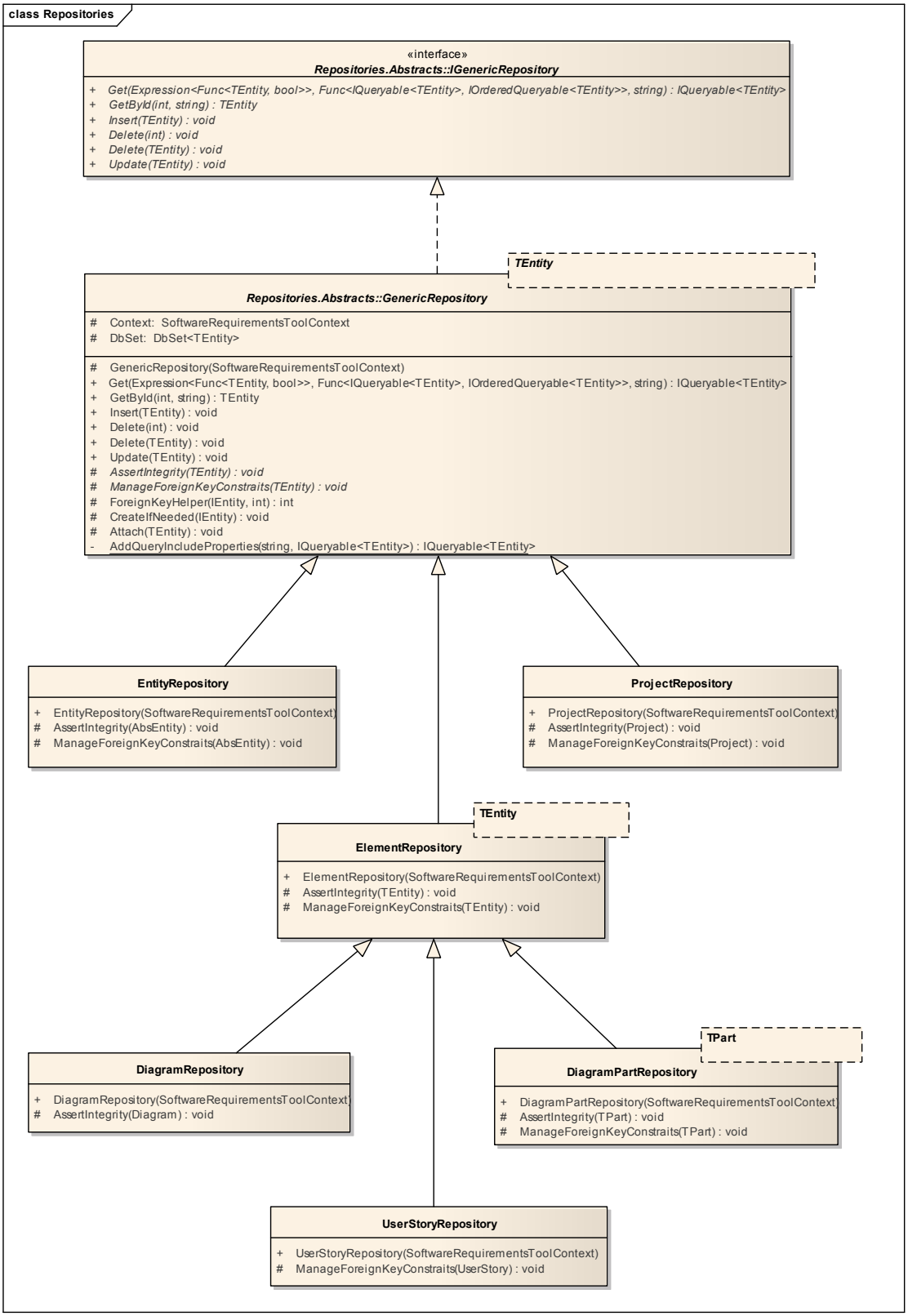
További diagramok tekinthetőek meg a DVD melléklet egyéb könyvtárában lévő K971UM_Lovas_Tick.eap nevű fájlban. A fájl megnyitásához Enterprise Architect program szükséges.

Entitás modell



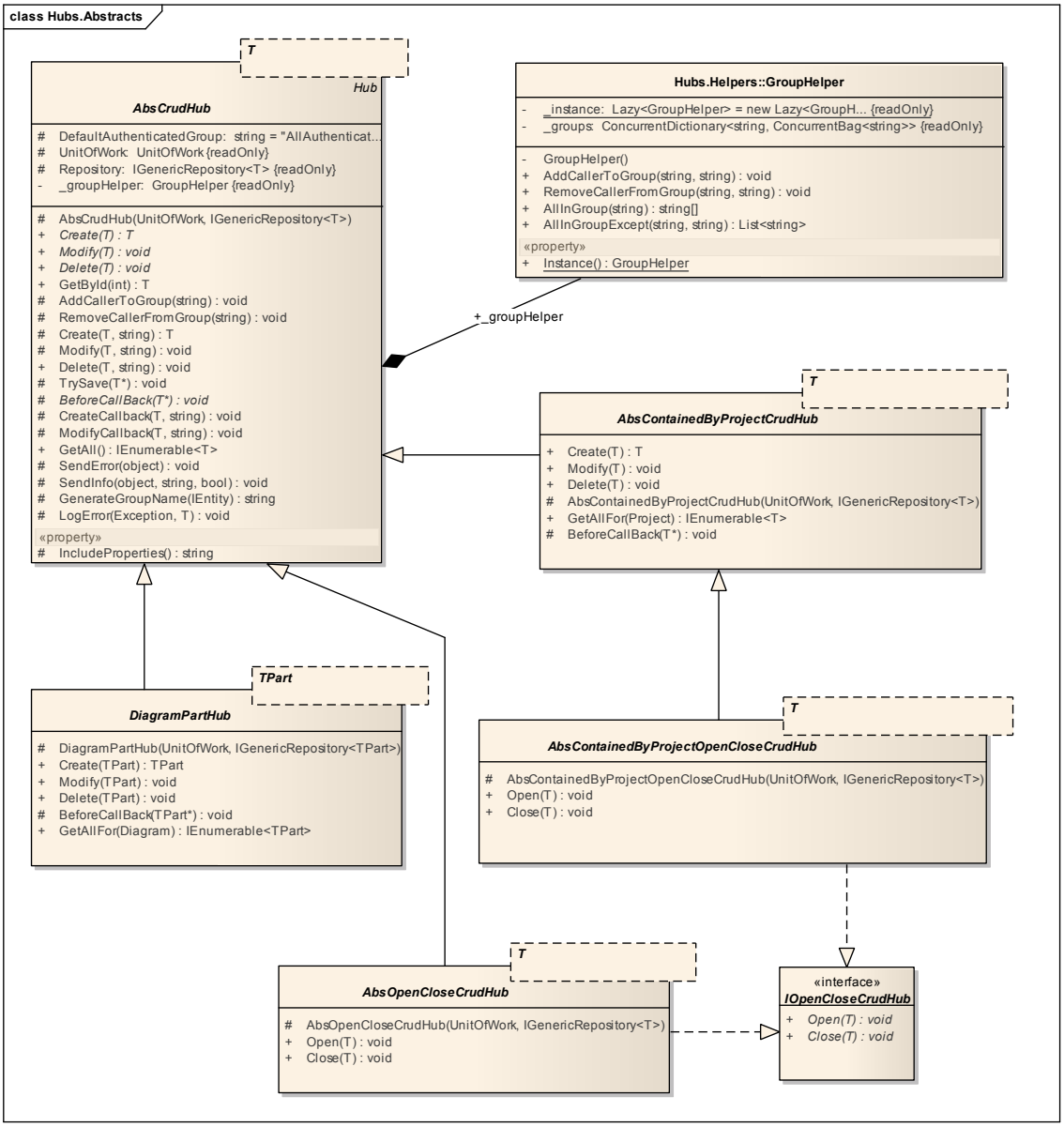
18. ábra: Entitás modell - osztálydiagram

A generikus raktár (repository) és leszármazotti hierarchiája



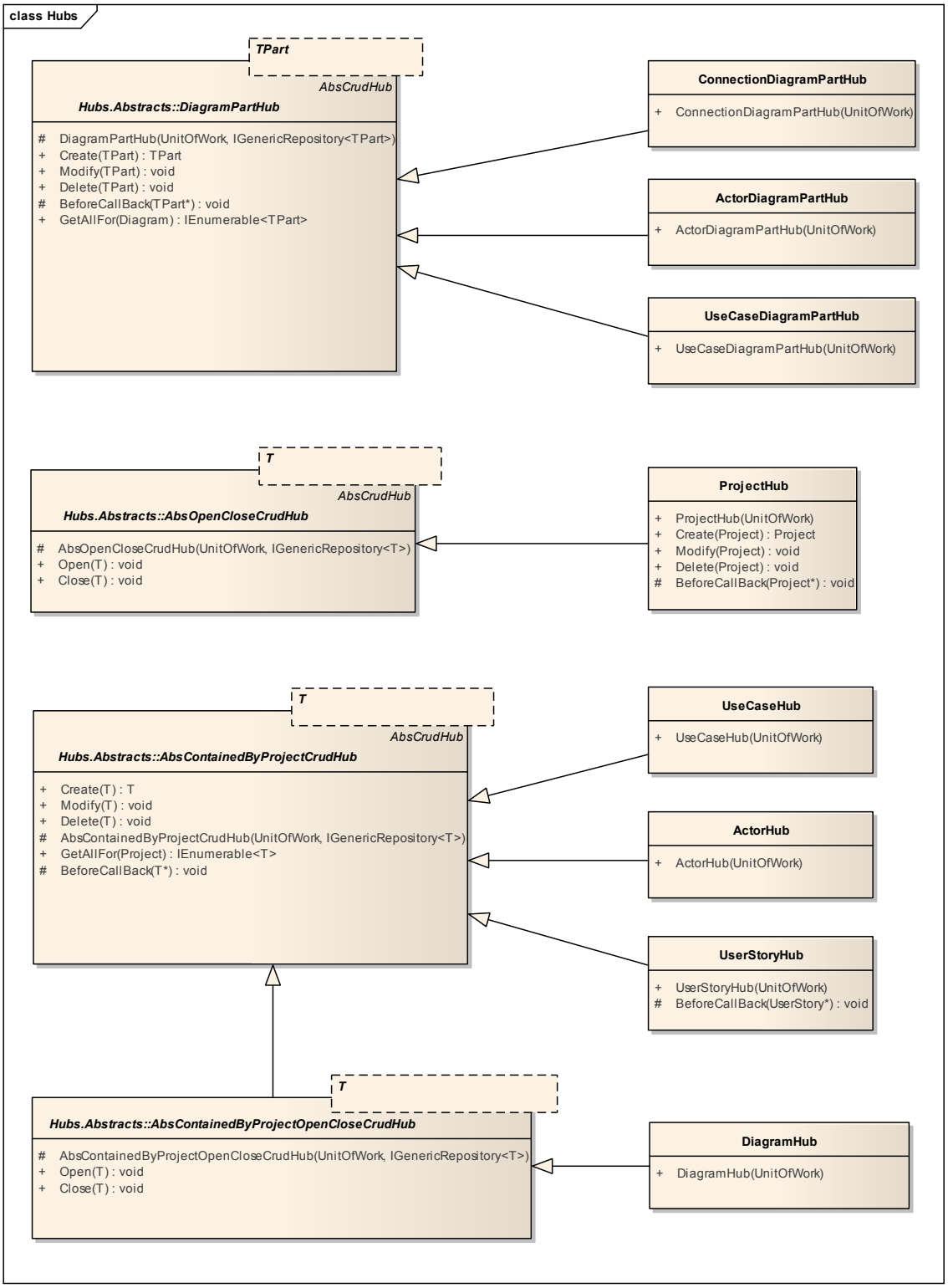
19. ábra: Generikus raktár (Repository) - osztálydiagram

A külső hozzáférési felületet nyújtó Hub osztályhierarchia: felületek



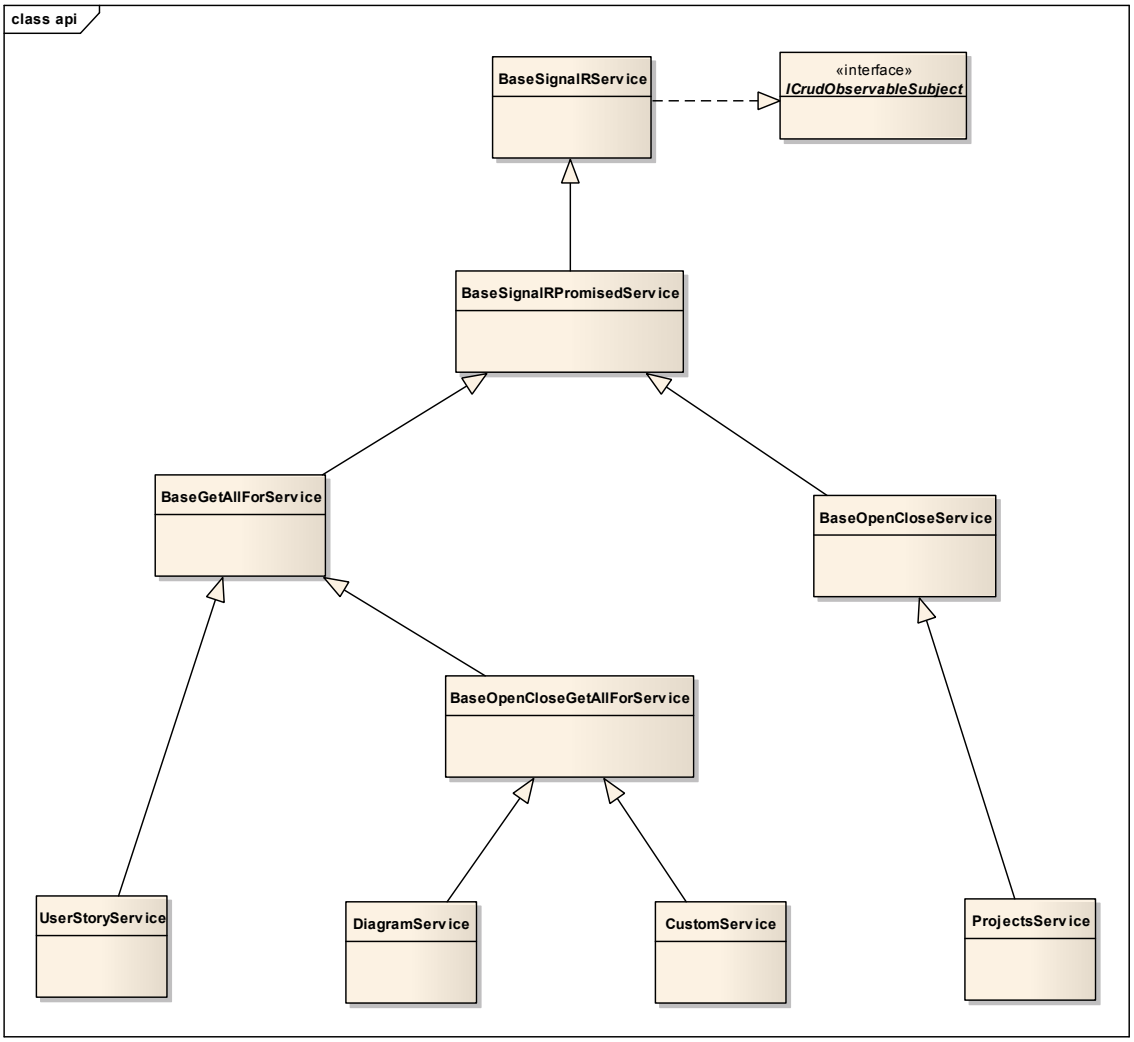
20. ábra: Hub-felületek kapcsolata - osztálydiagram

A konkrét Hub megvalósítások által alkotott API - osztályhierarchia:



21. ábra: Konkrét Hub-ok - osztálydiagram

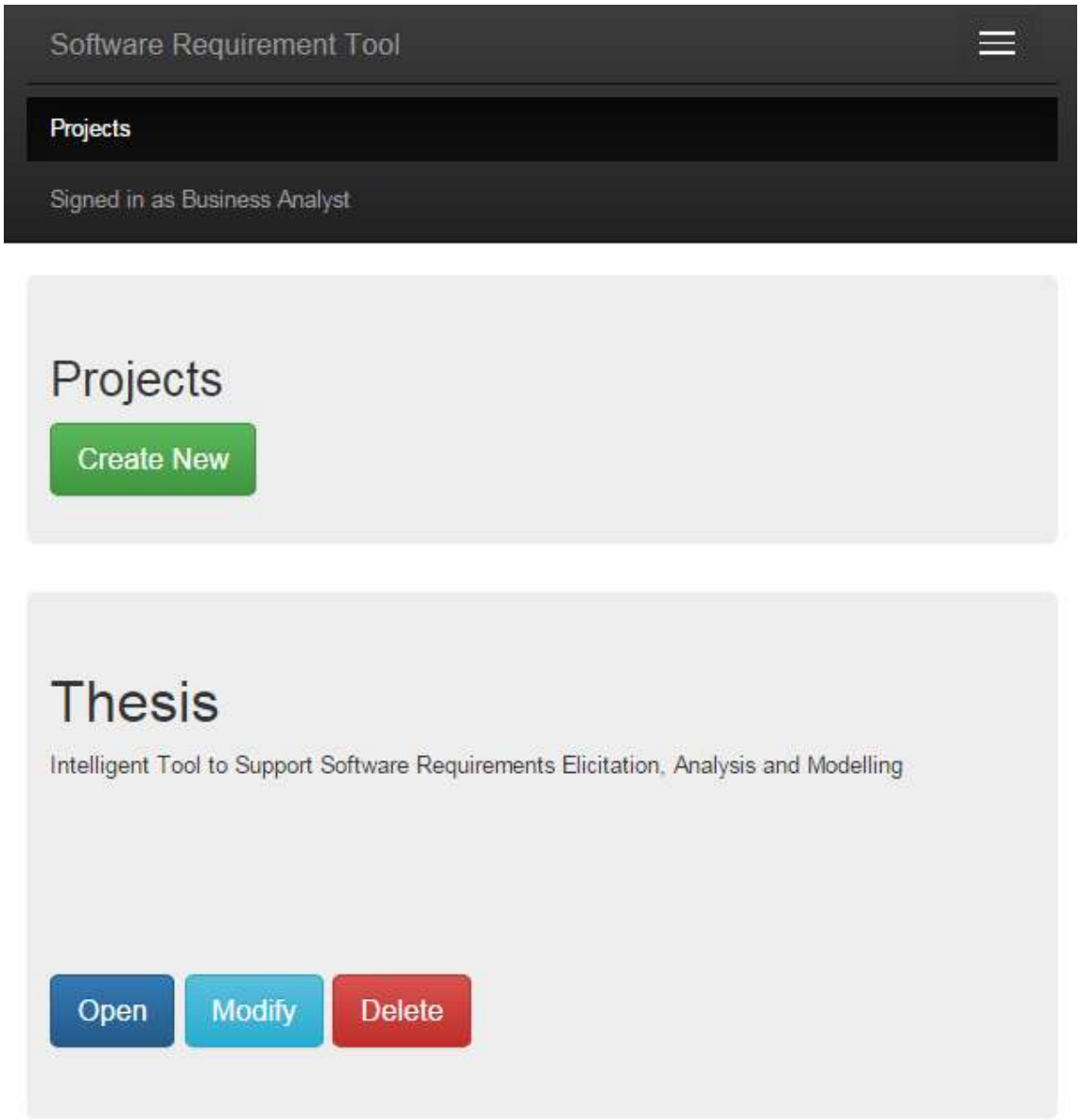
A kliens oldali CoreServices magas szintű osztálydiagramja



22. ábra: Kliens oldali CoreServices modul - magas szintű osztálydiagram

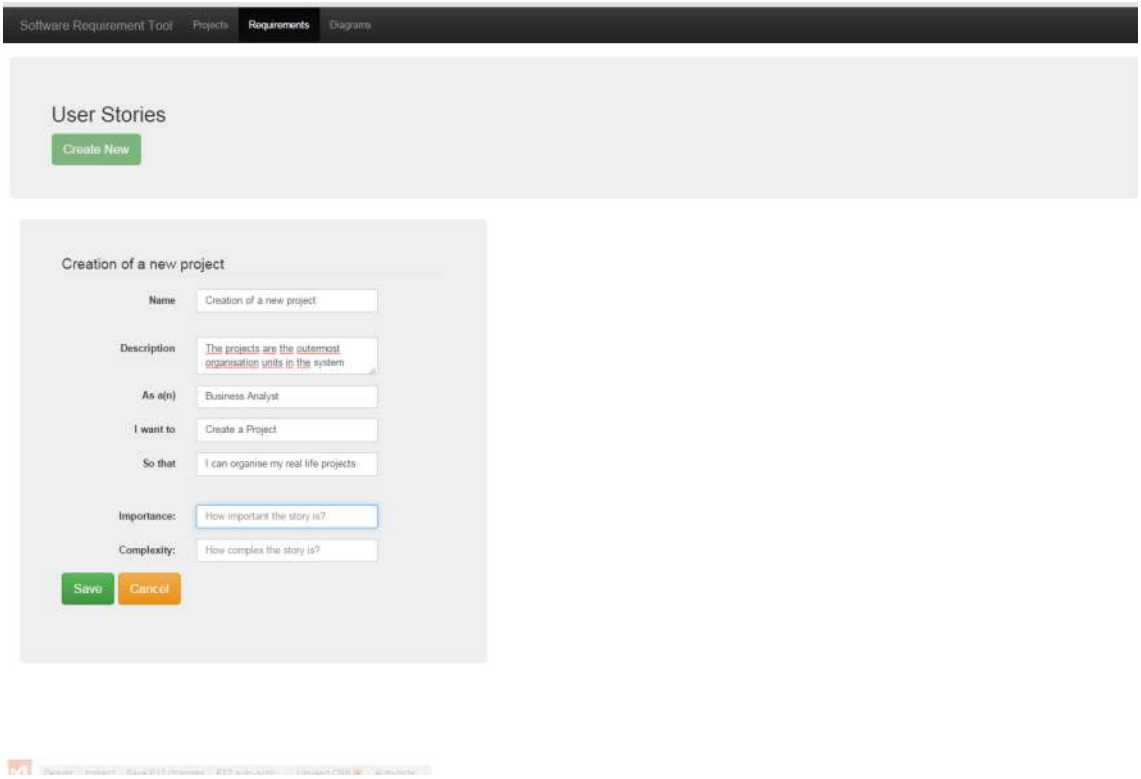
Képernyő képek

A projektkezeléshez tartozó aloldal

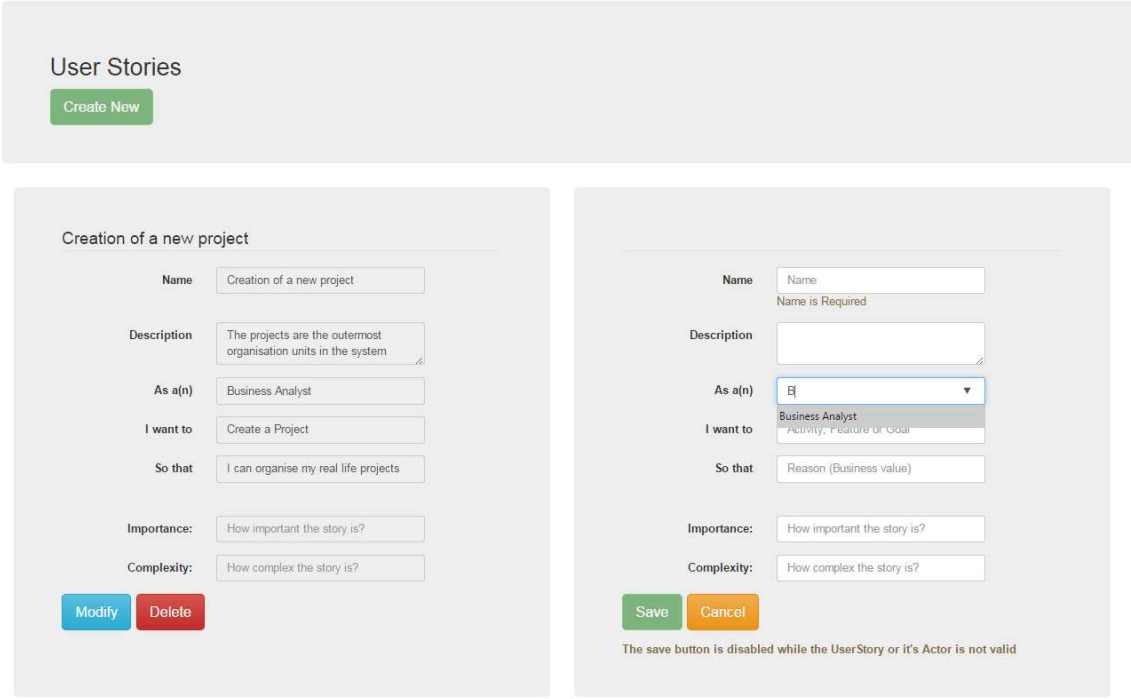


23. ábra: Projektkezeléshez tartozó aloldal - diagram

Követelmény feltárás aloldal

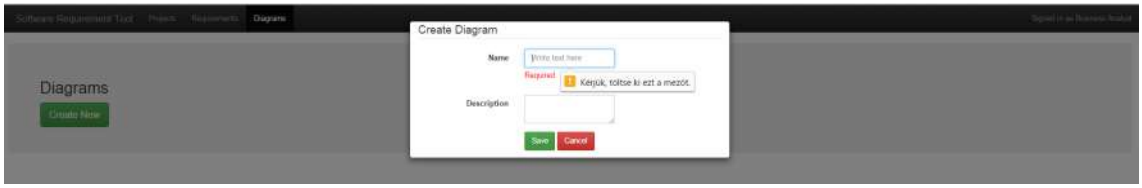


24. ábra: Felhasználói sztori felvétele - ábra



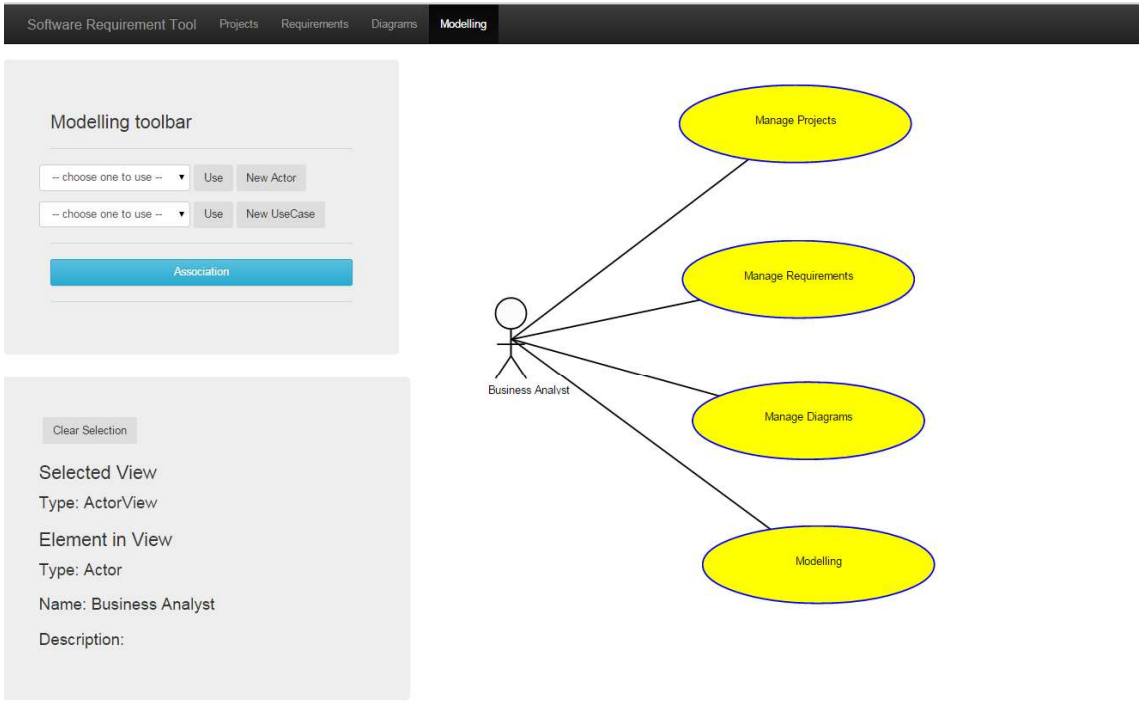
25. ábra: Felhasználói sztori, szereplő kiegészítés- ábra

Diagramok kezelése almenü



26. ábra: Diagram felvétele ablak, hibajelzés - ábra

Modellezés aloldal



27. ábra: Modellezés aloldal - használati eset diagram