XI. Erdélyi Tudományos Diákköri Konferencia

Kolozsvár, 2008. május 23-24.

UML diagramok összehasonlítása és összefésülése

Készítette:

Császár Előd

Babes-Bolyai Tudományegyetem
Matematika-Informatika kar
Informatika szak
3. évfolyam

Témavezető:

Dr. Darvay Zsolt egyetemi adjunktus

Babes-Bolyai Tudományegyetem
Matematika-Informatika kar
Programozási nyelvek és módszerek tanszék

Tartalomjegyzék

1. Bevezető

A dolgozatban a *UML Merge Tool* elnevezésű, UML modelleket megjelenítő, összehasonlító és összefésülő alkalmazást mutatom be. Ez az alkalmazás azért készült, mert jelenleg nagyon kevés olyan program létezik a piacon, amelyekkel UML modelleket lehet összehasonlítani és összefésülni. A létező programok nem rendelkeznek néhány olyan fontos tulajdonsággal és funkcionalitással, amelyek szükségesek az ipari alkalmazáshoz.

A legtöbb UML szerkesztő program a modelleket saját formátumú szöveges fájlba menti el. Jelenleg a *UML Merge Tool* az IBM Rational Rose modell fájlok kezelését teszi lehetővé, de tervezése és elkészítése úgy történt, hogy a lehető legegyszerűbben lehessen bővíteni más típusú fájlok kezelésére.

Fontos szempont, hogy a program ne legyen rendszerfüggő, ezért a Java programozási nyelvben készült.

Köszönöm Darvay Zsolt tanár úrnak és az evoline munkatársainak a dolgozatomhoz nyújtott segítségüket.

2. Elméleti háttér és gyakorlati megvalósítás

2.1. Az UML

Az Unified Modeling Language, azaz röviden **UML** az objektumorientált programozás szabványos specifikációs nyelve, amely grafikus jelöléseket használ rendszerek absztrakt modelljének leírására. A szabvány UML jelölést használó modelleket UML modelleknek nevezzük.

Egy UML modell több diagramból tevődik össze, amelyek két nagy csoportba oszthatóak: strukturális és viselkedési diagramok.

A strukturális vagy statikus diagramok a modellezett rendszer elemeire vonatkoznak. Altípusai: osztálydiagramok, komponensdiagramok, összetett struktúradiagramok, telepítési diagramok, objektumdiagramok és csomagdiagramok.

A viselkedési vagy dinamikus diagramok azt írják le, hogy minek kell történnie a modellezett rendszerben. Altípusai: aktivításdiagramok, állapotgép diagramok, felhasználói eset diagramok és interakciós diakramok. Az interakciós diagramok fogalmazzák meg a rendszerelemek közötti kommunikációt és ezen altípusokkal rendelkezik: kommunikációs diagramok, áttekintő diagramok, szekvenciadiagramok és időzítődiagramok.

2.2. Az UML Merge Tool születése és rövid bemutatása

A szoftverfejlesztés folyamatának egyik jelentős része a verziókövető rendszer használata. Ezek a rendszerek teszik lehetővé, hogy több fejlesztő dolgozzon ugyanazon a forrásanyagon akár egyidejűleg is, és ezek teszik lehetővé a szoftver régebbi verzióinak elérését. A fejlesztés során szükséges különböző verziók összehasonlítása és bizonyos verziók változásainak összefésülése egy már létező verzióval. A forráskódok könnyen összehasonlíthatóak különböző szöveges összehasonlító alkalmazásokkal, de az UML modellek egy eltérő, sokkal jellegzetesebb értelmezést és kezelést követelnek meg.

Az evoline cég munkatársainak körében, ipari tapasztalataikra alapozva, felmerült a

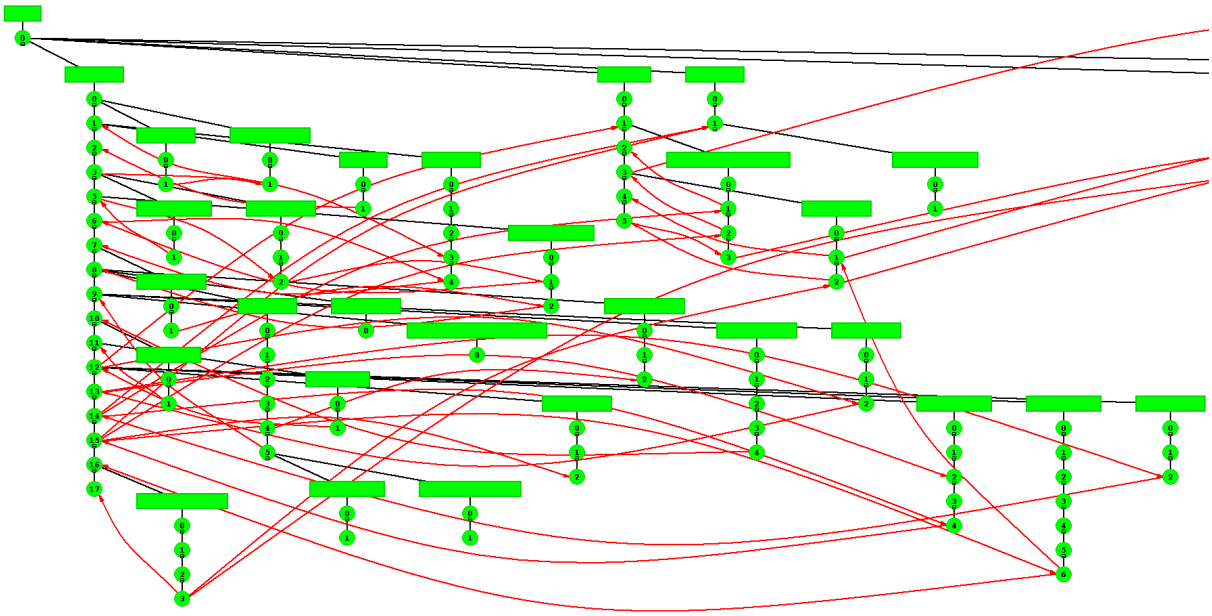
szükségessége egy olyan alkalmazásnak, mely teljesíti a következő követelményeket:

1. Legyen képes grafikusan megjeleníteni egy modellt.
2. Legyen képes összehasonlítani ugyanazon modell két különböző verzióját.
3. Automatikusan fésülje össze azon változtatásokat, melyek nem állnak konfliktusban és tegye lehetővé, hogy manuálisan döntsünk konfliktus esetén illetve azt, hogy megváltoztassuk az automatikus összefésülés eredményét.
4. Tegye lehetővé szűrők létrehozását. A kiszűrt változtatásokat ne vegye figyelembe az alkalmazás.
5. Az összefésült verzió a lehető legkevesebb változtatást tartalmazza a kiindulási fájlhoz képest.
6. Rose modellek támogatása szükségeltetik, de a kivitelezés úgy kell történjen, hogy könnyeden lehessen továbbfejleszteni más modelltípusokra is.
7. Legyen elindítható parancssorból megfelelő paraméterek használatával, ezáltal integrálható lesz ClearCase alá.
8. Az alkalmazás ne legyen operációs rendszer függő.

Jelenleg a piacon nem található olyan alkalmazás, ami teljesítené a fenti követelményeket, ebből kifolyólag született meg az ötlet egy ilyen program elkészítésére. A program neve *UML Merge Tool*.

A jelenleg létező alkalmazások közül az IBM Rational Rose *Model Integrator* nevű összefésülő programja áll a legközelebb a fenti követelményekhez. A *Model Integrator*-ból hiányzik a 5. követelmény által leírt funkcionalitás, mivel a program egy, a bemeneti fájloktól nagyon eltérő szerkezetű fájlt hoz létre összefésülés után. Ezen kívül csak kis mértékben biztosítja a szűrők használatát (4. követelmény), mivel a hasonló elemek elrejthetőek, de a változásokat nem lehet kiszűrni. A 6. követelményt is csak részlegesen teljesíti, mert csak a Rose modellek támogatását teszi lehetővé Ennek az alkalmazásnak a másik nagy hátránya az, hogy operációs rendszer függő, így a 8. követelménynek sem tesz eleget.

A *UML Merge Tool* alkalmazás teszi lehetővé mindazt, ami szükséges az UML modellek megfelelő összehasonlításához és összefésüléséhez. Ezért biztosít egy könnyen átlátható és kezelhető grafikus felületet, mely mögött komoly tudás rejtőzik az eredmény lehető legtökéletesebb megvalósítása érdekében. A program Java programozási nyelvben készült, így biztosítva a rendszerfüggetlenséget és a könnyű továbbfejlesztést.



2.1 kép: Egy fájl élete a verziókövető rendszerben

A *2.1. kép* egy fájl életciklusait mutatja születésétől kezdve néhány éven keresztül. A fekete vonalak jelzik a fő ágait a fájl életének. A piros vonalak összefésülést jelentenek, ez a két összekötött verzió összefésülése. A függőleges vonalak a számozott körök között összehasonlításokat jelölnek. Látható, hogy amíg a fájl egy úgynevezett stabil állapotba kerül, vagyis egy verziója létrejön a főágak valamelyikén belül, nagyon sok összefésülésen és összehasonlításon megy keresztül. Mivel eddig nem állt rendelkezésre olyan program, ami teljesíti a fenti követelményeket, ezeket az összehasonlításokat és összefésüléseket emberi kézzel, csak nagyon kis tudású (például szövegszerkesztő) alkalmazások segítségével lehetett elvégezni, ami nehézkes, nagymértékben megnöveli a hibalehetőségeket és nagyon időigényes.

Egy, már létező összehasonlító módszer szöveges fájlok összehasonlítására és összefésülésére a szöveges összehasonlítás és összefésülés. Az ilyen típusú összehasonlítás lényege, hogy karakterenként vagy soronként hasonlítja össze a fájlokat. Ez UML modellt tartalmazó fájl esetén nem használható, mivel nem veszi figyelembe az UML diagramok, objektumok létezését, hanem csak karakterek halmazának tekinti a fájlt. UML modellt tartalmazó fájl esetén nem számít a sorok sorrendje, az objektumokat leíró sorok felcserélhetőek egymással, mivel logikai szempontból nem változtatja meg a modell tartalmát, viszont szöveges összehasonlítás esetén már nem tudnánk követni az ilyen módosításokat. Egy másik probléma az, hogy az UML szerkesztő programok – ilyen például az IBM Rational Rose is – különböző verziói ugyanarról a modellről különböző formátumú fájlokat hoznak

létre, melyek logikailag azonos felépítésűek, de szövegesen összehasonlítva különböznek. A szöveges összefésülés esetén szintén olyan problémákba ütközünk UML modellek esetén, mint szöveges összehasonlításkor. Ezekből látszik, hogy a szöveges összehasonlítás összefésülés nem használható UML modelleket tartalmazó fájlok esetén.

2.3. A modell felépítése

A modell memóriában tárolt változata a modell-objektum. A modell-objektum egy olyan faszerkezetben tárolódik, melynek csomópontjai két nagy csoportba tartoznak:

1. egyszerűek: olyan elemek amelyek csak egy tulajdonságot tárolnak, ami a legtöbb esetben egy név és egy hozzá tartozó információ
2. összetettek: csomópontokként viselkednek, vagyis bizonyos tulajdonságokkal rendelkeznek és lehetnek gyerekeik. A rájuk jellemző tulajdonságok közé sorolhatóak a nevük, ID-jük, stb.

Fontos kiemelni, hogy a faszerkezet elkészítésekor nem az adatok fájlbeli sorrendje számít, hanem arra kell figyelni, hogy melyik elem mely más elemekkel áll kapcsolatban. A sorrend csak azért nem hanyagolható el, mert a lehető legkevesebb változást szeretnénk előidézni a bemeneti fájlokhoz képest, viszont ez logikai szempontból nem befolyásolja a modellt.

Az alkalmazás a modell-objektumot egy bemeneti fájl alapján készíti el. Miután kiválasztottuk a bemeneti fájlt az alkalmazás lexikálisan és szintaktikusan értelmezi azt a rekurzív leszállás módszerével és felépít a modell-objektumot.

A beolvasás során olyan plusz információk is eltárolódnak minden elem mellett, mint az adott elem fájlbeli alakja, azaz, hogy hány fehér karakter előzi meg illetve következik utána. Ennek az a jelentősége, hogy a lehető legkevesebb változást idézzük elő az összefésüléssel a bemeneti fájlhoz képest.

2.4. Modellek összehasonlítása

Az **összehasonlítás** két modell közötti különbségek meghatározása.

Nevezzük ezt a két modellt alap- és módosított verziónak. Az összehasonlítás során az alapverzióhoz viszonyított különbségeket vesszük figyelembe, melyeket három csoportba sorolhatunk: módosítások, törlések és hozzáadások.

A hozzáadás olyan új elem, amely nem szerepel az alapverzióban, viszont szerepel a módosítottban, így ez plusz információ az alapverzióhoz képest.

A törlések csoportjába tartoznak azon elemek, melyek szerepelnek az alapverzióban, de nincsenek benne a módosítottban.

A módosítások közé azon elemek tartoznak, melyeknek megváltozott valamelyik tulajdonsága. Ha az adott elemet azonosító tulajdonság változik meg, például az ID-je vagy bizonyos esetekben a neve, akkor ezt már nem tekintjük módosításnak, hanem úgy értelmezzük, mintha törölték volna az adott elemet és egy újat vezettek volna be helyette.

Az összehasonlítás konkrét folyamata során rekurzívan haladunk végig az alapverzió faszerkezetén és vele párhuzamosan haladunk a módosított verzión is, miközben lépésről lépésre hasonlítjuk össze a két modell-objektum hasonló típusú elemeit. Az elemek összehasonlítása már a modell típusától függően teljesen specifikus, mivel minden modellkészítő alkalmazás egyedileg tárolja az adatokat a modellfájlokban.

Ha az összehasonlítás során a két modell-objektum elemei megegyeznek, akkor egy erre alkalmas adatszerkezetben eltároljuk a kapcsolatot a két elem között, hogy később könnyedén találjuk meg egy adott elem párját a másik modell-objektumban. Ha különbségre bukkanunk, akkor ezt besoroljuk valamelyik, fentebb említett különbség-típusba és a neki megfelelő adatszerkezetben eltároljuk az elemet.

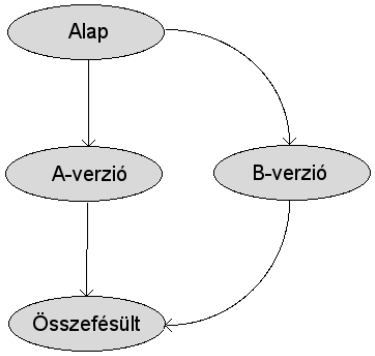
A megjelenítés során szükséges, hogy a különbség ne csak az adott elemnél látszódjon, hanem az elem szüleinél is egészen a faszerkezet gyökeréig, mert így sokkal átláthatóbb lesz a faszerkezetet és könnyebben megtaláljuk a különbségeket. Ezért az összehasonlítás utolsó lépésében összegyűjtjük az összes különbséget és ezek szüleit egészen a gyökérig és eltároljuk egy adatszerkezetben. Ezen összegyűjtött adatok segítségével könnyebben el lehet készíteni a képernyőn megjelenített fát.

2.5. Modellek összefésülése

Az **összefésülés** azt jelenti, hogy két, ugyanattól az őstől származó modellt egyesítünk. Ez általában verziókövető rendszerek használata esetén szükséges, amikor egy alkalmazás, és ebből kifolyólag a hozzá tartozó modell, párhuzamos, több szálon futó fejlesztése zajlik.

Modellek összefésüléséhez három modellre van szükségünk: *alapverzió*, *A-verzió* és *B-verzió*. Az *alapverzió* az az *A* és *B* verziók közös őse. Az összefésülés alatt azt értjük, hogy az *A-verziót* összefésüljük a *B-verzióval*. A közös ős ismerete azért szükséges, mert a különbségek megállapítása nem *A* és *B* között történik, hanem az *alapverzióhoz* képest.

Ezt szemlélteti a *2.2. kép* is.



2.2. kép: Összefésülés

Nézzük azt az esetet, amikor nem használjuk az *alapverziót*. Ilyen esetben csak azt tudjuk megállapítani az *A* és *B* elemeivel kapcsolatban, hogy az megváltozott vagy sem, de nem tudjuk eldönteni, hogy konkrétan melyik modellen is hajtották végre a változtatást. Ezt a következő példák is szemléltetik:

1. Ha egy elemet törölnek az *A*-ból, akkor nem lehet eldönteni, hogy az tényleg törlés-e, vagy pedig hozzáadás a *B*-hez.
2. Ha egy elemet módosítottak az egyik verzióban, akkor nem lehet eldönteni, hogy melyik verzióban módosult.
3. Ha egy elemet mindkét verzióban módosítottak, akkor csak annyit tudunk megállapítani, hogy az elem nem azonos a két verzióban, de nem tudjuk meghatározni, hogy melyik verzióban módosult és azt sem tudjuk meghatározni, hogy mindkettőben módosult.

Egy tetszőlegesen kiválasztott elem esetén az összefésült modellbe az elemnek azon

verziója kerül az *A* és *B* közül, amely megváltozott *A*-ban vagy *B*-ben az *alapverzióhoz* képest. Egy elemet akkor tekintünk megváltozottnak, ha az módosult, törölve volt vagy hozzáadódott a modellhez.

Ha nem változott az elem egyik modellben sem, akkor az *A-verzió*-belit választjuk. Ez az elem tartalmilag ugyanaz mind az *alap*-, *A*- és *B-verziókban*, de egyéb specifikus információkat is tartalmazhat, ami már különbözik.

Ha változott az elem *A*-ban vagy *B*-ben, de kizárólag csak az egyikben, akkor azt választjuk, amelyik megváltozott, viszont ha mindkét verzióban változott, akkor konfliktus lép fel. Konfliktus esetén az alkalmazás nem tud dönteni, ezért ilyen esetben a felhasználóra hárul a kívánt elem kiválasztása. Mindezeket figyelembe véve rájövünk, hogy konfliktus két esetben lép fel:

1. az elem módosult mindkét verzióban (*A*-ban és *B*-ben is)
2. az elem törlődött az egyik verzióból és módosult a másik verzióban (törlődött *A*-ból és módosult *B*-ben vagy törlődött *B*-ből és módosult *A*-ban)

Az esetek könnyebb áttekintését teszi lehetővé a *2.1. táblázat.*

A változtatás	Alap	A	B	Összefésült
Nem változott	x	x	x	x (A-ból)
Törlés A-ból és B-ből	x	0	0	0
Törlés B-ből	x	x	0	0
Törlés A-ból	x	0	x	0
Módosult B-ben	x	x	y	y
Módosult A-ban	x	y	x	y
Módosult mindkettőben	x	y	z	konfliktus
Hozzáadás A-hoz	0	x	0	x
Hozzáadás B-hez	0	0	x	x
Módosult A-ban, törlés B-ből	x	y	0	konfliktus
Módosult B-ben, törlés A-ból	x	0	y	konfliktus

2.1. táblázat: Az összefésülés szemléltetése

A *UML Merge Tool* az összefésülést a következő módon végzi:

1. Felépíti a három modell-objektumot a bemeneti fájlok (modellek) alapján.

2. Összehasonlítja az *alapverziót* az *A*-verzióval és eltárolja a különbségeket.
3. Összehasonlítja az *alapverziót* a *B*-verzióval és eltárolja a különbségeket.
4. A modellek és a különbségek alapján készít egy összesített fát, melynek minden csomópontja tartalmazza mindhárom verzióbeli elemet.
5. Az újonnan elkészült fát végigjárva automatikusan összefésüli azon elemeket, melyek esetében nincs konfliktus.
6. Grafikusan megjeleníti a négy modellt (a három kezdetit és az összefésültet) lehetőséget kínálva a felhasználónak az ellenőrzésre, módosításra és a konfliktusok eldöntésére.
7. A felhasználó parancsára összeállítja az összefésült modellt, kijavítva az esetleges összefüggésbeli hibákat és lementi azt egy fájlba.

Mint láthatjuk a konkrét összefésülés három fő lépésből áll: összesített fa felépítése, automatikus és manuális összefésülés, és az összefésült modell összeállítása.

Az összesített fa használata nagy mértékben csökkenti a program bonyolultságát és növeli a sebességét, mivel az adatok nagy része ugyanabban az adatszerkezetben kerül eltárolásra. Ha a modelleket nem vonjuk össze egy összesített fába, akkor az összefésülés végrehajtásához párhuzamosan kell haladni a három, hasonló, de nem egyforma faszerkezetben. Összefésülés után már négy fával kellene dolgoznunk, ez még tovább bonyolítaná a helyzetet és a program működését. Azért, hogy ne pazaroljuk a számítógép memóriáját, az összesített fa valójában csak hivatkozásokat (referenciákat) tartalmaz a modellekben levő elemekre. Ennek a fának a felépítése a három modell párhuzamos végigjárását vonja maga után.

Az automatikus összefésülés során a program minden olyan elem esetén az *A-verzió*-belit választja, amelyik nem áll konfliktusban. Ezt az összesített fa végigjárásával valósítja meg, és szintén ebben a fában eltárolja el az összefésült fa elemeit (amik szintén csak referenciák a megfelelő modellben található elemre). Konfliktus esetén a felhasználó dönt, ez az automatikus összefésülés végrehajtása után valósítható meg.

Ahhoz, hogy az összefésült modellünket fájlba tudjuk menteni, előbb fel kell azt építeni az összesített fa segítségével, hiszen itt vannak eltárolva a referenciák az összefésült fához tartozó elemekhez. Ez csak az automatikus és (ha szükséges) manuális összefésülés után valósítható meg.

Végeredményül egy újabb fájlt kapunk, ami az *alap-*, *A-* és *B-verziók* összefésült modelljét tartalmazza.

2.6. Adatok szűrése

Az **adatok szűrése** teszi lehetővé, hogy a modell bizonyos elemeit figyelmen kívűl hagyjuk. Ez modellek összehasonlításánál azt jelenti, hogy nem tekintjük különbségnek a kiszűrt elemeket, összefésülésnél pedig a kiszűrt elem *A*-beli verzióját tesszük az összefésült modellbe.

A gyakorlati tapasztalatok azt mutatják, hogy ritka az, amikor a felhasználó egyszerű elemeket szeretne kiszűrni, inkább az a megszokott, hogy egy adott struktúrát (osztály, névtér, függvény) teljes mértékben figyelmen kivűl szeretne hagyni. Ez úgy valósítható meg, hogy a kiszűrt elem faszerkezetbeli gyerekei is kiszűrődnek. Így a fa egy teljes ágát figyelmen kívül tudjuk hagyni egy szűrő bevezetésével.

A *UML Merge Tool* a szűrést a grafikus felületen egy táblázat segítségével teszi lehetővé, itt vihetünk be bizonyos jellemzőket, amelyek szerint szűrni kívánjuk az elemeket: típus, altípus és név. A szűrés végrehajtása során a program eltávolítja a kiszűrt elemeket a már eltárolt különbségek közül. A felhasználó dolgát könnyíti meg a szűrők automatikus elmentése a program bezárásakor illetve automatikus betöltése a program indításakor.

Ha a felhasználó nem kíváncsi a modellek azonos elemeire, csak a különbségekre, és csak ezeket szeretné látni, akkor ezt könnyen megteheti, hiszen az alkalmazás lehetőséget biztosít az azonos elemek elrejtésére. Ilyen esetben ha szűrőt is alkalmazunk, akkor a kiszűrt elemek nem jelennek meg a képernyőn. Ez nagy méretű és kevés változtatást tartalmazó modellek esetén a leghasznosabb, mert így azonnal észrevesszük a változtatásokat.

2.7. Továbbfejlesztési lehetőségek

Ebben a fejezetben az alkalmazás továbbfejlesztésére szeretnék néhány ötletet bemutatni.

Az első, és talán egyik leghasznosabb új funkcionalitás az lenne, hogy ne csak a különbségek esetében lehessen választani, hogy melyik modellből kerüljön az összefésültbe az elem, hanem azonos elemek esetében is választhassunk. Az azonos elemek csak tartalmilag egyformák, de a fájlbeli alakjuk különböző lehet, azaz másképp helyezkednek el körülöttük a fehér karakterek. Ez a plusz funkcionalitás nagyobb teret adna a felhasználónak a kimeneti fájl alakjának meghatározására.

Egy másik továbbfejlesztési lehetőség az automatikus összefésülés tulajdonságainak kibővítése. Ez elsősorban választási lehetőséget biztosítana arra, hogy szeretnénk-e használni az automatikus összefésülést, illetve megszabhatnánk pár apró feltételt az összefésüléssel kapcsolatban. Lehetőséget kell biztosítani a felhasználó számára, hogyha kezdetben nem alkalmazta az automatikus összefésülést, akkor legyen lehetősége később használni azt a még el nem döntött különbségek esetén.

Bizonyos, modelleket tartalmazó fájlokon belül az elemek közötti kapcsolatot hívatkozások oldják meg. Törölt elem esetén felléphet egy olyan állapot, hogy valamely elem egy kitörölt elemre hívatkozik. Ilyenkor figyelmeztetni kell a felhasználót a hibáról és lehetőséget kell neki adni annak orvosolására.

Egy, az előzőeknél sokkal bonyolultabb továbbfejlesztési lehetőseg az az összefésült modell logikai vizsgálata lenne. Még ha az összefésülés sikeres és konfliktusmentes is volt, akkor is felléphetnek olyan logikai hibák, melyek észrevétele és kijavítása nehézkes. Ezek a hibák már a modell használatakor vagy programozáskor léphetnek fel és olyan jellegűek, melyek a modell logikai felépítésénél alakulnak ki. Egy egyszerű példa: az összefésülés során törlődik egy olyan osztály, amely a szülője egy másik osztálynak, viszont a származtatott osztály benne marad a modellben – látható a logikai hiba, lesz egy osztályunk ami származik egy nem létező osztályból.

3. Összefoglalás

Az UML modellek összehasonlítására és összefésülésére nagy szükség van az iparban, mivel minden alkalmazás megtervezése és továbbfejlesztése esetén több ember dolgozik a modelleken és egy ilyen eszköz segítségével viszonylag könnyen össze tudják hasonlítani vagy fésülni munkáikat.

A *UML Merge Tool* egy ilyen alkalmazás, mely megvalósítja az UML modellek összehasonlitását és összefésülését, miközben olyan új funkcionalitásokkal rendelkezik, amilyennel más, hasonló programok nem. Az alkalmazás legnagyobb előnye, hogy könnyen továbbfejleszthető bármilyen UML modelleket tartalmazó fájltípusra, és mindemellett rendszerfüggetlenséget biztosít.

A *UML Merge Tool* egy hasznos segédeszköz minden UML modellekkel dolgozó ember számára.

4. Könyvészet

1. UML honlap: www.uml.org/
2. Vállalati modellezés UML segítségével: www.econ.klte.hu/oktatok/tarnoczi/DSS_N/szeminarium01.ppt
3. Wikipédia: http://hu.wikipedia.org/wiki/Uml
4. StarUML help
5. Rational Rose help
6. Model Integrator: http://www.interface.ru/rational/rose/rp8.htm
7. Az IBM Rational Rose petal fájl: http://www-1.ibm.com/support/docview.wss?rs=934&context=SSJPA5&dc=DB520&dc=DB560&uid=swg21134800&loc=en_US&cs=UTF-8&lang=en&rss=ct934rational