Diplomamunka

Varga István

Debrecen

2009

Debreceni Egyetem

Természettudományi Kar
Informatikai Intézet

Hálózati mérési keretrendszer kidolgozása

Témavezető: Dr. Végh János

Készítette: Varga István
Informatika – Matematika

Debrecen 2009

Tartalomjegyzék

1 Egy érdekes és bonyolult probléma

1.1 A probléma

Hogyan lehet egy fizikai kísérlethez használt mérőberendezéseket távolról vezérelni egy program segítségével úgy, hogy az megjelenésében és használhatóságában ne függjön az alkalmazott platformtól? Bármilyen mérőberendezés vezérlésére képes legyen a megfelelő beállítások után.

Az eddig fellelhető ilyen jellegű programok nagy része túlságosan is eszköz - és platformfüggő, azaz nem használható két teljesen különböző platformon és más eszközhöz. Vagy ha találnánk is ilyet, ami akárcsak részben is teljesíti a kívánalmakat, licenc szerződés nagyban korlátozza a használhatóságát, vagy szinte már csillagászati összegekbe kerülnek. Többek között ezért is választottam ezt a témát szakdolgozatom témájául.

A célom egy olyan keretrendszer, idegen szóval framework, kidolgozása és megtervezése amely, képes kezelni a különböző típusú mérőberendezéseket, azok adatait, jelzéseiket, felügyelni működésüket a futtatott operációs rendszertől és platformtól függetlenül. Egységes felületet biztosítani az információk, eszközök megosztásához és kezeléséhez. A keretrendszernek alkalmazkodnia kell tudnia a hardveres változásokhoz, anélkül, hogy újraírnánk az egész rendszert vagy annak egy részét is. Azaz minél többféle helyzetben általánosan felhasználható legyen. A modern méréstechnikában egyre bonyolultabb, összetettebb és több mérési berendezés együttműködését megkövetelő mérési elrendezésre van szükség, amelyeknek kezelése igen nagy feladatot jelet heterogén rendszerek esetén. A felhasználói hozzáférések is különbözőek lehetnek. A framework -nek képesnek kell lennie a beállítások elmentésére és visszatöltésére, a hibák felismerésére és kezelésére.
A projekt célja, hogy egy olyan keretrendszert hozzon létre, amelyben

- a mérőberendezéseket közvetlenül kezelő egységek és a munkát koordináló egység önállóan működjön (Windows: service, Linux: demon)
- az egységek egymással TCP/IP alapú kommunikációval tartják a kapcsolatot
- együttműködni képes a Ethernet alapú mérésmegjelenítő alapszoftver projekt

termékével

- az üzenetek lehetővé teszik parancsok és adatok továbbítását
- az állapotok konfigurációs fáj(ok)ba elmenthetők és visszaállíthatók
- a rendszer grafikus felület(ek)ről irányítható
- a konkrét mérőberendezések kezelésének beillesztése egyszerű mintha egy bővítményt adnánk a rendszerhez ami éppen az aktuális eszközt képes kezelni az adott környezetben

A beállításokat XML fájlokban lehet tárolni vagy akár egy adatbázisban , amelyekből könnyen be lehet olvasni és ellenőrizni. A felhasznált wxWiddets eszközrendszerben találhatóak XML kezelőeszközök. Az adatok tárolására és továbbítására is biztosítani kell az egységes felületet amelyen keresztül kezelhetőek és feldolgozhatóak lesznek. Erre szintén egy jó megoldást kínál az XML formátum mert szabványos és a hibák is könnyebben azonosíthatóak.

A keretrendszer(framework) együttműködő osztályok összessége egy bizonyos programtípus számára, amelyek egy újrafelhasználható szerkezetben egyesülnek. A keretrendszert úgy kell testre szabnunk egy adott alkalmazáshoz, hogy az alkalmazásfüggő osztályokat származtatjuk a keretrendszer elvont osztályaiból. Így a keretrendszer meghatározza az alkalmazás felépítését. Meghatározza az általános szerkezetet annak osztályokra és objektumokra bontását, illetve a kulcskötelességeket, vagyis azt hogyan működnek együtt az objektumok és osztályok, valamint meghatározza a vezérlés haladási irányát. A keretrendszer mindeme tervezési paramétereket előre megadja, hogy az alkamázas tervezőink, illetve megvalósítóinak csak az alkalmazás lényegére keljen koncentrálniuk. A keretrendszer azon tervezési döntéseket foglalja magában, amelyek az adott felhasználási területen általánosak.

A fizikai méréseknél nagyon fontos tényező az idő, ez alól ez a keretrendszer sem kivétel. Talán az egyik legfontosabb elvárások közé tartozik a megfelelő válaszidő a bekövetkezett eseményekre. Hiszen nem várakozhatunk addig amíg például egy virtuális gép befejezi a szemétgyűjtögetést a memóriában és csak ezután foglalkozik a a bekövetkezett eseményekkel mert közben értékes időt vesztünk.

Fontosabb kérdések:	
Hordozhatóság	Milyen keretek között?
Milyen eszközöket használunk fel a tervezés közben	Mik a felhasznált eszközök előnyei és hátrányai?
Tervezési döntések	Milyen következményekkel járnak a tervezés közben meghozott döntések,mik az előnyei és a hátrányai?

Ezért csak olyan megoldások jöhetnek szóba, ahol a válaszidő a lehető legkisebb és ez nem megy a hatékonyság rovására. Emellett a lehető legszélesebb körben használhatónak, jól dokumentáltnak kell lennie.

1.2 Hordozhatóság kihívása

Hordozhatóság alatt azt értjük, hogy ha egy programot, forráskódot átviszünk egyik gépről a másikra és ott elindítjuk, lefordítjuk akkor ugyanazt az eredményt kapjuk mint az eredetin anélkül, hogy változtatnánk egy kicsit is rajta. Függetlenül attól, hogy például forráskódok esetében milyen fordítót használunk, vagy milyen operációs rendszert. Ez talán az egyik legnehezebb kihívás a fejlesztés során, hiszen figyelni kell az implementációs különbségekre is a fordítók esetében. Többek között ezért is került a választásom a wxWidgets toolkit -re, mivel képes figyelembe venni az egyes fordítók közötti különbségeket, és elrejteni azokat anélkül, hogy komolyabb megvalósítási többletmunkát jelentene a kódolás során.

2 Felhasznált eszközök katalógusa

A tervezési és fejlesztési munkámat Ubuntu Linux alatt végeztem, de erre bármilyen Unix , Linux disztribúció vagy MS Windows is képes megfelelő támogatással, fordításhoz szűkséges csomagokkal és a futtatáshoz szükséges környezettel. Azért esett a választásom erre az operációs rendszerre mert, a fejlesztői eszközök széles skáláját biztosítja a programozáshoz és a felhasználónak egyaránt. Ezek az eszközök elérhetőek és egyszerűen telepíthetőek, általában a valamelyik csomagkezelőben egyszerűen kiválaszthatóak és telepíthetőek, de ha Windows rendszer alatt akarunk fejleszteni akkor a szükséges eszközöket telepítenünk kell a rendszerre az is lehet hogy egyenként vagy egy csomag keretében.

De talán az egyik legfontosabb dolog ami kiemeli az itt felsorolt eszközöket, programokat, rendszereket, hogy a forráskódjuk szabadon hozzáférhető és használható a GPL v2 licenc alapján, vagy hasonló licenc feltételekkel ingyen használható. Ami kiemeli a hasonló több platformon elérhető fejlesztőeszközök közül, és még egy további előnye ezeknek az eszközöknek, hogy részletekbemenően dokumentáltak, és több nyelven is elérhetőek, de elsődlegesen angolul. További fontos előnye még, hogy a felhasznált eszközök, a fejlesztéshez elérhetőek más platformokra is.

2.1.1 wxWidget tools(eszközök)

Ez egy GUI építő osztálykönyvtár amellyel platformtól függetlenül lehet fejleszteni c++, Python nyelven és manapság Perl nyelven is. Ezek az osztálykönyvtárak nagyban megkönnyítik és felgyorsítják a fejlesztés folyamatát, mivel már előre meg vannak írva olyan osztályok amelyek biztosítják az egységes megjelenést, rendszerhívásokat, adatbázis kapcsolatot stb. A platformfüggetlenséget úgy éri el ez az elemkészlet, hogy az egyes platformokra, fordítókra jellemző részeket makróhívások segítségével elrejti és így egy egységes programozási felületet biztosít a programozó számára. A toolkit minden egyes eleme jól dokumentált, minden osztálynak megtalálható a pontos dokumentációja és funkciója.

Mint oly sok mindent ezt is folyamatosan fejlesztik a projekthez. Én a 2.8.6-os verziójú elemkészletet használtam. Ez az elemkészlet a wxWidgets Library licenc alapján szabadon

használható és másolható, mint a GPLv2 -ben, de maga az osztálykönyvtár nem módosítható és nem is érdemes, mivel ez egy igen jól karbantartott és jól dokumentált kódok amik szabadon felhasználhatóak.

A wxWidgets C++ framework nem csak egyszerűen egy GUI építő egyéb lehetőségekkel. Kettes verzió jelenleg támogatja MS Windows, Unix GTK, Unix Motif, MacOs platformokat, de ezen kívül még fejlesztés alatt van többi platformra is mint például a Os/2, Windows CE. A választásom többek között azért is esett erre rendszerre mert nem csak, hogy nyílt forrású (open source) hanem azért is mert több fordítóprogram is támogatja, olyan nagyok is mint a MS VisualStudio, gcc, mingw. Ez a toolkit épp úgy alkalmas kis programok fejlesztéséhez mint nagy robusztus programrendszerekhez. Talán lehetetlen összefoglalni az előnyeit mivel igen sok van, de fel lehet sorolni egy párat:

- alacsony költség (ingyenes forrás)
- megkapjuk a forráskódot amit szabadon változtathatunk, ha megfelelően dokumentáljuk a változtatásainkat és az így kapott rendszerre szintén a GPL -licence lesz érvényes
- az összes népszerű platformhoz elérhető vagy elérhető lesz hamarosan
- majdnem minden C++ fordítóprogrammal működik és több porgrammozási nyelvhez is elérhető
- több mint 50 példaprogram és alkalmazások
- több ezer oldal dokumentáció, tippek trükkök és útmutatások
- egyszerűen használható objektum orientált API

[wxWidgets]

2.1.1 Autoconf

Ez egy konfigurációs szkript generátor, amellyel automatikusan konfigurálhatjuk a forráskódot és szoftver csomagokat bármilyen POSIX szabványnak megfelelő rendszerre. Ezzel az eszközzel adoptálhatjuk a forráskódunkat a célrendszernek megfelelően. Figyelembevéve az adott rendszerre jellemző és szükséges programfordítási beállításokat összegzi és előállítja a program lefordításához szükséges konfigurációs fájlokat. Figyelembe veszi a szükséges feltételeket, könyvtárakat, környezeti beállításokat, rendszer követelményeket, anélkül hogy a forráskódunkban egyetlen egy karaktert is módosítani kellene. Ezzel a lehetőséggel igen nagy hibalehetőséget is meg lehet szüntetni mivel nem kell különböző rendszerekhez és könyvtárakhoz külön külön megvalósításokat elkészíteni.
[Autoconf]

2.1.2 Kdevelop

Egy integrált fejlesztői eszköz Unix (Linux) rendszereken, amely több programozási nyelvet is támogat, hibakeresési eszközökkel és sok egyéb lehetőségekkel. Azért döntöttem ezen fejlesztői eszköz mellett, mert sok munkát meg lehet vele spórolni, mivel támogatja a moduláris programozást is,és ezt a projekt csak úgy lehet megvalósítani, hogy egynél több embert kell bevonni a közös fejlesztési munkába, ami sajnos eddig nem sikerül, mivel nem találtam megfelelő embert. A környezet eszközöket kínál a projekt fordítási egységekre bontására, egyesítésére függőségi fák kialakítására, programkönyvtárak létrehozására és még sok egyéb a fejlesztéshez nélkülözhetetlen eszközök kezelésére. Eredetileg a KDE Unix -os grafikus felület fejlesztéséhez készült de idővel kinőtte azt és több programozási és Script nyelvet támogató integrált fejlesztői környezet lett belőle.
[Kdevelop]

2.1.3 Automake

A forrás fordítási folyamatát leíró Make fájlok automatikus generálásához szükséges program, amely az autconf program által generált Make.am konfigurációs fájlok alapján hozza létre a fájlokat.

Ezt a programot a forrásfájlokat tartalmazó minden könyvtárra és alkönyvtárra futtatni kell, hogy a megfelelően történjen a függőségi fa felépítése a projekt fordításához. Ezen eszköznek a paraméterként átadott fájlnév lényegében egy script, amelynek szerkezete nagyon hasonló az Unixos Shell scriptekéhez.
[Automake]

2.1.4 Make

Fordítási folyamatot leíró Make fájlokat értelmező program. A program képes a beállítási fájlok alapján elkészíteni, újrafordítani az adott projektet és csak a feltétlenül szükséges forrásokat fordítja le vagy újra a függőségi fa alapján, így nem kell az egész projektet az elejétől újra fordítanunk és linkelnünk, hanem csak azokat a részeket amiket megváltoztattunk vagy szükségessé váltak a projekthez. Ezzel időt spórolhatunk meg, és nem kell foglalkoznunk az esetleges régebbi részekkel, amelyeket esetleg nem is mi fejlesztünk és tartunk karban. Nagyban leegyszerűsíti és megkönnyíti a moduláris programozást és fejlesztést.
[GNU Make]

2.1.5 C++ fordítókról általában

Unix (Linux) disztribúciók alatt leggyakrabban elérhető a fordító a GCC ezen fordítónak létezik Windows platformokon is elérhető változata (MinGW, Microsoft Visual C++), de léteznek más fordítók is, amelyek támogatják a szükséges wxWidgets kódok fordítását. Ezek a fordítók nem mindig a szabványos C++ hivatkozási nyelvet támogatják, vagy ha igen,

akkor is különböznek a hivatkozott szabványtól. Felvetődnek az implementációs kérdések is, mert a szabvány nem tér ki minden egyes részletre így a megvalósításnál igen komoly rejtőzködő hibalehetőségek jelentkeznek. Az egyik ilyen érdekes különbség és egyben hibaforrás is a lokális változók hatásköre.
PL: for(int i=0;i<10;++i) esetén az i változó hatásköre nem csak a cikluson belüli hanem a ciklust tartalmazó blokkra is kiterjedhet és ezt nem minden fordító kezeli le jó és programozókról akkor ne is beszéljünk. Nagy hiba forrása is lehet ahol a hiba okát igen nehéz felderíteni mivel nem ott jelentkezik ahol azt várnánk.

A C++ nyel igen hatékony programozási nyelv, de ennek a hatékonyságnak gyakran ára is van. Minél hatékonyabb egy C++ kód sokszor annál átláthatatlanabb és bonyolultabb megfelelő dokumentáció nélkül néha egyenesen érthetetlen vagy igen alaposan tanulmányozni kell az adott kódrészletet, ami igen időigényes a kód hosszától függően. Mivel ez a nyelv egyszerre tartalmaz OO- és eljárás orientált eszközöket ezért fontossá válik az igen alapos tervezés még mielőtt hozzákezdenénk a probléma megoldásához. El kell tervezni, hogy pontosan mit és hogyan szeretnénk egészen a kód szintjéig és ezt többször is meg kell tennünk.
[A C++ Programozási nyelv I.]
[A C++ programozási nyelv II.]

2.1.5.1 Gcc

Gcc vagy más néven GNU Compiler Collection (fordító gyűjtemény). Ez a kollekció több programozási nyelv gyűjteményét is magában foglalja többek között olyan „régi” nyelvekét is mint a C, de ezen kívül még C++, Pascal, Objective C, Java is. Elsődlegesen Linux, Unix és BSD rendszerekre (POSIX szabványnak megfelelő rendszerek), de ezen kívül létezik egyéb platformokra megvalósított változata is. Rengetek processzor és architektúra típust támogat így szinte bármilyen gép nyelvére le tudjuk fordítani a programjainkat anélkül, hogy olyan nagyon alacsony szintű programozási nyelvekhez kellene fordulnunk mint az Assembler. Bár kétségtelen ennél hatékonyabb nyelvet nem is találhatnánk, de ez nagyon gép specifikussá tenné a programjainkat és ez a hordozhatóság kárára menne.

A GCC fordító eredetileg C fordítóként kezdte, és ennek köszönheti sok nyílt forrású operációs rendszer a fejlődését. Az eltelt évek során igen hatékony és kényelmes eszközé vált ami már már nélkülözhetetlen a programozók számára. Persze mint minden fordítónak ennek is megvannak a hátrányai és előnyei.

[GNU Gcc]

2.1.6 Doxygen dokumentáció generáló toolkit

Dokumentáció generáló program különböző programozási nyelvekhez, úgy mint C++, Python, C , Java. Ennél a dokumentációkészítő rendszernél a dokumentációs megjegyzéseket a forráskódban kell elhelyezni, amit a program automatikusan elemez, felismer és elkészíti az adott rész dokumentációját a megjegyzésben foglalt parancsok alapján. Ezzel a toollal készíthetünk html, pdf, dokumentációkat egy időben és ezeket felügyelni és karbantartani is tudjuk. Opcionálisan használható hozzá a DOXYWIZARD nevezetű kiegészítési, amellyel pontosíthatjuk a beállításainkat egy felhasználói felületen keresztül , megadatjuk milyen formában szeretnénk látni a dokumentációnkat.
[Doxygen manual]

2.1.7 SVN

A Subversion verziókövető rendszert használtam a verziókövetésre. Ezen rendszer teljesen ingyenes (free/ open source) verziókövető rendszer, amely képes menedzselni könyvtárakat fájlokat és azok változásait az idők folyamán. Így lehetővé teszi számunkra, hogy akár egy régebbi változatra is visszaálljunk anélkül, hogy a változtatásaink teljesen elvesznének így megvizsgálhatjuk a változtatásainkat és elemezhetjük őket különösebb probléma nélkül. Ebben a vonatkozásában az emberek úgy tekintenek rá mint egy időgépre. De nem csak erre képes, hanem képes hálózaton keresztül is működni összekapcsolva több ember munkáját, és itt jelentkezik az igazi erőssége is hiszen a változtatásokat a feltöltés után mindenki elérheti és az éppen aktuális verzióval dolgozhat tovább, anélkül hogy személyesen

kellene elkérnie minden egyes embertől az adatokat aztán valahogyan összeillesztenie őket. Ezen eszköz egyik nagy előnye, hogy képes bármilyen típusú fájlokat kezelni a hozzájuk tartozó könyvtárszerkezettel együtt legyen az akár forráskód, dokumentumok vagy képek és videók.

2.1.7.1 Verziókövetésről általában

Talán az egyik legfontosabb kérdés miért is van szükségünk verziókövetésre és milyen előnyökkel illetve hátrányokkal jár?
A verziókövetés előnyei:

- Minden egyes változtatás visszavonható ha szükséges.
- Minden változtatás dokumentálva van, hogy ki és mikor változtatott a felügyelt fájlokon és könyvtárakon.
- Rossz változtatások kiszűrése és visszavonása
- Információmegosztás
- Lehetővé teszi a csoportos munkát egyazon a fájlon a felülírás és az adatvesztés veszélye nélkül.
- Központilag tárolt adatok helyi másolatokkal

Számtalan előnyét fel lehetne még sorolni, de a hátrányokkal is tisztában kell lennünk.
A hátrányai lehetnek a következők:

- Központlak tárolt adatok hibája esetén elveszhetnek adatok.
- Mindenki láthat minden információt úgy mint ki mikor mit csinált
- Hozzáférések kérdése, kinek mihez van joga

[SVN book]

2.1.8 Dia strukturált diagramszerkesztő

A keretrendszer tervezése során talán az egyik legfontosabb, hogy az egyes rendszerkomponensek közötti kapcsolatokat valamilyen vizuális formában megjelenítsük.

Ezzel nagyban megkönnyíthetjük a tervezési folyamatot és az egyes komponensek között kapcsolatokat is könnyebben megjeleníthetjük. Így megkönnyíthetjük az egyes komponensek közötti kapcsolatok megértését és átláthatóbbá válik a komplett rendszer legyen az akármennyire is bonyolult. Erre a célra leginkább az elméletben és sokszor a gyakorlatban is használt UML (Universal Modeling Language) felel meg, mivel elég sok területen alkalmazzák és használják a mindennapi gyakorlatban.
[Dia diagram]

2.1.8.1 UML(Universal Modelling Language)

Az objektumorientált programozás szabványos specifikációs nyelve. Az UML grafikus jelöléseket használ rendszerek absztrakt modelljének leírására. A szabvány UML jelöléseket használó diagramokat UML modellnek nevezzük. Az UML az Object Management Group (OMG) fejlesztése, amelyet azzal a céllal hoztak létre, hogy elősegítse az objektumorientált rendszerek elterjedését. Időközben nemzetközi szabvánnyá nőtte ki magát az UML nyelv, köszönhetően annak, hogy olyan nagy nevek is felsorakoznak az OMG -ben mint az IBM, Sun, Apple. Az UML 2.0 -s verzió 13 különböző diagram típust definiál, melyek különböző kategóriákba és alkategóriákba oszthatóak.

Strukturális diagramok, amelyek modellezett rendszer elemeire vonatkoznak. Altípusai a következők:

- Osztálydiagramok: Egy statikus modell, amely megmutatja az osztályok attribútumait továbbá az osztályszintű kapcsolatokat is.
- Komponensdiagrammok: A rendszer fizikai komponenseit és az azok közötti kapcsolatokat és függőségeket mutatja meg, ezek a komponensek lehetnek fájlok, header, csomag vagy akár egy futtatható állomány is
- Összetett struktúradiagramok: Az osztályok belső szerkezetét mutatja és azt, hogy az adott szerkezet milyen kollaborációkat tesz lehetővé.
- Telepítési diagramok: A rendszerimplementációhoz használt hardvert, a hardverre

telepített szoftverkomponenseket és a köztük lévő viszonyokat reprezentálja.

- Objektumdiagramok: A modellezett rendszer egy adott időpillanatbeli állapotát ábrázolja, mint egy pillanat felvétel. Az osztályok példányait és kapcsolatait jeleníti meg. Konkrétabb az osztálydiagramnál mivel az osztályok példányainak kapcsolatát jeleníti meg az osztálykapcsolatok helyett.
- Csomagdiagramok: Azt mutatja meg, hogyan szerveződnek a szoftverkomponensek csomagokba illetve hogyan viszonyulnak ezek egymáshoz.

A másik nagy kategória a viselkedési diagramok, amelyek azt írják le hogy minek kell történnie a modellezett rendszerben, ezen kategória alkategóriái a következők:

- Aktivitás diagrammok: A munkafolyamatot vagy angolul a workflow -t modellezik
- Állapotgép diagramok:A rendszer lehetséges állapotait modellezik vele.
- Use case diagramok: A rendszer használati eseteit fogalmazzák meg velük
- Interakciós diagramok
- Kommunikációs diagramok
- Szekvencia diagramok

Bár az UML igen széles körben elterjedt és használt szabvány igen gyakran kritizálják a hiányosságai miatt is:

- Túlságosan nagy és bonyolult mert a szabvány sok diagramot tartalmaz, amelynek nagy részét alig használják, nagy része pedig redundánsan található meg benne.
- Pontatlan szemantika, mert nincs a formális nyelveknél megszokott szigorú definíció.

3 Tervezés lépései

Még mielőtt bármihez is hozzákezdenénk először is tisztázni kell, hogy mit is akarunk pontosan csinálni,és mi is a feladat. Meg kell adnunk egy specifikációt a projekthez, ami pontosan leírja minden jellemzőjét, részleteit,a felhasználni kívánt eszközöket és a technológiákat, valamint a rendszerkövetelményeket. Tehát először is a feladat pontos megfogalmazása a legfontosabb és csak ezután kezdhetjük el a rendszer megtervezését.

Mivel ha nem így tennénk olyan projekthez kezdenénk, amiben nincs tisztázva a pontos cél így fennállna az a veszély, hogy a projekt végeredménye nem is arra a feladatra készült mint amire azt eredetileg kitűzték.

A programozási nyelv megválasztása is fontos szerepet játszik, mivel befolyásolja a programozó nézőpontját már a tervezés korai szakaszában anélkül, hogy bármit is kódolt volna. A választás alapján a nyelvi tulajdonságok meghatározzák, hogy mit lehet és mit nem lehet könnyen megvalósítani.

3.1.1 Modell- View-Controler

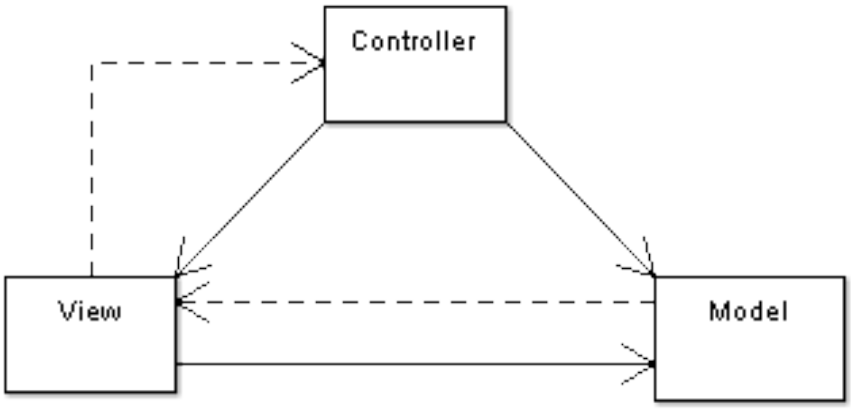
Ez a tervezési minta legfontosabb jellemzője, hogy az alkalmazások, keretrendszerek tervezését és annak megvalósítását 3 fő részre bontjuk. Ezáltal az üzleti logika határozottan elkülönül a prezentációs logikától, amely jelentősen megkönnyítheti a megvalósítás fázisát is. Az alkalmazás vagy keretrendszer vezérlése központosított. A különböző funkciókat ellátó egységek jól definiáltak és határozottan elkülönülnek egymástól, így csökkentve a felesleges ismétlődéseket és növelve az újrafelhasználhatóságot.

- Modell: A modell feladata, hogy a rendszer szempontjából nélkülözhetetlen erőforrásokat megvalósítsuk az adatstruktúrákkal. A modell felelőssége az adatok kezelése.
- View(megjelenítő): Feladata a felhasználói felület más néven a GUI(Graphical User

Interface) megjelenítése. Ide tartoznak, azon eszközök felhasználása, amelyek az adott felületen kulcsfontosságúak. Például egy beállítási felületen az adatok bevitelére szolgáló mező valamint az ezek kezelését elősegítő gombok is.

- Controller (vezérlő): A vezérlő feladata, hogy a felhasználói lépéseket (pl.: adatbevitel, adatlekérés, menüpont választás) parancsokká illetve függvényekké történő leképezése . Az adatok feldolgozása után értesíti view-t a változásokról, ami megjeleníti az újonnan kapott információkat, a megfelelő formátumban.

Ezen komponensek szétválasztásával és a kapcsolódási felületek pontos leírásával és megvalósításával az elkészített alkalmazás részei között laza csatolás lesz a jellemző. Ez a laza csatolás lehetővé tesz, hogy a későbbi fejlesztések során ne keljen az egész rendszert újraírni és az újraírás következtében a fellépő hibák nagy része is kiküszöbölhető ezen elvek használatával. A következő ábra az MVC minta egységeinek kapcsolatát reprezentálja, ahol a folytonos vonalak a közvetlen kapcsolatot jelentik, míg a szaggatott vonalak a közvetett elérést.



1. ábra. Az MVC egységek kapcsolatai.

3.1.2 Specifikáció

A specifikáció talán az egyik legfontosabb része minden projektnek, mert ebben a részben kerül pontos megfogalmazásra, hogy a készülő terméket mire használhatjuk, és mire nem, továbbá meghatározzák a pontos rendszerparamétereket, elvárásokat, továbbfejlesztési lehetőségeket.

A keretrendszer arra készült, hogy egy vázat képezzen és továbbfejlesztési irányt mutasson a hasonló típusú alkalmazásokhoz. Ezt a keretrendszert a későbbiekben tovább lehet fejleszteni, ki lehet bővíteni, további funkciókkal. Erre keretrendszerre a GPL v2 licenc vonatkozik, így a fejlesztéséhez bárki csatlakozhat, a licenc feltételei alapján.

A keretrendszerrel szemben támasztott egyik legfontosabb igény a több platformon való alkalmazhatóság és használhatóság komolyabb változtatás nélkül, ezért a keretrendszerhez használt wxWidget minimális követelménye a választott platformon futó C++ fordítóval szemben, egyben meghatározz a keretrendszer szükséges minimumot is ami:

- MS windows rendszerek esetében 32 vagy 64 PC-k esetében:
  - Windowsos jelenleg támogatott fordítók: MS Visual C++ ,Borland C++, Watcom C++, Cygwin, MinGW, Metrowerks CodeWarrior, Digital Mars C++.
    Egyéb támogatott fordítók megtalálhatók a wxWidgets dokumentációban.[wxWidgets]
  - 120 MB szabad hely a lemezen, a forráskódnak, programkönyvtáraknak, és alkalmazás lefordításához szükséges függőségi fa elkészítéséhez.
- Unix (Linux)
  - Majdnem minden C++ fordító, beleértve GNU C++ (EGCS 1.1.1 vagy újabb).
  - Majdnem minden Unix munkaállomás, és amelyiken: GTK+ 1.2, GTK+ 2.0, Motif 1.2 vagy magasabb verziószámú.
    A wxX11 port használata esetén nem szükséges egyébb widget készlet.
  - Legkevesebb 120 MB szabad hely a a forráskódnak, programkönyvtáraknak és további hely az egyéb választható könyvtárakhoz.

• Mac OS/ Mac OS X rendszerek esetén:

◦ Power PC -futó Mac OS 8.6/9.x vagy Mac OS X 10.x.

◦ CodeWarrior 5.3 vagy 7- es a klaszikus Mac OS -hez

◦ Apple Developer tools (mint a GNU C++), CodeWarrior

◦ Legkevesebb 120 MB szabad hely a a forráskódnak, programkönyvtáraknak és további hely az egyéb választható könyvtárakhoz.

Az elkészült keretrendszerrel szemben talán az egyik legfontosabb követelmény, hogy minden eseményre, a lehető leghamarabb reagáljon.

A keretrendszert a kliens szerver modell alapján, terveztem és ezt tartottam a legfőbb szempontnak a tervezés során. A szerver egyszerre több klienst is képes legyen fogadni és foglalkozni velük. A rendszer egy szerverprogram, amely kezeli a hozzá érkező kéréseket és felügyeli a mérőeszközöket kezelő folyamatokat és ezek eseményeit és jelzéseit figyelembe veszi. A hardver eszközöket kezelő folyamatokat a szerverprogram csak felügyeli és adataikat kezeli, de nem jeleníti meg őket. A hardvereszközöket kezelő egységek nem részei a programnak és a keretrendszernek, a szerver csak egy pontosan meghatározott felületen keresztül kommunikál az egységeket közvetlenül kezelő folyamatokkal. A processzek közötti kommunikáció is többféle lehet ezt a beállítási XML fájlokban lehet megadni. A beállítási fájlokban lehet megadni, hogy az adott eszköz pontosan hol is található (helyi eszköz vagy távoli elérésű eszköz), valamint tartalmazza a berendezés által elfogadott parancsok listáját, paramétereit, valamint a berendezés vezérléséhez szükséges környezeti feltételeket (DLL, shared library).

A szerver képes beolvasni egy szabványos XML dokumentumot, ha az rá vonatkozó beállításokat tartalmaz vagyis a dokumentum root node -jának a neve „nmcsettings”.

3.1.2.1 Beállítási fájlok szerkezete

A beállítási fájlok szabványos XML fájlok, amelyek a következő alrészeket tartalmazhatják:

• Felhasználóra vonatkozó információk

• Szerver beállításaira vonatkozó információk

• A kezelt eszközökre vonatkozó információk

• További beállítási fájlokra mutató elérési utak amelyeket be kell tölteni.

3.1.2.1.1 Felhasználói információk

A felhasználói beállítások az XML dokumentumban XML node -ként jelennek meg, amelynek a következő részei vannak:

• Felhasználónév: Nem lehet két azonos felhasználónevű, de különböző felhasználó a beolvasott XML beállítási fájlokban.

•Jelszó: A felhasználó jelszava mely mellet tulajdonságként megjelenik a jelszó tárolására vonatkozó információk is, amit kötelező megadni a fájlban.

•Felhasználó azonosító

•Csoport azonosító (a felhasználó fő csoportja)

•Felhasználó egyéb adatai, mint a teljes neve ,címe …

Egy felhasználó bejegyzése a beállítási fájlokban a következőképpen néz ki:

```
<user>
        <uname>user</uname>
        <groupid>vendeg</groupid>
        <password>
```

```
            <type>text</type>
            <pwd>rossz</pwd>
        </password>
</user>
```

A rendszer a felhasználókat adatbázisból is vehetné, de ez a lehetőség nem rész a jelenlegi verziónak, ez irányban is tovább lehet fejleszteni a rendszer.

3.1.2.1.2 Csoportokra vonatkozó információk

Minden felhasználó tagja kell hogy legyen egy csoportnak, amelyhez egy adott berendezésre vagy berendezésekre szóló jogok vannak kiosztva. Ezek a jogok a következők lehetnek:

• Olvasási jog: a felhasználó csak olvashatja keletkezett adatokat és beállításokat, de nem módosíthatja őket.

• Beállítási jog : Az egyes eszközök paramétereit megváltoztathatja.

• Admin jog: felhasználói jogokat oszthat ki, beállításokat módosíthat.

Egy csoportnak csak egy -féle joga lehet.

Minden csoportnak van egy egyedi azonosítója. Nem lehet kétszer ugyanaz a nevű vagy id -jű csoportot definiálni.

Egy csoportra vonatkozó XML - node -nak a következő részei lehetnek:

- gname: a csoport nevét tartalmaza
- groupid: a csoport egyedi azonosítója
- right (jog): a csoporthoz tartozó felhasználók jogát tartja nyilván, ezek lehetnek „read”, ”settings”, „admin” jogok.
- Userids: a csoporthoz tartozó felhasználók egyedi azonosítójának felsorolása.

Egy csoportra vonatkozó beállítás a következőképpen jelenhet meg a beállítási fájlokban:

```
<group>
        <gname>
                csoportnev
        </gname>
        <grupid>
                10007
        </groupid>
        <right>
                admin
        </right>
        <userids>
                <userid>1555</userids>
                <userid>1556</userids>
        </userids>
</group>
```

3.1.2.1.3 Szerverbeállításai

A szerver beállításaiból csak egyetlen egy szerepelhet minden rendszerhez, ez a beállítás a szerver által használt portot tartalmazza jelenleg. A jövőben esetleg kiegészülhet még egyéb opciókkal is, az aktuális verzióban csak ez szerepelhet. Ez a port alapértelmezés szerint a 3000-es számú. Ehhez a porthoz kapcsolódhatnak a kliensek a hálózaton keresztül.

Egy szerverbeállítás a következőképpen jelenhet meg a beállítási XML fájlban:

```
<nmcserver>
        <port>
                3000
        </port>
</nmcserver>
```

3.1.2.1.4 Eszközök beállításai

Ez az XML node -tartalmazza az eszközökre vonatkozó paramétereket. A kezelőfolyamat típusát, elérési útját és paramétereit. Továbbá tartalmaz egy eszköz azonosítót is amelynek minden kezelt eszköznél egyedinek kell lennie. A program által felügyelt műszerek nem feltétlenül abban a gépben találhatóak ahol a szerver alkalmazás fut, hanem egy távoli gépen, vagy esetleg egy másik szerver alkalmazás felügyelete alá tartozik és hálózaton keresztül érhető el. Ekkor azonban szerepelnie kell itt az adott szerverhez tartozó elérési utakkal és a hozzáféréshez szükséges információknak is, mint a felhasználónév.

A beállítások között szerepel az, hogy az adott eszköz mely csoportokhoz tartozik és így az egyes csoportoknak különböző jogosultságaik vannak.

Az eszközbeállításait tartalmazó XML node a következőképpen jeleni meg a beállítási fájlokban:

```
<nmcdevice>
    <type>
        local
    </type>
    <name>
    detector
    </name>
    <deviceid>
        5221562
    </deviceid>
    <driver>
        <path>
            /usr/lib/shard_lib.so
        </path>
        <arg name=”system call 1”>
```

```
                <call>1</call>
                <param>1</param>
                <param>2</param>
            </arg>
            <arg name="syscall 2">
                <call>1</call>
                <param>3</param>
                <param>1</param>
            </arg>
 </driver>
</nmcdevice>
```

3.1.2.2 Eszközmeghajtó programok illesztése a rendszerhez

Az eszközmeghajtók kezelése és elérése a legegyszerűbb egy osztott programkönyvtár (Unix, Linux) vagy MS windows alatt DLL (Dinamic Linked Libary) így egy egységes felület alakítható ki amin keresztül az alkalmazások kommunikálni tudnak az eszközökkel és felügyelni is tudják őket. Ezeket a programkönyvtárakat is a wxWidgets -el érdemes elkészíteni legalábbis a felület illesztését a keretrendszerhez, ezen kívül még egyéb plusz funkció is belekerülhetnek nem csak az illesztések és a driverek.

Az illesztett rendszerhívásnak (programkönyvtár hívás) esetén egy olyan wxPanel objektumot kell visszaadnia amin keresztül be lehet állítani a készülék egyes jellemzőit és illeszteni lehet, a grafikus felület egy wxPanel felületére. Így elérhetjük azt, hogy a mérőegységet kezelő egység és a szerver egymástól függetlenül működjön, mivel a GUI ezen a részén történő változtatásokat csak továbbítja és nem ő hajtja végre, hanem a konkrét kezelőprogram.

Az eszköz kezelőprogram mint külön folyamat fut a rendszeren és csak egyetlen eszköz teljes körű kezelését teszi lehetővé. Az esetlegesen előforduló hibákat, jelzéseket,

figyelmeztetéseket és adatokat továbbítja a hívó folyamat felé, ez esetleges sürgős beavatkozást igénylő események kezelés a meghajtóprogram hibakezelőrészében történik, mivel csak itt lehet a leghatékonyabban kezelni a felmerülő kritikus hibákat. A hibákat jelezni kell a hívó folyamat felé is, hogy a hibakezelés itt is megtörténjen. A meghajtóprogram alapértelmezett hibakezelésének dinamikusan felülbírálhatónak kell lennie és a hívó folyamat számára hozzáférhetőnek. Azaz a kezelőprogram mint egy beépülő modul úgy kapcsolódik a keretrendszerhez és annak felügyelete alá tartozik.

Ezzel a beépülő modul módszerrel egy szerűvé válik az új eszközök csatolása, hiszen egy újabb beépülő modul hozzáadása egyszerűen elvégezhető az alap menü segítségével.

A keretrendszerbe léteznie kell egy olyan lehetőségnek is, amely segítségével létrehozhatunk saját alap modulokat is amiket később kibővíthetünk és szerkeszthetünk is.

3.1.2.3 Tervezési minták

Ahhoz hogy újrahasznosíthassuk a tervet, muszáj rögzítenünk a döntéseket, az alternatívákat és a döntések következményeit is. A következő szempontok szerint elemzem a felhasznált tervezési mintákat:

- **A minta neve és besorolása**: A minta neve a minta lényegét közvetíti rövid formában. A jó név létfontosságú, mivel a tervezési szókincs részévé válik.
- **Cél:**Rövid leírás, amely következő kérdésekkel foglalkozik: Mit csinál az adott tervezési minta? Mi az értelme és a célja? Milyen sajátos tervezési problémára ad választ?
- **Feladat:** Forgatókönyv, amely bemutatja a tervezési problémát és azt hogy az osztályok és objektumok hogyan oldják azt meg. A forgatókönyv segít a minta következő, elvontabb leírásának megértésében.
- **Alkalmazhatóság**: Melyek azok a helyzetek , ahol az adott tervezési minta

használható? Melyek azok a rossz tervek, amelyek leváltását a minta megcélozza?

• **Szerkezet**: A minta osztályainak grafikus szerkezetének szemléltetése, az ODT-n (Object Modeling Technique) alapuló jelölések használatával. Együttműködési diagramokat (interakciódiagramok) is használok, hogy szemléltessem a kérelmek és együttműködések sorozatát az objektumok között.

• **Résztvevők**: Az osztályok, illetve objektumok, amelyek részt vesznek a tervezési mintákban, valamint azok feladatai.

• **Együttműködés:** Hogyan működnek együtt az objektumok, hogy végrehajtsák feladataikat?

• **Következmények**: Hogyan támogatja az adott minta a kívánt célokat? Mik a minta használatának előnyei és hátrányai? A rendszer mely szerkezeti elemeit változtathatjuk szabadon az adott mintát használva?

• **Megvalósítás**: Milyen buktatókra kell ügyelni, milyen módszereket érdemes használni a mint megvalósításakor? Vannak nyelvi sajátosságok?

• **Példa**: Programkód töredék, amely bemutatja, hogy hogyan valósítható meg a minta C++ nyelven.

• **Kapcsolódó minták**: Mely tervezési minták kapcsolódnak szorosan az adott mintához? Mik a legfontosabb különbségek? Milyen más mintákkal együtt célszerű használni az adott mintát?

A tervezési minták által megengedett változtatható elemek		
Cél	**Tervezési minta neve**	**Változtatható elemek**
Létrehozási	Absztrakt gyár	Leszármaztatott objektumok családja
	Építő	Hogyan készül az összetett objektum
	Gyártófüggvény	Egy példányobjektum alosztálya
	Prototípus	Egy példányobjektum osztálya
	Egyke	Egy osztály egyetlen példánya
Szerkezeti	Illesztő	Felület egy objektumhoz

	Híd	Egy objektum megvalósítása
	Összetétel	Egy objektum szerkezete és összetétele
	Díszítő	Egy objektum kötelességei leszármaztatás nélkül
	Homlokzat	Felület egy alrendszerhez
	Pehelysúly	Objektum tárolásának költsége
	Helyettes	Hogyan érjük el az objektumot; az objektum helyzete
Viselkedési	Felelősséglánc	Az objektum, ami a kéréseket teljesíti
	Parancs	Mikor és hogyan teljesül egy kérelem
	Értelmező	Egy nyelv szabályai és értelmezése
	Bejáró	Hogyan érjük el és járjuk be egy aggregátum elemeit
	Közvetítő	Mely objektumok hatnak egymásra, és hogyan
	Emlékeztető	Milyen privát információk tárolódnak az objektumon kívül, és mikor
	Megfigyelő	Sok másik objektumtól függő objektum; hogyan maradnak a függő objektumok naprakészek
	Állapot	Egy objektum állapotai
	Stratégia	Egy algoritmus
	Sablonfüggvény	Egy algoritmus lépései
	Látogató	Olyan műveletek; amelyek alkalmazhatóak objektum(ok)ra az osztályok megváltoztatása nélkül

[PMUEOP]

3.1.2.3.1 Hogyan oldják meg a tervezési minták a tervezési problémákat?

A tervezési minták számos olyan mindennapi problémákat oldanak meg, amelyekkel a tervezők szembetalálkoznak, méghozzá sokféleképpen. Következőkben felsorolok párat ezen problémák közül és leírom hogyan oldják meg ezen problémákat a tervezési minták.

Megfelelő objektumok keresése: Az objektum magában foglalja az adatokat és az azon végezhető műveleteket, amelyeket metódusoknak hívunk. Egy objektum akkor hajt végre egy műveletet ha arra kérés érkezik hozzá egy klienstől és csakis akkor. Belő adatait csak a metódusai segítségével tudjuk megváltoztatni így egy felületet kínálnak fel. Csakis ezen a felületen keresztül tudjuk módosítani az objektum belső adatait. Elrejtik az adatokat a külvilág elől, így valósítják meg az adatrejtést.

Az objektumközpontú tervezés nehézsége abban rejlik hogy a rendszert hogyan bontsuk fel együttműködő, egymással kapcsolatban lévő objektumok összességére. A feladat azért nehéz, mert sok tényezőt kell számításba venni: az egységbezárást (betokozást), a „finomság” (részlettesség), függőségek, rugalmasság, teljesítmény, továbbfejleszthetőség, újrahasznosíthatóság és így tovább. Ezek mind befolyásolják a rendszer felbontását sokszor egymásnak ellentmondva.

Talán az egyik legfontosabb kérdés a tervezés során, hogy mi is legyen egy objektum? A tervezési minták erre a kérdésre is választ adnak.

A homlokzat minta azt írja le, hogyan ábrázolhatunk egész alrendszereket objektumként, míg a pehelysúly minta azt, hogy hogyan támogassunk nagyon sok, „nagy finomságú” (kis részleteket leíró) objektumokat. Más tervezési minták különböző módokon írnak le nagy objektumok kisebb objektumokra történő lebontását. Az elvont gyár és építő minták olyan objektumokat eredményeznek amelyeknek egyetlen feladata más objektumok létrehozása, a látogató és parancs minták pedig olyanokat, amelyek egyetlen célja kérelmeket intézzenek egy objektumhoz vagy objektum csoportokhoz.

Minden objektum által objektum által leírt művelet megadja a művelet nevét, az objektumokat amelyeket paraméterül kap, és a visszatérési értékét. Ezt hívjuk a művelet

aláírásának vagy „szignatúrájának”. Az objektum műveletei által meghatározott összes ilyen aláírás halmazát az objektum felületének nevezünk. A felület jellemzi az objektumnak küldhető kérelmek teljes halmazát. Bármely, az objektum felületében lévő aláírásnak megfelelő kérelem elküldhető az objektumnak.

Az osztály adja meg az objektum belső adatainak ábrázolását, illetve a műveleteket amelyeket az objektum végre tud hajtani. Ez a következőképpen jelenik meg UML diagram formában:

Ahol az attribútumok szerepelenek ott a láthatóságra vonatkozó ionformációk is megjelennek, speciálisan C++ nyelv esetén (+ public, # protected, - privát), ugyanezek a láthatósági jelölése k igazak a metódusokra is.

Az objektumok egy osztály példányosításával jönnek létre. Az objektum az osztály egy példánya. Az osztály példányosításának folyamata során az osztály adattagjaihoz hozzárendelődik a tárhely és egy kezdő érték.

A már létező osztályból örökléssel hozhatunk létre új osztályokat. Ha egy alosztály örököl egy szülőosztálytól, akkor a szülőosztály által leírt minden adattagot és metódust örököl. Az alosztályok felülbírálhatják, vagy megváltoztathatják a szülő osztály leírt viselkedésmódot. Ezzel megkapják az esélyt arra, hogy a hozzájuk érkező kérelmeket ne a szülőosztályuk kezelje, hanem saját maguk.

Az absztrakt osztályok lényege, hogy általános felületet ír le az alosztályai számára.

3.1.2.3.2 Megvalósítás helyett felületre programozás

Az osztályöröklés csak egy eszköz arra, hogy az alkalmazás szolgáltatásait a szülő osztály szolgáltatásaival bővítsük. Segítségével gyorsan hozhatunk létre új objektumokat a régiek alapján. Különösebb erőfeszítés nélkül kaphatunk új megvalósításokat, egyszerűen

örökölve a létező osztályokból, amire szükségünk van.

Mindazonáltal a megvalósítás újrahasznosítása még nem minden. Az öröklés azon tulajdonsága, hogy egyező felületű objektumok családját képes meghatározni szintén fontos, mert ezen alapul a többalakúság. Ha figyelmesen használjuk az öröklést, akkor minden, adott absztrakt osztályból származó osztály osztozik majd a szülő felületén, ebből következik, hogy egy alosztály a műveletekhez csak hozzáadhat, vagy felüldefiniálhatja azokat, de a szülőosztály metódusait nem rejtheti el. Így minden leszármaztatott osztály tud válaszolni az absztrakt osztály felületében szereplő kérelmekre.

Két előny is származik abból, hogy ha az objektumokat kizárólag az absztrakt osztályban meghatározott felület szintjén kezeljük:

1.Az ügyfelek egészen addig nem veszik figyelembe a felhasznált objektum típusát amíg az objektumok felülete az ügyfelek által vártnak megfelel.

2.Az ügyfelek nem veszik figyelembe az ezeket az objektumokat megvalósító osztályokat. Csak a felületet meghatározó absztrakt osztály(oka)t ismerik.

Így annyira le tudjuk csökkenteni az alrendszerek közötti megvalósítási függőséget, hogy az az újrahasznosítható objektumközpontú terv első alapelvéhez vezet:

Programozzunk felületre a megvalósítás helyett.

Ne a konkrét osztályok példányaiként vezessük be változóinkat, hanem csak az elvont osztályban szereplő felületet bővítsük ki. Persze valahol a rendszerünkben példányosítanunk kell a konkrét osztályokat (vagyis meg kell adnunk a tényleges megvalósítást), és a létrehozási mintákat (Absztrakt gyár, Építő, Gyártófüggvény, Prototípus, Egyke) éppen ezt teszik lehetővé. Az objektumok létrehozás- folyamatának elvonatkoztatásával ezek a tervezési minták különböző lehetőségeket adnak arra, hogy egy felületet a megvalósításával anélkül kapcsolhassunk össze, hogy egyetlen osztályt is példányosítanánk és hogy az implementációról nincs tudomásunk. A létrehozási minták biztosítják, hogy a rendszer felület, és ne implementáció-központú legyen.

3.1.2.3.3 Újrahasznosítási szerkezetek használata

A legtöbb ember megérti az objektumok, felületek, osztályok és az öröklés fogalmát. Az igazi nehézség abban mutatkozik meg, hogy alkalmazásukkal rugalmas, újrafelhasználható programokat és rendszereket építsünk fel, és a tervezési minták pontosan ebben nyújthatnak nagy segítséget nekünk.

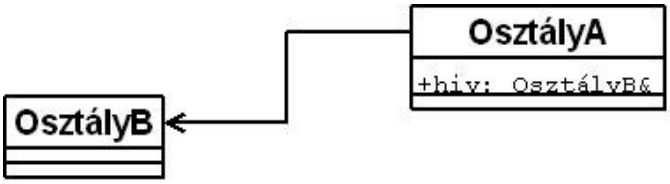
Az objektum-összetétel dinamikusan futási időben történik, olyan objektumokon keresztül amelyek hivatkozásokat kapnak más objektumokra. Az összetételhez szükséges, hogy az objektumok figyelembe vegyék egymás felületlét, amihez viszont pontosan és figyelmesen megtervezett felületekre van szükség, amely lehetővé teszi, hogy az objektumot sokk másikkal együtt tudjuk használni. Ennek a módszernek nagy előnye, hogy az objektumokat csak a felületükön keresztül érjük el és így nem sértjük meg az egységbezárás elvét sem. Bármely objektumot lecserélhetünk egy másikra futási időben, azzal a feltétellel hogy a típusuk megegyezik. Továbbá mivel az objektumok megvalósítása az objektum felületek segítségével épül fel, sokkal kevesebb lesz a megvalósítási függőség. Az objektum-összetételnek még egy nagy hatása van a rendszer szerkezetére vonatkozóan: az örökléssel szemben segít az osztályok egységbezárásán, és abban hogy csak a kitűzött feladatra koncentráljanak. Az osztályok és osztályhierarchiák kicsik maradnak, és nem nőnek akkorává hogy ne tudjuk őket kezelni. Egyik fontos következménye, hogy az objektumok száma a rendszerben jócskán megnő és ezeket mind tárolni kell és gondoskodni kell a megfelelő kezelésükről.

Ez vezet el minket az objektumközpontú tervezés második alapelvéhez:
Használjunk objektum-összetételt osztályöröklés helyett, amikor csak lehetséges.

3.1.2.3.4 Képviselet

A képviselet (delegáció) az összetétel olyan erejű újrahasznosítási módszerré tétele, mint az öröklés. A kérelmek kezelésekor ekkor két objektum vesz részt : Egy fogadó objektum, és a képviselője, amelyre a fogadó műveleteket ruház át. Ez az alosztályok szülőkhöz intézett kérelmeinek átirányításához hasonló. De az öröklésnél az örökölt műveletek mindig utalhatnak a fogadó objektumra, amit a C++ programozási nyelvben a this valósít meg. Hogy ezt megvalósíthassuk a fogadó objektum átad magáról egy referenciát a képviselő objektumnak, hogy az átruházott művelet a fogadóra utalhasson.

Például a tervezett keretrendszerben a Hálózati beállításokat kezelő osztály megkap egy referenciát a beállításokat menedzselő osztályra, hogy közvetlenül elérhetőek legyenek számára bizonyos adatok, amelyek fontosak a helyes működéshez. Ez a diagrammon egy egyszerű nyílban végződő vonalként jelenik meg, amely azt mutatja hogy egy osztály egy másik osztály egy példányára hivatkozik.



3. ábra

A képviselet legfőbb előnye, hogy segítségével könnyen alakíthatunk ki viselkedésmódokat futási időben, és megváltoztathatjuk összetételük módját.

A képviseletnek ugyanaz a hátránya, mint általában a többi objektum-összetételi rugalmasságot növelő eljárásnak: a dinamikus, erősen paraméterezett programokat nehezebb megérteni mint a statikusabbakat, mert futási időben történik minden lényeges lépés, ami első ránézésre a kódból közvetlenül nem derül ki.

Több tervezési minta is közvetlenül felhasználja ezt a módszert, és közvetlenül függnek tőle. Más tervezési minták kevésbé használják a képviseletet. A közvetítő minta bevezet egy közvetítő objektumot, ami a hozzá érkező kérelmeket továbbítja a megfelelő objektum felé, vagy akár hivatkozást is küldhet magáról, vagyis valódi képviseletről van szó.

3.1.2.3.5 Öröklés és paraméterezett típusok

Ilyen típusú újrahasznosításra ad lehetőséget például C++ -ban az osztálysablonok használata, amelynél elég csak az osztály felépítésének a vázát megadni konkrét típusok nélkül és anélkül hogy minden általa felhasznált típust megadnánk vagy minden egyes típusra külön külön implementálnánk ugyanazt a viselkedésmódot. Talán az egyik legismertebb ilyen C++ -ban a standard tárolók.

A paraméterezett típusokkal egy harmadik módszert kapunk az öröklés és az összetétel mellé, amellyel objektumközpontú rendszerek viselkedését építhetjük fel és írhatjuk le. Sokféle tervet valósíthatunk meg e három módszer alkalmazásával.

De ennek a módszernek is megvannak a maga hátulütői is: C++ esetében például az egyik legnagyobb ilyen probléma éppen a hordozhatóság kapcsán merül fel. Ugyanis nem mindegyik fordító értelmezi helyesen a sablonokat, egyes régebbi fordítók nem is ismerik és ha megpróbálnánk lefordítani az addig tökéletesen működő kódot, csak fordítási hibák tömkelegét kapjuk. Ezért oda kell figyelni, arra is, hogy az adott C++ implementáció pontosan melyik nyelvi szabványt is támogatja. Itt felmerülnek már az implementációfüggő kérdések is, és ezek sokszor olyan rejtett nehezen felderíthető hibákhoz vezetnek, amelyek csak bizonyos speciális esetben jelentkeznek.

Az általam használ wxWidgets könyvtárban többek között ezen okok miatt nem javasolják ezen absztrakciós eszköz használatát, mivel nagyban csökkenti a forráskód hordozhatóságát is. Erre a problémára létezik egy megkerülő megoldás, amely C++ makróhívások segítségével valósul meg. Igaz hogy ez nem egy tökéletes megoldás, de igen jól és hatékonyan használható. A legfontosabb tulajdonságai közé tartozik, hogy igen részletesen és jól dokumentált, és ez óriási segítséget jelent mind a tervezésnél mind az implementációnál.

3.1.2.3.6 Változásra tervezve

Az újrahasznosíthatóság kulcsa a az új igények és a létező igények változásának megérzésében és megbecsülésében rejlik, és abban, hogy úgy tervezzük meg a rendszerünket, hogy ennek megfelelően fejlődhessen.

Ahhoz, hogy olyan rendszert tervezzünk, ami a változásoknak is megfelel, figyelembe kell vennünk, hogy a rendszernek milyen változásokon kell átesnie élete során. Ha ezeket nem vesszük figyelembe akkor azt kockáztatjuk, hogy esetleg az egész keretrendszert az alapjaitól újra kell terveznünk, hogy megfeleljen az újonnan felmerült igényeknek. Ezen változások között előfordulhatnak osztályok újbóli meghatározása és újra implementálása, az ügyfél változása és újra tesztelése. Ez az újratervezés a keretrendszer számos részét érintheti, és a váratlan változások kivétel nélkül költségesek, mind a ráfordított idő mind az emberi erőforrás tekintetében.

A tervezési minták alkalmazásával elkerülhető, segítségükkel biztosítható, hogy a rendszer meghatározott módokon módosítható legyen az egyes változástípusokkal szemben.

A leggyakoribb okok amiért akár egy teljes rendszert újra kell gondolni és tervezni:

1.Objektum létrehozása konkrét osztály megadásával

2.Konkrét metódusokra és műveletekre támaszkodás

3.Függőség a hardver- és programkörnyezettől

4.Függőség az objektumok ábrázolásától és megvalósításától

5.Algoritmikus függőségek

6.Szoros csatolás

7.A működés kibővítése alosztályokkal

8.Osztályok kényelmetlen módosítása

3.1.2.3.7 Hogyan válasszunk tervezési mintát?

A tervezési minták közül nem könnyű, nem könnyű kiválasztani azt amelyikre éppen szükségünk van az adott probléma megoldásához. Ezért választáskor az alábbi fontos szempontokat kell figyelembe vennünk és ezek alapján hozzuk csak meg a megfelelő döntést, ezeken kívül még lehetnek egyéb szempontok is de ezek talán a legfontosabbak.

Először is gondoljuk át az alábbi szempontok szerint a kitűzött feladat részeit a következő

- A tervezési minták hogyan oldják meg a tervezési problémánkat.
- Nézzük meg az erre a részre vonatkozó részeket
- Tanulmányozzuk a hasonló célú mintákat
- Vizsgáljuk meg az újratervezés okait
- Gondoljuk át, mit tegyünk változtathatóvá a rendszerünkben.

Ha már kiválasztottuk a tervezési mintát, akkor azt kell először is tisztáznunk, hogy azt hogyan is akarjuk használni. Talán ha a következő pontokon végigmegyünk, akkor világossá válok hogyan használhatnánk fel az adott mintát a leghatékonyabban a tervezés során:

1. Olvassuk át a mintát, hogy legyen róla fogalmunk és olvassuk el figyelmesen az alkalmazhatóságra és a követelményekre vonatkozó részeket, hogy biztosak lehessünk benne, hogy ez a megfelelő minta a problémánk megoldására.
2. Térjünk vissza a Szerkezet, a Résztvevők és az Együttműködés részekre és győződjünk meg, hogy valóban értjük a minta lényegét és a benne szereplő osztályok és objektumok közötti kapcsolatokat.
3. Válaszunk olyan neveket a mintában szereplő elemeknek, amelyeknek értelmesek lesznek az adott környezetben.
4. Keresünk példákat, hogy lássuk hogyan működik a minta élesben.
5. Határozzuk meg az osztályainkat és azok felületeit, öröklési viszonyaikat,

példányváltozóikat, adat- és objektum hivatkozásaikat. Határozzuk meg azokat az osztályokat, amelyekre hatással lesz az adott tervezési minta és ennek megfelelően módosítsuk azokat.

6.Adjunk az alkalmazásra jellemző neveket a mintában szereplő műveleteknek úgy, hogy azok könnyen felismerhetőek és értelmesek legyenek. Jellemeze a művelet tulajdonságait és konkrét feladatát.

7.A megvalósítás

8.A tesztelés talán az egyik legfontosabb, mert itt még egyszerű a kisebb hibákat megtalálni.

Ezek csak javaslatok, de ez a sorrend a legmegfelelőbb.

3.1.2.4 A wxWidgets és a tervezési minták

3.1.2.4.1 Eseménykezelés

Ez az egyik legfontosabb és legtöbbet használt, része ennek az elemkészletnek. Az eseményeket a wxEvent osztály és leszármazottjai reprezentálják, ezen osztály objektumai csak akkor kerülnek példányosításra, ha valamilyen esemény bekövetkezik vagy direkt kiváltunk egyet. Az esemény objektum információt tartalmaz arról, hogy az adott esemény hol és milyen körülmények között váltódott ki, ezek az információk lekérdezhetőek az objektumtól. Ezen információk a későbbiekben nagy hasznukra lesznek az egyes események lekezelésében.

Az események kezelő osztályok a wxEventHandler osztály leszármazottjai, amelyek az események kezelésére egy eseményeket és azok kezelésére szolgáló metódusok nevét tartalmazó táblát tartanak nyilván. Ezzel a táblázattal lehet összekapcsolni az eseményeket és az őket lekezelni képes objektum metódusait. Az összerendelés csak az implementációs részben történhet meg, kivéve az alapértelmezések esetén. Ennek az összekapcsolásnak két fajtája van, ezek a kódban is különbözőképpen jelennek meg és a működésük is más, ezek a

következők:

1.**Statikus eseménykezelés**:

Ekkor az eseményeket kezelő metódusok a már a fordítási időben hozzárendelődik a eseményekhez és már ekkor meg lehet mondani milyen eseményeket kezelését vesszük át egy objektumtól. A kód implementációs részében a következőképpen jelenik meg:

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
  EVT_MENU(wxID_EXIT, MyFrame::OnExit)
  EVT_SIZE(MyFrame::OnSize)
  EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
END_EVENT_TABLE()
```

Itt a példában egy wxForm -eseményeihez rendeljük hozzá a kezelő metódusokat. Jelen esetben a wxID_EXIT eseményhez rendeltük hozzá a MyFrame::OnExit metódusát, amely a kilépés menüpont választása esetén váltódik ki. Hasonlóan a rendeltük hozzá az átméretezés eseményét és a gomb lenyomását.

2.**Dinamikus eseménykezelés**:

Ekkor az eseményeket lekezelő objektumok futási időben dinamikusan rendelődnek hozzá egy-egy objektumhoz amelynek az eseményeit le akarjuk kezelni. Ezek a kódban a következőképpen jelennek meg az implementáció során:

```
main->Connect(NMCMainFrame::ON_EXIT,  wxID_ANY,
                      wxEVT_COMMAND_MENU_SELECTED,
                      wxCommandEventHandler(NMCMainFrame::OnQuit) );

main->Connect(NMCMainFrame::ON_HELP_ABOUT,  wxID_ANY,
                      wxEVT_COMMAND_MENU_SELECTED,
                      wxCommandEventHandler(NMCMainFrame::OnAbout) );

main->Connect(NMCMainFrame::ON_DEVICE_NEW_LOCAL,  wxID_ANY,
                      wxEVT_COMMAND_MENU_SELECTED,
                      wxCommandEventHandler(NMCMainFrame::OnNewLocalDevice) );

main->Connect(NMCMainFrame::ON_DEVICE_NEW_NETWORK,  wxID_ANY,
                      wxEVT_COMMAND_MENU_SELECTED,
                      wxCommandEventHandler(NMCMainFrame::OnNewNetworkDevice) );

main->Connect(NMCMainFrame::ON_SERVER_START,  wxID_ANY,
                      wxEVT_COMMAND_MENU_SELECTED,
                      wxCommandEventHandler(NMCMainFrame::OnStartServer) );
main->Connect(NMCMainFrame::ON_SERVER_STOP,  wxID_ANY,
                      wxEVT_COMMAND_MENU_SELECTED,
                      wxCommandEventHandler(NMCMainFrame::OnStopServer) );
```

A kódrészletben egy wxFrame objektumhoz rendeljük hozzá az eseménykezelő metódusokat amelyek jelen esetben a saját osztály részei, ezeket lehetett volna statikusan is hozzárendelni, de ezen részei a programnak azok amelyek a legdinamikusabban változnak a program futása során. Mivel a keretrendszerben éppen az ilyen rugalmasságra építünk, azaz futásközben lecserélhetünk akár egy teljes esemény kezelő objektumot is egy másikra anélkül hogy a programot újra kellene írnunk és fordítanunk. Ezen lehetőség kihasználásával sok időt megspórolhatunk, de nagyon alapos tervezést igényel és nagy odafigyelést.

A wxWindws eseménykezelő része makróhívások segítségével elrejti az egyes platformok eseménykezelése közötti különbséget, erre a célra több különböző makróegyüttes is a rendelkezésünkre áll ezen makróknak meg kell adni paramétereket. A makrók közül nem mindegyik szerepelhet bárhol a kódban. Vannak olyan makrók amelyek csak az adott osztály definíciójában szerepelhetnek a megfelelő paraméterekkel és sehol máshol, mert különben fordítási hibát eredményeznek. Ilyen makró például a DECLARE_EVENT_TABLE, amely csak az osztály deklarációs részének a legvégén szerepelhet. Hasonló a helyzet a BEGIN_EVENT_TABE makrókkal, amelyek csak az implementációban szerepelhetnek és egy adott osztályhoz hozzák létre a z események kezeléséhez szükséges esemény táblát.

3.1.2.4.2 Gyermekosztály létrehozása wxWindges -el

Általában a szabványos C++ öröklés jellemző, azaz egy osztálynak akárhány szülőosztálya lehet, szabályos deklaráció. Egy új vezérlőelemet létrehozhatunk egy már korábban létező vezérlőelemből egyszerű leszármaztatással és kiegészítve ezt az elemkészlet sajátosságaival.

```
class myStaticText : public wxControl
{
public:
      DECLARE_DYNAMIC_CLASS(myStaticText);

      myStaticText () {}
      myStaticText (wxWindow* parent, wxWindowID id, const wxString& txt)
        : wxControl (parent, id) { SetLabel (txt); }

      wxSize DoGetBestSize() const;
      void OnPaint(wxPaintEvent& event);
      void SetLabel( const wxString& label );

      DECLARE_EVENT_TABLE();
};

IMPLEMENT_DYNAMIC_CLASS(myStaticText,wxControl);

BEGIN_EVENT_TABLE(myStaticText, wxControl)
      EVT_PAINT(myStaticText::OnPaint)
END_EVENT_TABLE()
```

4. ábra Gyermekosztály deklarációja.

Itt is megjelennek a makróhívások amivel elrejtjük a platformok közötti különbségeket. Egyik ilyen makró a DECLARE_DYNAMIC_CLASS a futásidőben történő típusinformációk kezeléséhez nyújt segítséget és eszközöket, ezt a makrót a deklaráció. Az öröklés nagyon fontos szerepet játszik a tervezési minták kialakításában, de ugyan ilyen fontos a helyettesíthetőség is abban az esetben, ha egy szülő osztályt valamelyik gyermekével akarunk helyettesíteni.

3.1.3 Az NMC Szerver alkalmazás tervezése

A specifikációs részben megadtuk, hogy milyen feladatokat és kötelességeket kell ellátnia a tervezett keretrendszernek, de tervezés közben esetleg belefuthatunk olyan részproblémákba ami a specifikáció kibővítését és esetleges újragondolását vonja maga után ezt sohasem szabad figyelmen kívül hagyni és dokumentálni. Valamit meg kel határoznunk mit és hogyan szeretnénk megtervezni és esetleg megvalósítani.

Az NMC Szerver alkalmazást az alábbi szempontok szerint vizsgáltam a specifikációban megfogalmazott feladatok alapján:

1.Beállítási adatok szerkezete és tárolásának módja

2.A felhasználói felület felépítése

3.Mérőeszközök felügyeletét ellátó modulok

4.Mérőeeszközöktől érkező adatok tárolása

5.Adatok továbbítása

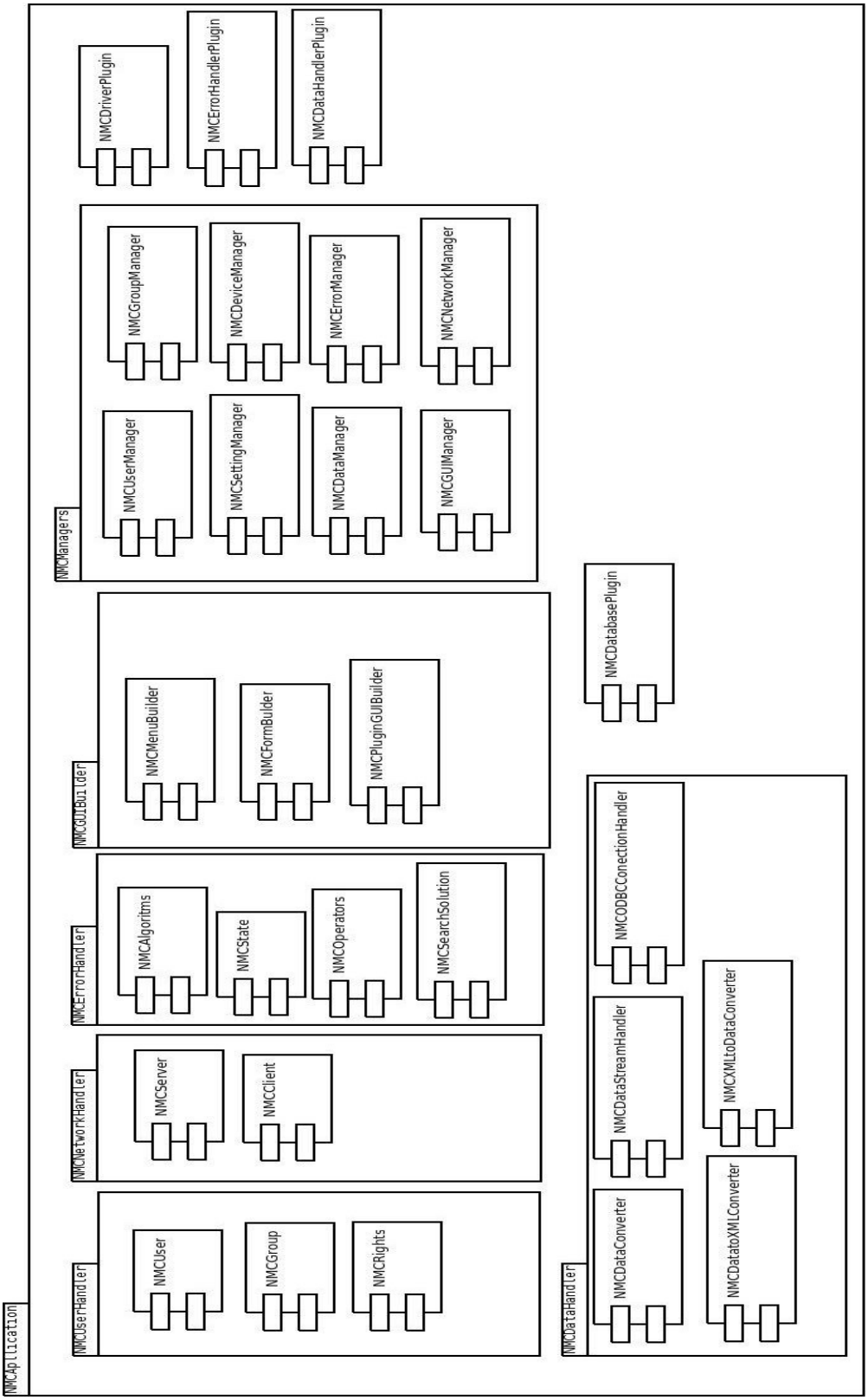
6.Felhasználói és csoport jogok kezelése

7.Felhasználói kérelmek teljesítése, beavatkozási lehetőségek

8.Hálózati erőforrás megosztás

9.Hibakezelés és felismerés esetleges súlyos hibák kezelése

A következő ábra a rendszer a keretrendszer főbb komponenseit mutatja be:


NMCApplication
NMCUserHandler
NMCUser
NMCGroup
NMCRights
NMCNetworkHandler
NMCServer
NMCClient
NMCErrorHandler
NMCAlgoritms
NMCState
NMCOperators
NMCSearchSolution
NMCGUIBuilder
NMCMenuBuilder
NMCFormBuilder
NMCPluginGUIBuilder
NMCManagers
NMCUserManager
NMCSettingManager
NMCDataManager
NMCGUIManager
NMCGroupManager
NMCDeviceManager
NMCErrorManager
NMCNetworkManager
NMCDriverPlugin
NMCErrorHandlerPlugin
NMCDataHandlerPlugin
NMCDatabasePlugin
NMCDataHandler
NMCDataConverter
NMCDataStreamHandler
NMCDataXMLConverter
NMCXMLtoDataConverter
NMCODBCConectionHandler

5. ábra

3.1.3.1.1 Beállítási adatok szerkezete és tárolásának módja

Az első kérdés amit fel kel tennünk milyen formában jelenek meg a beállítások? A legkézenfekvőbb az ha valamilyen szabványos formát választunk a beállítási adataink tárolására, mert ezzel is sok időt megspórolhatunk magunknak, hiszen nem nekünk kell újból feltalálni a spanyol viaszt és nem kell leírni minden egyes részletet ami csak felmerül elég csak az adott szabványra hivatkozni.

Továbbá érdemes azt is átgondolni menyire kell hogy rugalmas legyen ez az adat tárolási módszer, hiszen ha biztosan nem változik és mindig ugyanaz marad akkor felesleges egy nagy és bonyolult adatstruktúrát és tárolási módot alkalmazni. Elég a legegyszerűbb forma is például egy egyszerű bináris fájl, de ez idővel karbantarthatatlanná válhat és a mérete is megnőhet, nem is beszélve arról, hogy ezeknek a beállításoknak hordozhatónak kell lenniük. Ezért nem alkalmas csak egy egyszerű bináris fájl a beállítások tárolására. Többek között ezért is eset a választásom az XML nyelvere. Az XML nyelven írt dokumentumoknak megvannak azok a tulajdonságaik amikre nekünk feltétlenül szükségünk van a kitűzött részfeladatunk megoldásához. Ezek a tulajdonságok a következők: hordozhatóság, azonos megjelenési forma platformtól és rendszertől függetlenül, szabványos dokumentum szerkezet és felépítés.

Egy másik nagy előnye a szabványos felépítésnek, hogy a dokumentum módosítása, feldolgozása és felhasználása nagyban leegyszerűsödik és nem ad plusz feladatokat és kötelezettségeket.

Mivel a wxWidgets könyvtárban már található olyan eszközkészlet, amellyel az XML dokumentumainkat kezelni tudjuk, és így jelentős tervezési és fejlesztési időt spórolunk meg. Az egyedüli kihívást az jelenti hogy ezeket az eszközöket a rendeltetésüknek megfelelően és szabályosan használjuk fel. A keretrendszerbe történő integrálása egyszerűnek mondható.

3.1.3.1.2 Felhasználói felület felépítése

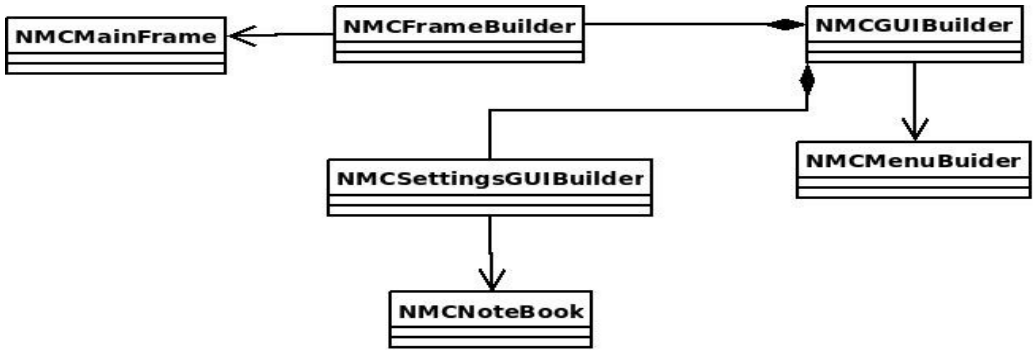
Felhasználói felület tervezésekor először is jól át kell gondolnunk pontosan mit is akarunk megjeleníteni és mindezt milyen formában szeretnénk megtenni. Mindezt úgy kell megtennünk, hogy a végeredményként előálló felhasználói felület könnyen kezelhető, és áttekinthető legyen. A felhasználónak nem keljen sokat keresnie ahhoz, hogy megtalálja amit keres.

Első lépésben a bontsuk fel részekre a felhasználói felületen megjeleníteni kívánt részeket.

A felhasználói felület felépítéséhez felhasználtam az építő(Builder) tervezési mintát a rendszer tervezése során. A felhasználói felületet felépítéséért a NMCGUIBuilder osztály a felelős, ez az osztály építi fel a felhasználó által a képernyőn megjelenő form -ot. NMCGUIBulider objektum létezésének egyetlen egy célja van a felhasználói felület elemeinek összerakása és köztük a megfelelő kapcsolat kialakítása, ez az objektum tartalmazza a menürendszer felépítését végző NMCMenuBuilder osztály egy objektumára történő hivatkozást. Továbbá tartalmaz még ezen kívül a felhasználói felület egy egy részének felépítéséért és módosításáért felelős egyéb objektumokat. A közvetlen kérések NMCGUIBuilder objektumhoz futnak be, és csak ezután továbbítódnak a megfelelő alrendszer kezeléséért felelős objektumhoz, vagy a „plugin” -ek kezelésért felelős objektumokhoz.

A következő UML diagram a GUI alrendszer komponensei közötti összefüggést ábrázolja a 1 ábra ábrázolja.

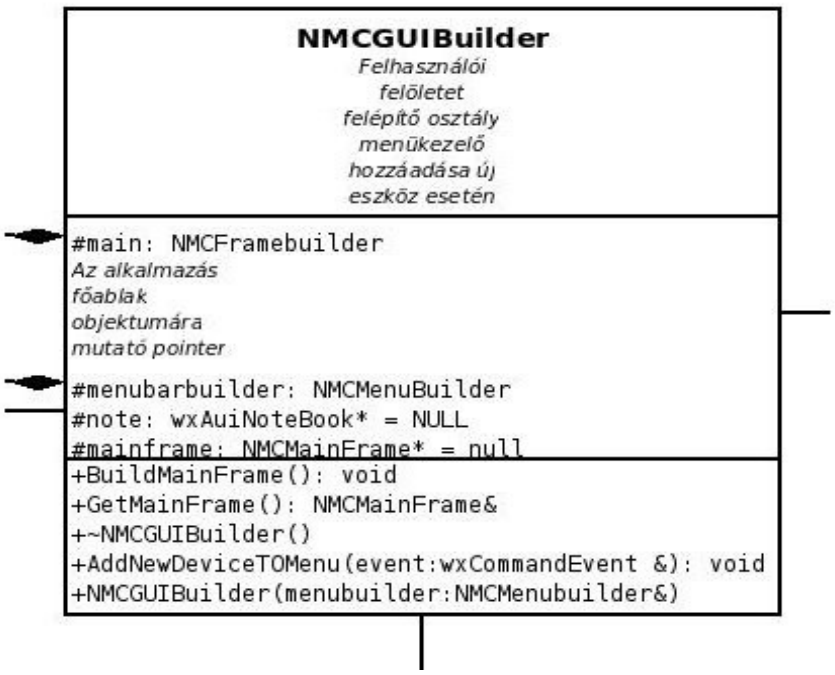
A nyílban végződő vonalak az osztályok közötti hivatkozásokat jelölik, míg a rombuszban végződő vonalak az osztályok közötti összetételt jelöli.



6. ábra. Grafikus alrendszer részei

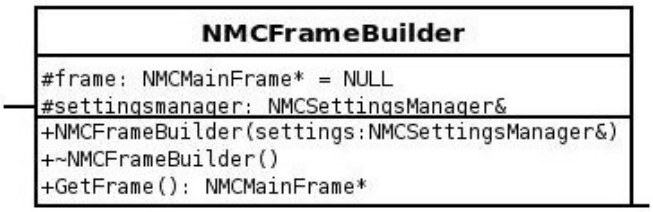
Az egyes osztályok feladatai a következők:

NMCGUIBuilder: Az egyes részek a felépítésében és módosításáért felellős objektumokra tárolja a hivatkozásokat és ha kérelem érkezik hozzá az adott rész módosításához, akkor azt továbbítja a megfelelő objektumnak, ezzel egy újabb egységes felületet biztosítva a felhasználói felületek módosítására futási időben. Az osztály konstruktorának meg kell adni beállításokat kezelő NMCSettingManager osztály egyetlen objektumára egy hivatkozást. A 2 ábra ábrázolja a részeit, ezek az ábrákon csak a fontosabb részek vannak feltüntetve.



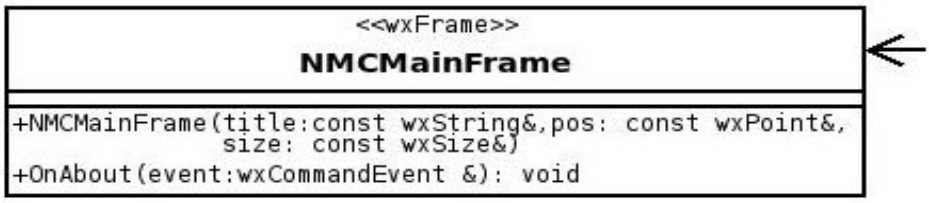
7. ábra NMCGUIBuilder főbb részei

NMCFrameBuilder: A beállítási adatok alapján és az alapértelmezések felhasználásával felépít egy alap wxForm -ot későbbi módosítások és fejlesztések esetén ezt az osztályt módosítva lehet tovább fejleszteni a rendszert. Az előállított végterméket a GetMainFrame metódussal kaphatjuk meg, ha rendszer futása során módosítani kell az adott form megjelenésén azt csak az előállított terméken tehetjük meg. Ez az osztály csak a létrehozással foglalkozik. A részeit a 3. ára ábrázolja.



7. ábra NMCFrameBuilder főbb részei

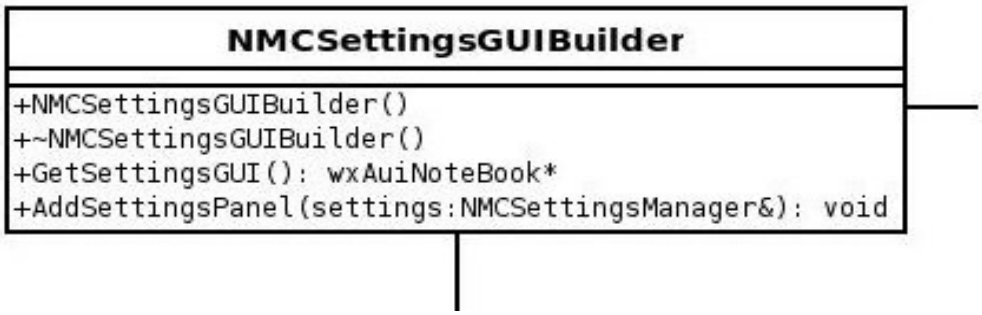
•NMCMainFrame: A keretrendszer főablakának osztálya ezt példányosítva a megfelelő paraméterekkel a konstruktornak megadva elkészül egy egyszerű üres ablak, amin csak az ablak címe és a méretezéshez, bezáráshoz szükséges gombok vannak jelen.



8. ábra

•NMCSettingsGUIBuilder: Az eszközök beállításainak megjelenítésért felelős objektumok előállításáért végző osztály az ehhez szükséges adatokat a beállításokból olvassa ki és ezt használja fel. Ennél az osztálynál jelenik meg a leghatározottabban a mérőeszközök eszközök illesztés, ugyanis ahhoz, hogy illeszteni tudjuk közbe kell iktatnunk még egy absztrakciós réteget a keretrendszer és a mérőegységet kezelő egységek közé, hogy egy egységes felületet biztosítsunk ezzel növelve a rugalmasságot az eszközök változásával szemben. Ennek a rétegnek képesnek kell lennie megadni a csak az adott mérőeszközre jellemző beállításokat és ezeket egységes formában. Ezért az eszközcsatoló rétegnek vissza kell tudnia adni egy olyan wxPanel objektumot ami csak az adott berendezésre jellemző, de a rajta

bekövetkező változtatásokat a keretrendszer csak továbbítja. A részek közötti kapcsolattartásról az eseménykezelő rendszer gondoskodik. A részeit az 5. ábra tartalmazza.



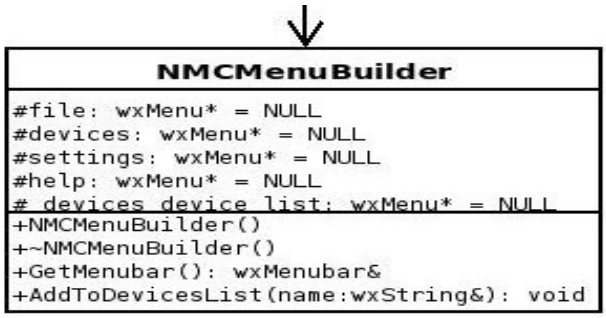
9. ábra

NMCNoteBook: A wxNoteBook egy leszármazott gyermekosztálya, amely a megkapott információk alapján a kész felületen fogja megjeleníteni az egyes eszközök beállításait. Új eszköz hozzáadása esetén, ez az osztály végzi az új eszközbeállításának megjelenítését is. Minden eszköz külön fülön jelenik meg. Ebből az osztályból csak egyetlen egy példány létezik a program futása során. A következő kép megmutatja, hogyan jeleníti meg a felhasználói felületen:

10. ábra

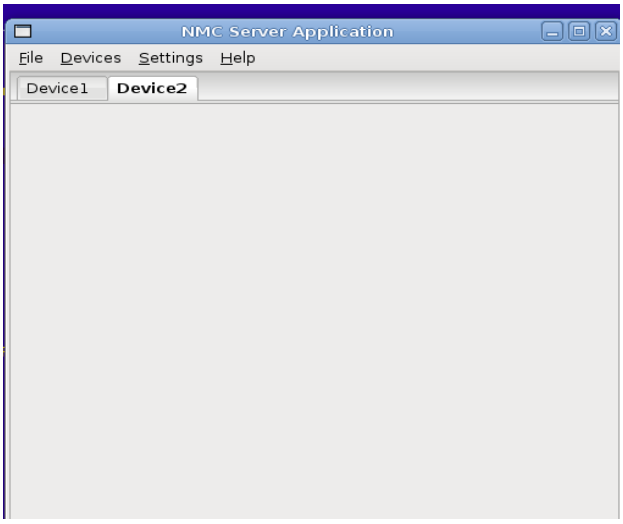
A menüsor alatt megjelenő fülek egy - egy eszköz beállítására használhatóak, mindegyik oldalon megjelenik egy mentés gomb amivel el lehet menteni a módosításokat a beállítási fájlokba és később vissza lehet azokat tölteni a rendszer újbóli elindítása során.

NMCMenuBuilder: A menürendszer felépítésért és karbantartásáért felelős minden, menü külön - külön módosítható a új elemekkel bővíthető és talán, ami a legfontosabb, új menüelemek tölthetők be a egy .xrc fájlból, de csak az eszközök menü bizonyos részét lehet így módosítani, ahol az eszközök felsorolásra kerülnek egyenként, a hálózati eszközök neve mellett egy ikon is megjelenik ezzel is figyelmeztetve a felhasználót, arra hogy nem helyben megtalálhat eszközről van szó. A 7. ábra mutatja az osztály általános részeit:



11. ábra

Ez az osztály egy igazi építő, mert minden egyes menü elemeket egyenként építi fel. Emelet más feladatokat is ellát a keretrendszerben, az új menüelemek beillesztésére is lehetőséget biztosít, ezzel újabb funkciókkal is gazdagíthatja az eszközkezelő alrendszert, mivel közvetlen menüelérést tesz lehetővé a felhasználó számára.



12. ábra

Az egyes eszközök a Devices menü Connected Devices almenüjében találhatóak meg. Az itt felsorol eszközök valamelyikét kiválasztva a NMCNoteBook által kezelt ablakrészben az adott eszköz beállításai válnak aktívvá, és ezeket a beállításokat lehet módosítani. A Devices menü többi eleme az különböző eszközök hozzáadásában lehet nagy segítségünkre .

Ezen eszközök a következő két típus valamelyikébe tartoznak:

- Helyi gépben kezelt eszköz(ök).
- Távoli gépben kezelt eszközök.

Ezeket az eszközöket NMCDeviceMeneger felügyeli és kezeli. Új mérőeszközöket fel lehet venni a megfelelő „plugin" betöltésével, aminek a pontos elérését meg kell adni a felbukkanó párbeszédablakban. Ez a menüpont felválthatja az előző kettőt, ha létezik pontosan megírt konfigurációs fájlunk, vagy beépülő modulunk az adott eszközhöz és azt sikeresen be is tudjuk tölteni a programunkba. Sikeres betöltés alatt azt értjük, hogy a beállítási fájlban szereplő osztott programkönyvtárat vagy programot sikeresen betöltöttük a memóriába a megfelelő paraméterekkel és semmilyen egyéb hiba nem lépett fel a betöltés során.

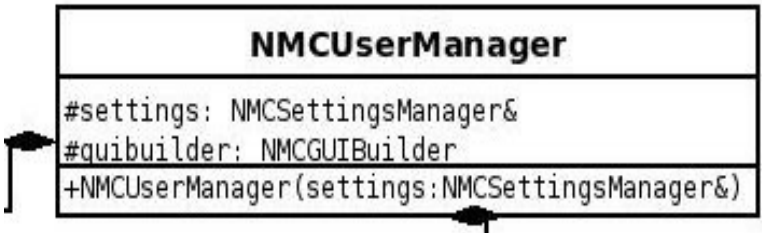
Az NMCMenubuilder objektum amellett felépíti az alkalmazás menüjét egyéb fontos feladatok elvégzésére is képes, a hozzá beérkező kérelmeket továbbítja a megfelelő beállításokat kezelő alrendszer felé.

3.1.3.1.3 NMCUserManager alrenedszer:

Ez az alrendszer magában foglalja a felhasználói felület kezelő alrendszert valamint a beállításokat kezelő alrendszert is. Ezzel egy újabb réteget képvisel az alkalmazásban, amely más objektumok felé egy egységes és nem megkerülhető felületet biztosít. Ezen a felületen keresztül lehet csak szolgáltatásokat kérni a megjelenítéssel foglalkozó alrendszertől, továbbá a továbbítja a nem általa kezelt alrendszereknek szóló kérelmeket és olyan eseményeket fogad a többi alrendszertől, amelyről a felhasználónak feltétlenül tudomást kell szereznie. Ilyen

események lehetnek pl.: hibaüzenetek, figyelmeztetések, és állapot jelentések.

NMCUserManger osztály részeit a következő ábra mutatja.



13.ábra

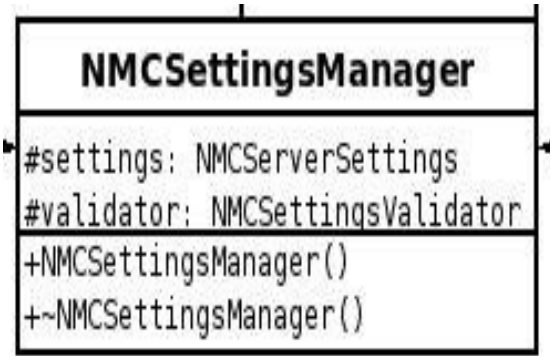
Ezen osztály objektumai hivatkozásokat tárolnak a következő alrendszereket kezelő osztályokra: NMCGUIBuilder, NMCSettingManager. Mindezt azért teszi, hogy közvetítő szerepet játszhasson a felhasználó és a rendszer egyéb elemei között. Ezt a közvetítő szerepet csak úgy tudja ez az osztály elvégezni, ha minden olyan résznek el tuja küldeni a megfelelő kéréseket amihez feltétlenül hozzá kell hogy férjen. Ezek a részek általában a beállításokkal kapcsolatosak. A beállításokat kezelő alrendszerre csak egy hivatkozás tárol és semmi mást. A felhasználói felületlétrehozásakor, ezen osztály egyetlen egy objektumlétezik, amely létrehozáshoz átadja a szükséges beállításokat a felhasználói felületépítő alrendszernek.

3.1.3.1.4 Beállításokat kezelőalrendszer részei és feladataik

Ezen alrendszernek igen fontos feladatai vannak. Ennek az alrendszernek kell biztosítani a beállítások megőrzéséről és a későbbi visszaállításáról. Mindeközben fel kell tudnia ismerni a beállítások hibáit és ezt jeleznie kell. Mivel minden manager osztály leszármazottja az NMCManager absztrakt osztálynak, amely leszármazottjainak garantálniuk kell, hogy ismernek egy olyan objektum hivatkozást, amely a hibák kezelésével foglalkozik. Azaz a rendszerben léteznie kell egy olyan alrendszernek is, ami a hibák kezelésével foglalkozik. Míg a hibák felismerése az egyes alrendszerek saját feladata, mivel az esetlegesen felmerülő kisebb hibák kezelés helyben a leggyorsabb és csak akkor kell továbbadni a megfelelő hibakezelőnek ha azt helyben nem lehet kijavítani és lekezelni.

3.1.3.1.4.1 NMCSettingsManager osztály

Ennek az osztálynak az a feladata, hogy kezelje a beállításokkal foglalkozó teljes alrendszert és egységes felületet biztosítson a többi alrendszer szükséges beállításaihoz való hozzáféréshez. Ezzel biztosítva, hogy mindenki csak ahhoz férjen hozzá amire éppen szüksége van a munkája elvégzéséhez, és semmi máshoz. Ezen osztály egyetlen objektumának garantálnia kell a kölcsönös kizárást a beállítások módosítása esetére, és gondoskodnia kell arról, hogy a beállításokban történt esetleges módosítások minden érintett alrendszer esetében rendelkezésre álljanak a lehető legaktuálisabb formában. Ezen osztály objektumai valósítják meg a közvetítő szerepét, aki minden érintett felet informál a változásokról ha erre kérés érkezik hozzá az általa megvalósított felületre.



14. ábra

A 11. ábra azt mutatja hogyan jelenik meg az UML osztálydiagramban. Az osztály objektuma tartalmaz egy hivatkozást a fő beállítási adatokra amelyekből a szükséges információkat kinyerheti a felületére érkezett kérések alapján. Ezen adatok aktualitása és hibamentessége a legfontosabb az egész keretrendszer szempontjából, ha ezek az adatok valamilyen hibát tartalmaznak annak súlyos következményei lehetnek a rendszer működésére vonatkozóan. Ezért itt kell meghozni a legfontosabb döntéseket arra vonatkozóan, hogy milyen adatok kerülhetnek bele ebbe az „adatbázisba” és, hogy ezeken az adatokon milyen műveleteket végezhetünk el. A műveletek elvégzése előtt azonban, az adatokat gondosan ellenőriznie kell az alrendszernek és csak ezután léphetünk tovább a beállítások alkalmazása felé. Ezért ennek az alrendszernek létezni kell egy validátor részének is ami teljes egészében elvégzi a beállítás hibaellenérzését is, és az esetleges hibákról visszajelzéseket küld.

3.1.3.1.4.2 NMCSettingsValidator osztály

Ezen osztály felelős, a beállítások hibamentességéért, az új beállítások teszteléséért a végleges alkalmazás előtt és a kimentés előtt. Ezenkívül ha valamilyen hiba lépett fel az új beállítások alkalmazása során, képes a hibát felismerni és ezeket jelezni a megfelelő hibakezelő processnek, amely a beépített megoldáskeresővel megpróbálhatja megoldani a problémát és elvégezni a szükséges módosításokat, vagy súlyos hiba esetén a felhasználói beavatkozást kérni.

3.1.3.1.4.3 NMCSettingsTester osztály

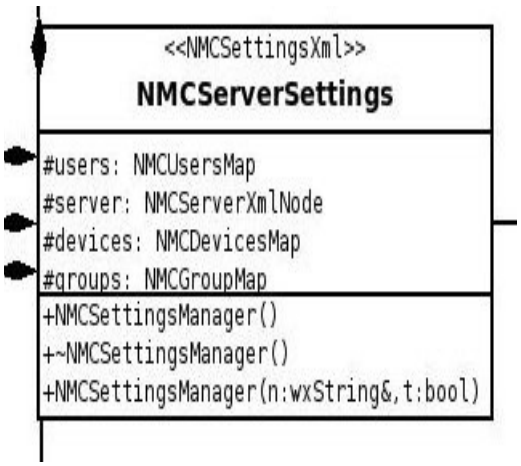
Ezen osztály objektumainak feladata, hogy az újonnan érkező beállításokat tesztelje bizonyos előre megadott paraméterek alapján. Az ilyen típusú objektumoknak a feladat a beállítási fájlok ellenőrzése és hibainformációk előállítása. Itt kell garantálnunk azt, hogy nem kerül olyan beállítása a rendszerbe ami esetleg kárt tehet benne. Ez csak egy alap osztály, amely mintát ad arra vonatkozóan hogyan lehet integrálni az XML típusú beállítási fájlok ellenőrzését egy ilyen keretrendszerbe. Ez az egyik ilyen illesztési pont amit a későbbiekben tovább lehet és kell is fejleszteni, mert a beállítások nagyon fontos részét alkotják a rendszernek. A beállításoknál figyelembe kell vennünk azt a tényt, hogy nem csak egyetlen fajta mérőberendezést felügyelünk egy kísérlet során, hanem egy egész berendezés családot, amiknek a beállításai nem biztos, hogy átvihetők egyik eszközről a másikra módosítások nélkül.

Nem cserélhetjük össze például a detektorok típusát, mert ez az eset komoly következményekkel járna. Ezért a keretrendszerben csak egy illesztési felületet határozhatunk meg komolyabb megkötések nélkül általánosan. A megvalósítás már erőteljesen platform és eszközfüggő. Ezért az illesztési felületet a következőképpen érdemes kialakítani:

Minden berendezéshez léteznie kell egy olyan beállítási fájlnak, ami csak az eszközre alkalmazható beállításokat tartalmazza, továbbá egy olyan a keretrendszer által feldolgozható beállítási fájlnak, amely meghatározza a rendszernek a következőket:

- Honnan töltse be a megfelelő drivereket ?
- Milyen paraméterekkel kell betölteni ezeket?
- Hol találhatóak azok az osztott programkönyvtárak, amelyekkel a berendezés illeszthető a keretrendszerhez?
- Ezeket az osztott programkönyvtárakat milyen paraméterekkel lehet betöltenünk akár a futó programunkba is menet közben?
- Milyen lehetőségeket kínál egy- egy ilyen könyvtár a vezérlésre?

3.1.3.1.4.4 NMCServerSettings osztály



15. ábra

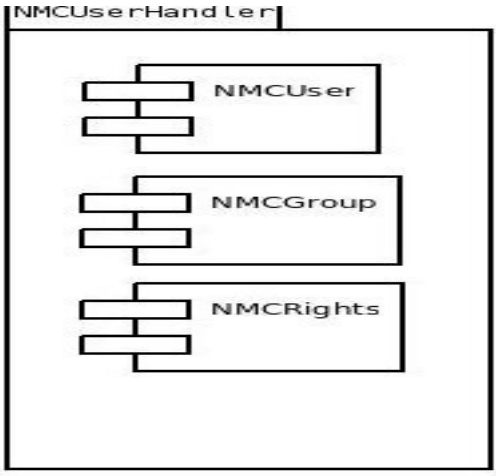
Ezen osztály egyetlen objektuma tartalmaz egy konfigurációs fájlt, melyben megtalálhatók az szükséges beállítások. Minden beállítást típusa szerint szétválogat már a fájl betöltésekor és ezen beállítások érhetőek el közvetlenül a felületén keresztül. A felületét a NMCSettingsXmlNode osztálytól örökölte. Ez az osztály egységes eszközöket nyújt a beállítási fájlok kezelésére és módosítására, ugyanakkor biztosítania kell a kölcsönös kizárást is, hiszen a beállítások egy olyan erőforrást alkotnak amelyet meg kell védeni attól, hogy az egyes folyamatok önkényesen módosítsák azokat figyelmen kívül hagyva az időközben bekövetkezett változásokat.

Mivel a beállítási információk a rendszer futása közben dinamikusan változnak ezért

egyetlenegy részfolyamat sem rendelkezhet saját példánnyal a beállításokról, hanem ezt egy közvetítő objektumon keresztül teszik, egységes felületet biztosítva ezáltal a beállítások kezelésének. Ezen osztály tartalmazza az egyes beállítás típusokat tartalmazó hash táblázatokat is, amik a következők:

- NMCGroupsMap: a csoportokat tartja nyilván XML csomópontként így egyszerűsítve le a keresést.
- NMCUsersMap: A rendszer által ismert felhasználókat tartja nyílván a hash tábla kulcsa a felhasználónév, ami alapján könnyen megkereshetjük az adott felhasználót.
- NMCDevicesMap: A rendszerben definiált eszközök beállításait tartalmazó csomópontokat tartja nyilván. A has táblázat elsődleges kulcsa a devic id -ja, amely minden eszköznél egyedi.
- NMCSettingsFiles: További beállítási fájlok betöltéséhez szűkséges információkat tartalmaz, amiket akár egyenként is elmenthetünk, ha módosítottunk valamit bennük.

Ezen osztály leszármazottaival megvalósítható a beállítások esetleges adatbázisban való tárolása is. Ezzel is bővítve a keretrendszer lehetőségeit. Az a beállítási információkat át lehet adni XML adatokként ennek az osztálynak és vissza is lehet tölteni az adatbázisba XML adatként.



16. ábra

3.1.3.1.4.5 NMCSettings osztály

Ez az osztály egy XML dokumentumot reprezentál, amelynek a root node -jának neve nmcsettings. Ez a dokumentum tartalmazza a beállításokat, amelyek a rendszer működéséhez szükségesek. Az osztály a wxXmlDokument osztályból van leszármaztatva, ezzel nagyban megkönnyíti az implementálást és egy jó kidolgozott felületet használunk a beállításaink kezelésére, amely lehetőséget biztosít akár az adatfolyamokból történő betöltésre is. Az adatfolyamokból történő betöltés nagy segítséget jelen abban, hogy a beállításainkat átvigyük egyik gépről a másikra, arról nem is beszélve, hogy akár közvetlenül adatbázisból is tölthetjük a szükséges adatokat. A beállítások részei megfelelnek egy adatbázis tábláiban található információknak, amiket közvetlenül le is kérhetünk és a válaszokat xml -ben kapjuk.

Ezen előnyök mellett, ezen osztályok objektumai létrehoznak egy-egy hash táblát is a betöltés közben és feltöltik azokat az egyes beállítás típusoknak megfelelő értékekkel. Ezekben a hash táblákban könnyen és gyorsan tudunk keresni értékeket, emellett a karbantartásuk is egyszerű.

Ezen osztály felületet nyújt az adatok módosításához is, így például a felhasználók, csoportok és eszközök kezeléséhez is. Ezen osztály minden egyes példánya egy egy beállításokat tartalmazó XML fájlt reprezentál, melyben az objektumon végzet műveletek a mentés során megjelenek és később visszatölthetőek lesznek a rendszerbe, ha szükségünk van rá.

Azért származtattam le ezen osztályt a wxXmlDokument osztályból, mert így könnyebben ellenőrizhetjük, hogy a fájlunk megfelel -e az XML szabványnak és ténylegesen a keretrendszernek szóló részeket tartalmaz.

Az osztály által tartalmazott hash táblák a következők:

- NMCGroupHash: ebben a táblában a felhasználói csoportok vannak tárolva groupid szerint.
- NMCGroupNameHash: ebben a táblában a felhasználói csoportok vannak tárolva csoport név szerint.

- NMCDeviceNameHash: Az eszközök beállításait tartalmazó tábla, ahol a kulcs az eszközneve.

Mindegyik tábla a wxWidgets hash map -jét használja, és az ehhez szükséges makrókat hívjuk meg a definiálásukra a megfelelő paraméterekkel. Így egyből használatra kész eszközöket kapunk és kevesebb hibalehetőséget amit elkövethetünk az implementálás során.

3.1.3.1.4.6 NMCGroups osztály

Ez az osztály reprezentálja a felhasználói csoportokat a rendszerben. Minden csoporthoz tartozik egy csoportnév, és egy csoport azonosító valamint jogokat megadó adat, amiben a következők szerepelhetnek:

- olvasási(read) jog: ezzel a joggal csak olvasni lehet a beállításokat, és az adatokat módosítani nem. Ez az alapértelmezett.
- írási(write) jog: ezzel a jogosultsággal változtatni lehet a beállításokon.
- Futtatási jog: ezzel a joggal el lehet indítani a mérési folyamatot. Így ezt nem teheti meg minden felhasználó, csak bizonyos csoportok.
- Távoli bejelentkezési jog, ezzel a joggal távolról kapcsolódhat az adott felhasználói csoport a rendszerhez. Ha ez tiltva van a rendszer visszautasítja a távolról történő hozzáférési kísérletet

Egy egy eszközhöz csoportszinten tudunk jogosultságokat hozzárendelni, az adott csoporthoz tartozó felhasználók csak a csoport számára engedélyezett műveleteket hajthatják végre. Minden eszközhöz hozzá van rendelve legalább egy csoport, ami az eszközt kezeli és beállításait módosíthatja. Így egy eszközhöz csoportokat rendelünk hozzá jól meghatározott jogokkal. Továbbá lennie kell egy helyi felhasználónak, aki minden beállítás módosíthat az adott gépen és esetleg távolról is bejelentkezhet

Az NMCGroup objektumok a következő hash táblákat is tartalmazza:

- NMCUserHash: a felhasználókat tartalmazó hash táblázat, ahol a kulcs a felhasználó uid (user id) -ja, így a felhasználók között tudunk keresni uid -szerint ebben a

táblában.

- NMCUserNameHash: a felhasználókat tartalmazó has táblázat ahol a kulcs a felhasználónév(username), így a felhasználók között tudunk keresni user name -szerint ebben a táblában.

Ezekkel a táblákkal könnyebben kezelhetjük a felhasználókat. Léteznie kell egy olyan felhasználónak és felhasználói csoportnak, aminek minden joga megvan a helyi rendszeren.

3.1.3.1.4.7 NMCUser osztály

Ezen osztály egyetlen objektuma reprezentál egy felhasználót. Az osztály objektuma tartalmazza a felhasználó egyedi azonosítóját, felhasználó nevét, teljes nevét és adatait. Mivel a jogokat csak a csoportokhoz rendeljük hozzá, ezért az objektum tartalmazza a felhasználó elsődleges csoportjának azonosítóját.

3.1.3.1.4.8 NMCDeviceSettings osztály

Az osztály objektumai kezelik az egyes berendezések beállításait, ez az osztály szintén a wxXmlNode- osztályból van leszármaztatva megkönnyítve a kezelését és a felhasználást. Ez az egyik legfontosabb része a beállításoknak ezért ki lett egészítve az alaposztály felülete több extra komponenssel.

Egy-egy ilyen objektum a következő adatokat tartalmazza:

- Az eszközhöz hozzárendelt csoportok azonosítóját
- A driverek betöltéséhez szükséges információkat:
  - Elérési utak
  - Paraméterek
  - Szükséges programkönyvtárak a keretrendszerhez illesztéshez.
  - Vezérlő elemekhez szükséges információk, ami alapján a GUI ezen részét felehet építeni és össze lehet kapcsolni az eszköz driverekkel.
  - Erőforrásfájlok elérése, amikre szükség van a vezérléshez.
  - Osztott programkönyvtárak elérése és a betöltésükhöz szükséges paraméterek.

- Adatok kezelésére vonatkozó információk
    - Honnan érkeznek az adatok?
    - Hová kell továbbítani és milyen formában?

Mivel az adatokat az illesztő osztott programkönyvtárak(DLL -ek) XML formátumban adják át a keretrendszernek így nem kell a rendszernek ezeket feldolgoznia. A hibajelzések egy külön csatornán (ipc) érkeznek be a rendszerbe, melyekről feldolgozásuk után visszajelzést küld..

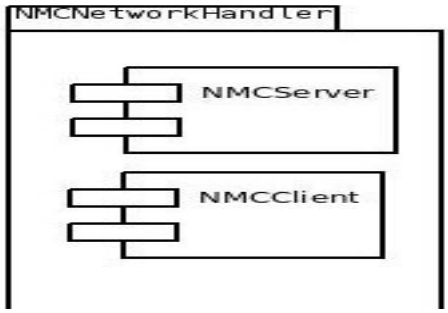
3.1.3.1.5 Hálózat kezelő alrendszer és részei

Ez az alrendszer fontos részét képezi a keretrendszernek, mivel ez biztosítja a kapcsolattartást a különböző rendszerek és felhasználók között; mint ilyen alapos tervezést és elemzés igényel, hogy a lehető legjobban megfeleljen az aktuális igényeknek és a várhatóan felmerülő igényeknek. Ezért a lehető legáltalánosabban kell megtervezni. Talán az egyik legnehezebb feladat a más rendszerekről érkező adatok kezelése és továbbítása, ha erre szükség van a beállítások miatt. De mindenek előtt azt kell leszögezni, hogy a beérkező kérésekre a lehető legrövidebb időn belül válaszolnia kell a rendszernek, ezért az események kezelését szét kell választani, és külön kell kezelni azok típusait. Így külön kell kezelni a következő eseményeket:

- felhasználói felület eseményei
- hálózattal kapcsolatos események(csatlakozási kérelmek, bejelentkezés, adatok)
- helyi eszközök eseményei
- távoli eszközök eseményei
- távvezérlés

Mivel igen sok váratlan esemény van a rendszerben, ehhez ismerni kell a wxWidgets eseménykezelési mechanizmusát és le kell bontani külön önálló egységekre amik képesek egymással együttműködni. Azaz fel kell építeni egy olyan, eseménykezelőkből álló egymással kapcsolatban lévő alrendszert, amely képes megfelelően kezeli a rendszerben fellépő

eseményeket. Egyik ilyen alrendszert a NMCNetworkManager osztály valósítja meg. Ezen osztály objektuma felügyeli a hálózati kapcsolatokat. Külön process felügyeli a szerverhez tartozó socketet így a szerverhez érkező kapcsolódási kérelmeket. Ezt a funkciót az NMCServerProcess típusú objektum látja el. Emellett a külön folyamat kezeli a már bejelentkezett klienseket, és a tőlük érkezett kéréseket is. A kliensek kezelését is a NMCNetworkClient típusú objektumok látják el ezek közül egy egy NMCClientProcess egyszerre többet is kezelhet a terheléstől függően, be lehet állítani, hogy hány klienst kezeljen egyszerre.



17. ábra

Talán az egyik legfontosabb kérdés az erőforrásokhoz történő konkurens hozzáférés kérdése, szerencsére ebben a kérdésben is segítségül hívhatjuk a wxWidgets tool -okat és stratégiát választhatunk.

3.1.3.1.5.1 NMCNetworkManager

Az ilyen típusú objektumnak fontos szerep jut a keretrendszerben. Egyrészről távolról történő felügyeletet kell biztosítania a rendszerhez, másrészről adatokat is továbbítania kell a megfelelő csomópont felé. Ezen alrendszer nagy mértékben támaszkodik kisebb részrendszerekre és építőelemekre. Az egyik ilyen építőelem a szálak kezelésére a wxWidget által biztosított eszközkészlet vagy más néven tools. Ezen eszközök figyelembe veszik az egyes rendszerek közötti eltéréseket és egy egységes felületet biztosítanak a szálak és processzek programozására, elfedve az egyes rendszerek közötti különbségeket. Tervezésnél,

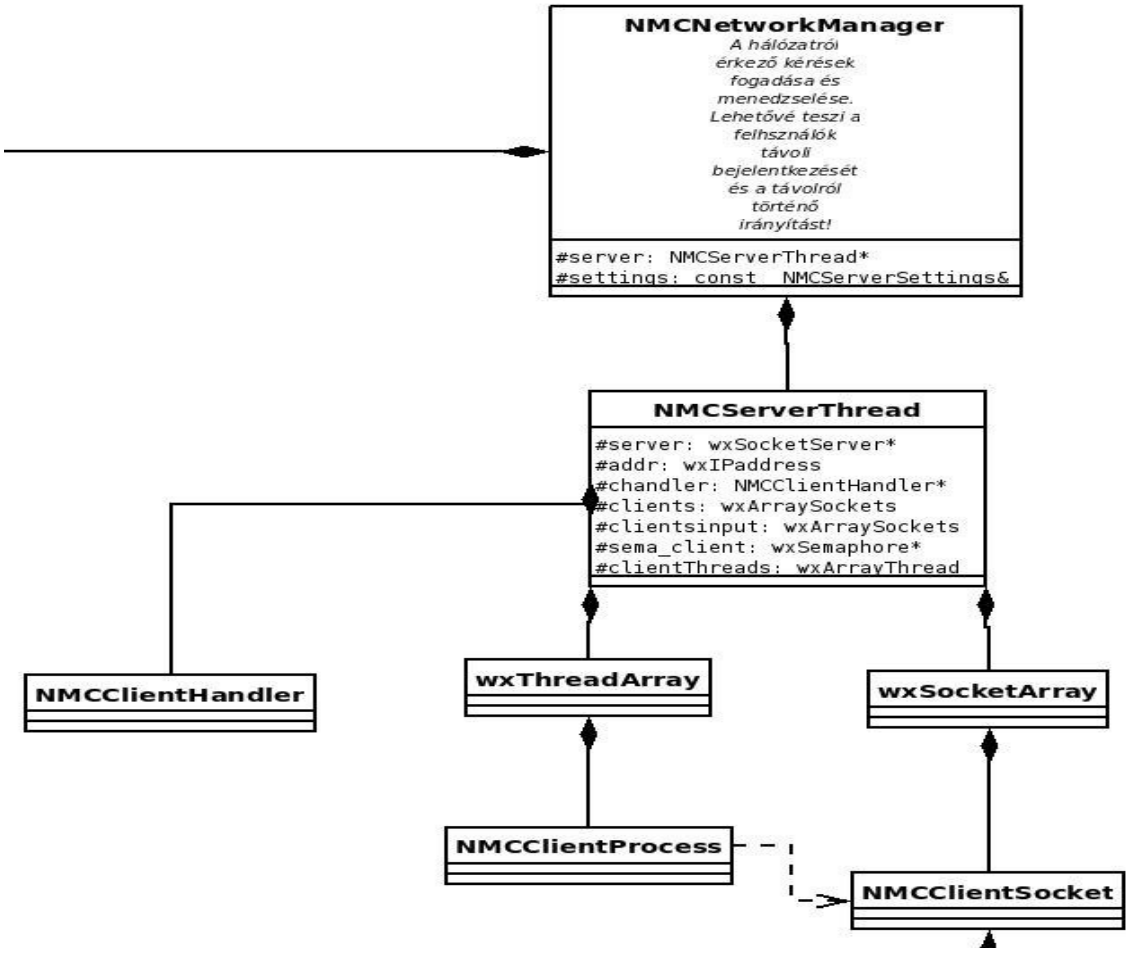
implementálásnál és tesztelésnél figyelembe kell venni olyan operációs rendszer specifikus dolgokat és jellemzőket, amiket a wxWidget elfed, de ott vannak a háttérben. Egy ilyen jellemző lehet akár a gyermek folyamatok maximális száma és a CPU teljesítménye, amit arra tud fordítani, hogy kiszolgálja az egyes végrehajtási szálakat. Ezen osztálynak két fontosabb adattagja van:

- NMCServerThread: hivatkozás a szerverhez érkező kapcsolódási kérelmeket kezelő processre.
- wxThreadArray: klienseket kezelő végrehajtási szál objektumok tömbje, amellyel felügyelhetjük a kliensek folyamatait. Ez csak wxThread típusú objektumok tárolására szolgáló általános tároló.

Ezek az adattagok mindegyike közvetlen kapcsolatban van a szálak kezelésével a keretrendszerben, ugyanis ebben az alrendszerben igen fontos feladatokat töltenek be.

Az NMCServerThread kezeli a beérkező kapcsolódási kérelmeket és a felhasználók autentikációját. Ezt a folyamatot külön kell választani a többi hálózati eseménytől, mivel viszonylag kevés információ átvitelt kíván meg, de a korrekt végrehajtás és feldolgozás annál fontosabb.

A wxThreadArray adattag tartalmazza talán a legfontosabb objektumokat a hálózat kezelő alrendszer szempontjából, mivel benne helyezkednek el a távoli felhasználókat kiszolgáló folyamatok objektumai. Ennél az adattagnál garantálni kell, hogy minden folyamat szabályosan befejeződjön, ha nincs rájuk szükség többet. Egy klienshez egy folyamat tartozik, de NMCNetworkManager képes arra, hogy nagyobb igények esetén további folyamatokat indítson a kérések kiszolgálására.



18. kép

3.1.3.1.5.2 NMCClientThread osztály

Ezen osztály objektumai végzik a távoli felhasználók kiszolgálását, vagyis fogadják a kliensek kéréseit és információt továbbítanak feléjük. Erre a célra minden objektum rendelkezik egy alkalmas tárolóval, amiben a kliens kapcsolódási adatait tartalmazza. Egy kliens ha befejezte a kapcsolatot a folyamat képes fogadni újabb klienst miután bezárta a kapcsolatot. Ezen osztálynak egységes felületet kell biztosítania a kliensek munkamenetének valamint a biztonsági eszközök kezelésére.

Az osztály leszármazottja wxThread osztály amely örökli a felületének egy részét, valamint kibővítheti azt.

3.1.3.1.5.3 NMCServerThread osztály

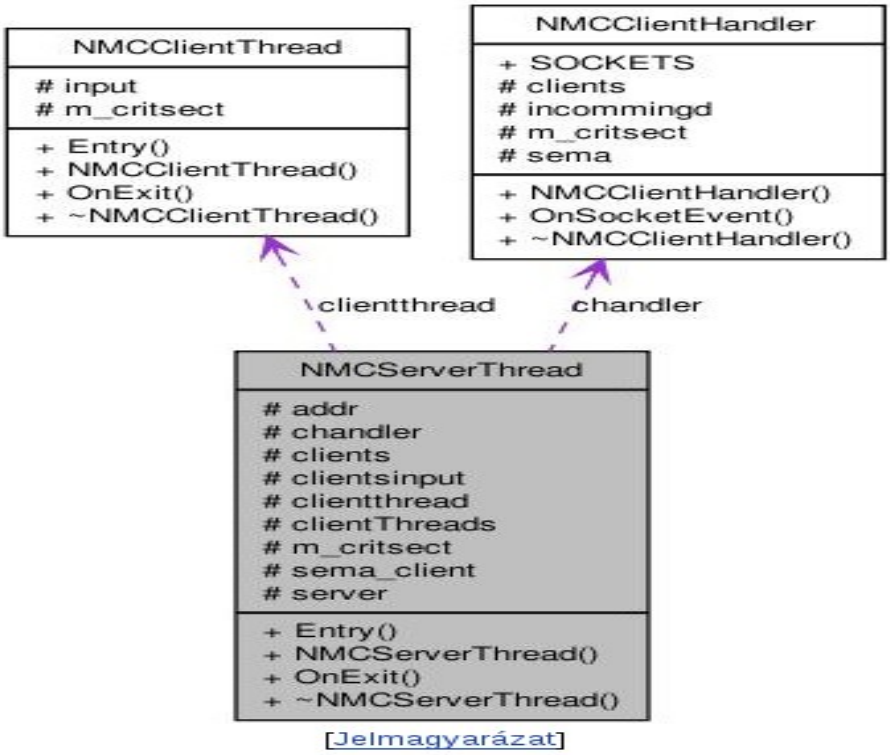
Szerversoceket kezelésére és a felhasználói munkamenetek indítására specializálódott osztály. Futás közben alapesetben csak egyetlen ilyen típusú objektum létezik. Ez az objektum kezeli a szerveroldalon a kliensek kapcsolódási kérelmének kezelését.

Egy ilyen típusú objektum reprezentál egy végrehajtási szálat, amely egyetlen egy szerversocketet kezel. A socket által meghatározott portra érkező kérelmekkel foglalkozik és ezekhez rendeli hozzá a kérelmet kiszolgáló folyamatot.

Az osztály objektumai által reprezentált szálakat csak egy NMCNetworkManager objektum indíthatja el vagy állíthatja meg. Megállítás esetén a teljesen megszünteti az addigi kapcsolatokat és felszabadítja a lefoglalt erőforrásokat. Szabályos befejezés esetén a tőle függő folyamatok is megszűnnek és szabályosan befejeződnek vagy befejezteti működésüket kilépés előtt.

Az osztály példányai a szerver oldalon nyilvántartanak egy a klienseket tartalmazó dinamikus tömböt, amely a szerverhez csatlakozott és bejelentkezett klienseket tartja nyilván. Valamint a klienseket kiszolgáló folyamatoknak is nyilvántartanak egy dinamikus tömböt, amelyben tárolt folyamatok arra várakoznak, hogy egy kérés fusson be a szerverhez és azt teljesíthessék. A terhelés növekedésével az ebben tárolt folyamatok száma is növekszik persze nem korlátlanul és egyre több kliens kérést tud kiszolgálni egyre gyorsabban a program. Ezen kívül az osztálypéldány(ok) tartalmaznak a kölcsönös kizárást elősegítő eszközöket is mint szemaforok és mutexek a kliensek tömbjéhez történő hozzáféréshez. Ezekhez is kitűnő eszközöket biztosít a wxWidgets osztálykönvtár.

A következő ábra megmutatja milyen fontosabb részei vannak ennek az osztálynak és hogy milyen kapcsolatban áll más felületet megvalósító osztályokkal. Az ábra csak vázlatosan jeleníti meg ezeket a kapcsolatokat.



19. kép

A képen az rendszer szempontjából legfontosabb osztályok vannak feltüntetve, persze ezen kívül még kapcsolatban állnak egyéb osztályokkal is, de az átláthatóság miatt egyszerűbb így ábrázolni és a későbbiekben ki is kell bővíteni jó néhány egyéb funkcióval is, hogy megfeleljen a kihívásoknak.

3.1.3.1.6 Eszközkezelő alrendszer felépítése és részei

Mivel minden berendezést, mint egy kiegészítést csatolunk a keretrendszer felületéhez, ezért egy olyan felületet kell megtervezni, hogy az valóban minden igénynek és elvárásnak megfeleljen épp úgy most, mint a jövőben is. Ennél a lényeges résznél igen alaposan át kell gondolni és számba kell venni a jelenlegi és a közel jövőben esetlegesen felmerülő elvárásokat , mivel az a berendezések megfelelő kezelése a legfontosabb.

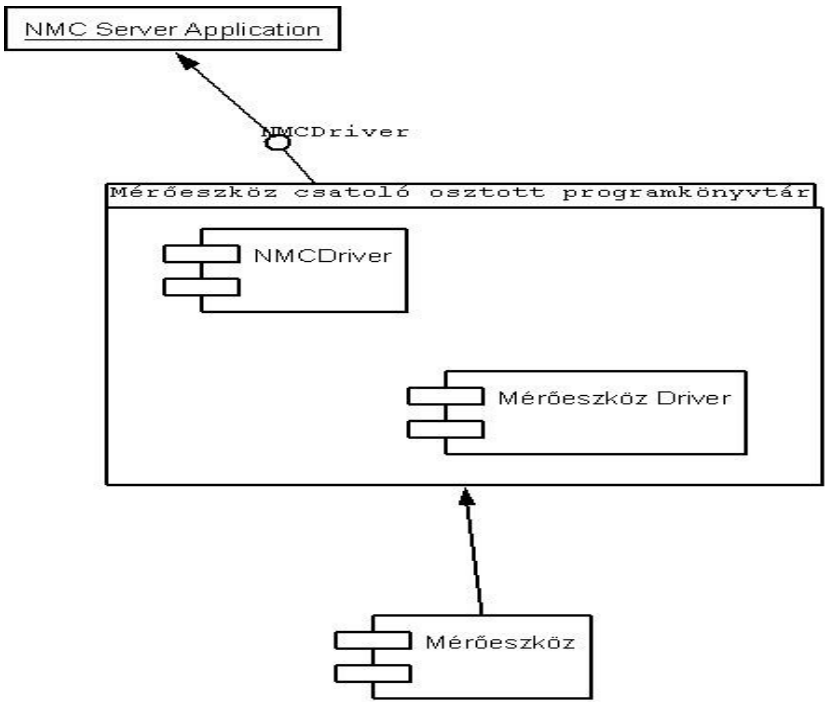
Az eszközkezelővel szemben támasztott legfontosabb követelmények a következők:

- Egyszerre több eszközt is kezelnie kell
- A beérkezett adatok alapján döntéseket kell tudnia hozni arról, hogy mi is történt és szükség van-e további felhasználói beavatkozásra, vagy azonnal le kell állítani az egész folyamatot. Mesterséges intelligenciában használt eszközökkel és algoritmusokkal kell a megfelelő döntéseket meghoznia a lehető leggyorsabban.
- Berendezések által szolgáltatott adatokat továbbítania kell az adatkezelő alrendszer felé.
- Több külön csatornát kell kiépítenie a közvetlenül a berendezést kezelő modullal, egy csatornát a vezérlésre, eredmények és visszajelzések fogadásra, hibajelzésre
- Beállítási lehetőségeket kell biztosítania, a saját működésének befolyásolására, azaz parancsokat képes fogadni.
- Minden eszközt külön folyamatcsoport kell hogy kezeljen, annak érdekében hogy a lehető leghatékonyabban tudjuk kezelni őket.
- Egységes hibajelzési és adatcsatornák

Ezeken az általánosan felsorolt követelmények listája nem teljes, valamit a tervezés és fejlesztés előrehaladtával további fontos kulcskötelességek kerülhetnek még szóba, amelyeket szintén be kell illeszteni a projekt ezen részébe. Így egy igen szerteágazó és bonyolult kérdéskört jelent a berendezések automatizált felügyelete, amely több tudomány terület fontos részét foglalja magában. Ide lehet sorolni a számításelméletet, a mesterséges intelligenciát,

programozást és még sok a témához kapcsolódó területet is.

A következő ábra az eszközök csatolását szemlélteti:



22. ábra

3.1.3.1.6.1 Mesterséges intelligenciai vonatkozások az alrendszerben

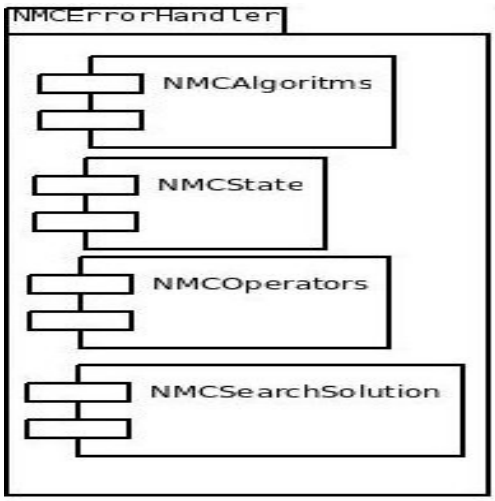
Először is, azért hogy egységesíteni tudjuk a berendezések kezelését egységesítenünk kell a jelzéseiket, mint a hiba jelzéseket, állapot jelzéseiket. Ezért meg kell alkotnunk egy általános mérőeszköz absztrakt modelljét és annak jelzéstípusait, valamit a jelzésekre adható válaszokat külön külön. Meg kell adnunk egy állapotteret, ami a lehető legjobban leírja az eszközöket és szükség esetén további állapotokkal is ki lehet egészíteni.

Továbbá meg kell adnunk operátorokat, vagyis a berendezésen elvégezhető műveleteket le kell írnunk, ahhoz hogy megfelelően tudjuk lekezelni kell írnunk ezek hatását is.

Vagyis egy teljes minden részletre kiterjedő állapottér reprezentációt kell adnunk egy absztrakt modell mérőeszközre, amire már implementálni tudjuk a szükséges algoritmusokat a döntéshozatalhoz. Szabványszerűen biztosítani, hogy minden egyes beépülő modulnak az ehhez szükséges feltételeket biztosítania kell.

A döntési algoritmusokat úgy kell kiválasztani, hogy az esetlegesen az állapottérgráfban előforduló köröket is kezelni tudja és a lehet leghamarabb véges időn belül választ adjon a feltett kérdésre! Nem várakozhatunk túlságosan sokáig ezért a megoldás keresésére érdemes párhuzamosan futtatható algoritmusokat alkalmazni.

A következő ábra a hibakezelőbe építendő fontosabb komponenseket ábrázolja, mint egy csomag részeit:



20. ábra

Az ide tartozó komponensek a következők:

- NMCAlgorithms: a megoldás kereső algoritmusokat tartalmazza általános formában.
- NMCState: a lehetséges állapotok általános leírását tartalmazza, amelyek plugin-nel kibővíthetők
- NMCOperators: Az elvégezhető operátorok általános absztrakt leírását tartalmazza.
- NMCSearchSolution: megoldások kereséséhez nyújt támogatást ez a komponens.

Ezeken a főbb komponenseken kívül egyéb kiegészítéseket is be lehet illeszteni a rendszerbe, amikkel növelni lehet a keresés és a hibakezelés hatékonyságát.

3.1.3.1.7 Adatkezelő alrendszer részei

Míg az előző alrendszerben a fizikai eszközökről és erőforrások kezeléséről eset szó nagy vonalakba, addig elérkeztünk talán az egyik legfontosabb erőforráshoz magához az információhoz. Talán ez a legértékesebb és ugyanakkor a legsérülékenyebb erőforrás amennyiben nem jól bánunk vele.

Az adatok kezelése és tárolás vagy továbbítás igen sok kérdést vet fel az alrendszer működésével kapcsolatban, amelyek a következők:

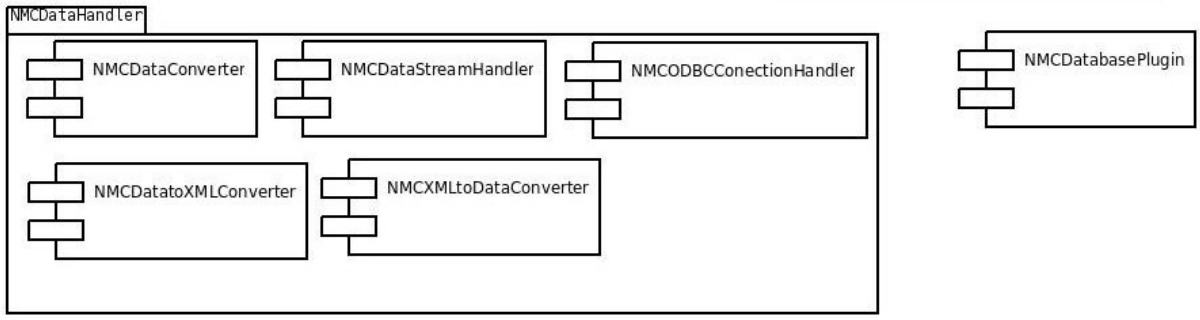
- Milyen formában érkeznek be az adatok?
- Kell -e valamilyen átalakítás vagy ellenőrzést végezni az adatokon?
- Mi történjen az adatokkal a beérkezésük után?

Ezekre a kérdésekre kell pontos válaszokat adnia a rendszernek. Ebből a szempontból ez az alrendszer igen sok sajátosságot mutat az informatikából már jól ismert rendszerteljesítmény figyelési megoldásokhoz:

Az adatokat lehessen a helyi fájlrendszerben tárolni további feldolgozás céljából vagy közvetlenül egy adatbázisba továbbítani, de ehhez egységesíteni kell az adatok formátumát, hogy később újra felhasználható legyen további elemzéshez. Azonban az adatokat át kell alakítani valamilyen köztes és egységes formára ez lehet például egy XML dokumentum.

Az XML formátumra történő átalakítás azért is célszerű, mert így egységesé és szabványossá válik az adatok formátuma és minimális adatátalakítást kíván meg.

Az adatkezelő alrendszernek a felhasználó által vezérelhetőnek kell lennie akár távolról is, valamint különböző eszközspecifikus beépülő moduloknak is illeszthetőnek kell lennie a rendszerhez. Ezért a rendszer egésze szempontjából igen fontos a beépülő modulok (pluginek) korrekt és biztonságos kezelése. Következő ábra mutatja a rendszer ezen részének főbb komponenseit:



21. ábra

Az ábrán látható komponensek egy lehetséges felosztást kínálnak. Az egyes komponensek feladatkörei a következők lehetnek:

- NMCDataConverter: Az adatokon végzet átalakításokért felelős komponens.
- NMCDataStreamHandler: Adatfolyamok kezelését végző komponens.
- NMCODBCConectionHandler: Adatbázishoz történő kapcsolódást kezelő komponens, amely felhasználja az ODBC drivereket a kapcsolódás közben, hogy elrejtse az egyes adatbázisok közötti különbségeket.
- NMCDatatoXMLConverter: Az adatok korekt módon történő XML formátumra történő konverzióért felelős komponens.
- NMCXMLtoDataConverter: az adatok XML formátumról történő visszaalakításáért felelős komponens.

4 A rendszer vázlatos folyamatábrája

A mellékletben megtalálhatóak a rendszer vázlatos működését ábrázoló folyamatábra folyamatábra „folyamatábra.jpg” néven.

5 Összegzés

A hálózati mérési keretrendszer, mint minden keretrendszer tervezése igen bonyolult és nehéz feladat, mert minden apró kis tervezési döntésnek következményei lehetnek, amelyek sokszor nem is nyilvánvalóak és nehezen észrevehetőek. Ahhoz, hogy egy teljesen a gyakorlatban is használható keretrendszert tervezünk meg kellett ismerkedni a rendelkezésre álló eszközökkel és technológiákkal, amit fel is tudunk használni a tervezési folyamatban alapos ismereteket követel meg. Maga a feladat sem volt mindennapi, mivel át kellett látni a különböző rendszerek működését és lehetőségeit, meg kellett ismerni az adott operációs rendszer sajátságainak egy részét és a kiválasztott wxWidgets framwork eszközeit. A tervezési folyamat során igyekeztem mindig szem előtt tartani ezeket a sajátosságokat és felhasználni azokat a munkában.

Az alapos tervezés rendkívül időigényes lassú folyamat és menet közben tervezni nem mindig a legokosabb döntéseket eredményezi a tervezés során. Vagyis amit már lekezdtünk implementálni rögtön a tervezés után időközben kiderülhet, hogy teljesen át kell alakítani vagy teljes egészében ki kell cserélni valami újabbra mert abban a formában működésképtelen koncepció és így újra vissza kell térni a tervezéshez. A folyamatosan ismétli egymást az újratervezés és implementálás ciklusa egyre kifinomultabbá téve a rendszert, de ez egy igen időigényes folyamat. Összességében az előzetes tervezés fázisáig jutottam el és egy kis részét sikerült átültetni a gyakorlatba, de az koránt sem elégséges egy kifinomult és kiválóan működő keretrendszerhez.

A közben megszerzett ismeretek a rendszerekről és működésükről, a felhasznált eszközökről időközben igen hatalmasra gyarapodott, de így is csak a keretrendszer egy igen kis részét sikerül csak megtervezni, megalapozni. Sikerült az alapokat lefektetni amire később építeni is lehet.

Összességében tapasztalatszerzés szempontjából kiválónak bizonyult a kitűzött feladat, bár nem sikerült teljes egészében elérni a kitűzött célokat, csak az azokhoz vezető út alapjait sikerült letenni. A kitűzött célok még igen érdekes problémákat és megoldásokat rejtegethetnek a megoldásukhoz vezető úton, melyeknek részletek-bemenő kifejtése és

analizálása meghaladná akár egy teljes könyv kereteit is. A tervezés és implementálás közben szakmailag igen sokat fejlődtem és igen sok érdekes részproblémákkal sikerült megismerkednem.

6 Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani:

- Dr. Végh Jánosnak, amiért lehetőséget adott számomra, hogy szakdolgozatomat nála írhassam meg.
- Tanáraimnak, amiért biztattak és tanítottak.
- Szüleimnek és nagyszüleimnek, amiért biztattak és támogattak a tanulmányaim kezdetétől fogva.
- Páromnak, Pusztai Magdolnának, aki mindig mellettem állt bátorított, segített.

Irodalomjegyzék

wxWidgets: , wxWidgets, , www.wxwidget.org
Autoconf: , , , http://www.gnu.org/software/autoconf/manual/autoconf.pdf
Kdevelop: , , , http://docs.kde.org/development/en/kdevelop/kdevelop/
Automake: , , , http://www.gnu.org/software/automake/manual/automake.pdf
GNU Make: , ,
A C++ Programozási nyelv I.: Bjarne Stroustrup, A C++ programozási nyelv, 2001
A C++ programozási nyelv II.: Bjarne Stroustrup, A C++ Programozási nyelv, 2001
GNU Gcc: , , , http://gcc.gnu.org/onlinedocs/gcc-4.3.3/gcc.pdf
Doxygen manual: , , , http://www.stack.nl/~dimitri/doxygen/manual.html
SVN book: , , , http://svnbook.red-bean.com/en/1.5/svn-book.pdf
Dia diagram: , ,
PMUEOP: Eric Gamma; Richard Helm; Ralp Johson; Jhon Vilissides, Programtervezési Minták Újrahasznosítható elemek objektumközpontú programokhoz, 1995

Autoconf dokumentáció

http://www.gnu.org/software/autoconf/autoconf.html

Gcc fordító dokumentáció

http://gcc.gnu.org/onlinedocs/

Gcc fordító dokumentáció

http://www.cs.utah.edu/dept/old/texinfo/gcc/gppFAQ.html

Kdevelopp fejlesztői környezet

http://kdevelop.org

wxWidget toolkit dokumentáció és források

http://wxwidgets.org

wxWidget toolkit dokumentáció és források

http://wiki.wxwidgets.org