Szakdolgozat

Uzonyi László

Debrecen

2007

Debreceni Egyetem

Informatika Kar

Gépjárműkövető rendszer fejlesztése

Témavezető:

Kollár Lajos

számítástechnikai munkatárs

Készítette:

Uzonyi László

programozó matematikus

Debrecen

2007

Tartalomjegyzék

1 Bevezetés

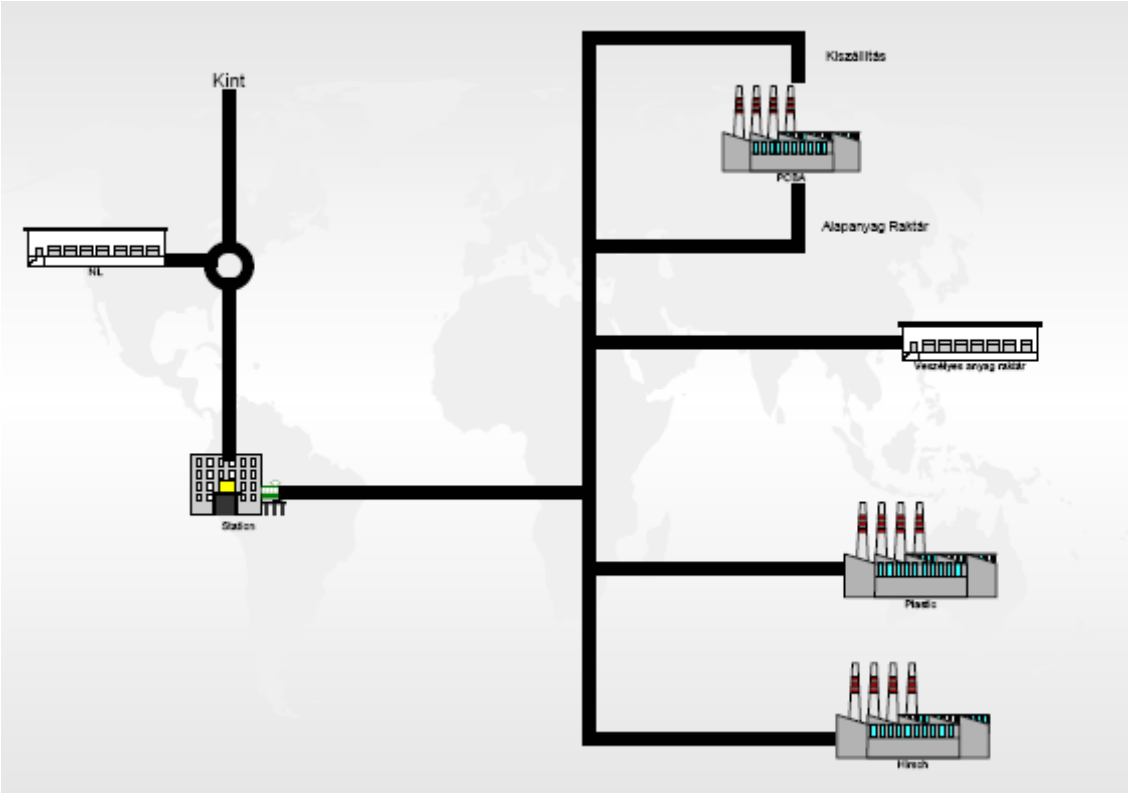
1.1 A szakdolgozat témája

Szakdolgozatom témája egy valós, ipari környezetben felmerülő biztonsági problémára elkészített megoldás ismertetése.

Jelenlegi munkahelyem, egy amerikai központú multinacionális cég, mely a világon közel 100 telepen folytat (bér) termelést, köztük a 4 magyarországi telephellyel. Mindegyik magyar telepen jelentős napi szintű kamion és egyéb gépjármű forgalom van, melynek nemcsak adminisztrálása, hanem biztonsági szempontból történő nyomkövetése is számos problémát vetett fel. Ezek közül néhány megoldásának elősegítésére készítettem a Truck Tracer 1. 1 nevű alkalmazást, melynek fejlesztési állomásait és az alkalmazott megoldások ismertetését szeretném kifejteni a továbbiakban.

Előjáróban, az alkalmazás maga Java nyelven lett fejlesztve és kihasználja a Java Web Start technológia által nyújtott előnyöket főleg a telepítést és verziókövetést érintő területeken. Az alkalmazás adatbázisa egy Microsoft SQL 2000 Serveren felépített adatbázis, melyen lévő tárolt eljárások, felhasználói függvények, nézetek valósítják meg a rendszer „üzleti logikáját”. A rendszer logikailag elkülönülő funkciói jól szeparáltak, melyek felhasználókhoz rendelhetők az igényeknek megfelelő kombinációban. Tekintettel a szóba jöhető igényekre, a rendszer az egyes funkciókat tekintve valós időben skálázható a fizikai folyamatok csekély mértékű változásával. A legfőbb szempontok, melyek meghatározták a fejlesztés menetét az első sorban a jelen fizikai folyamatokhoz leginkább illeszkedő alkalmazás megfejlesztése volt, illetve a minél kiterjedtebb adatgyűjtés, és a rögzített információk visszanyerhetőségének a valós igényekhez való igazítása.

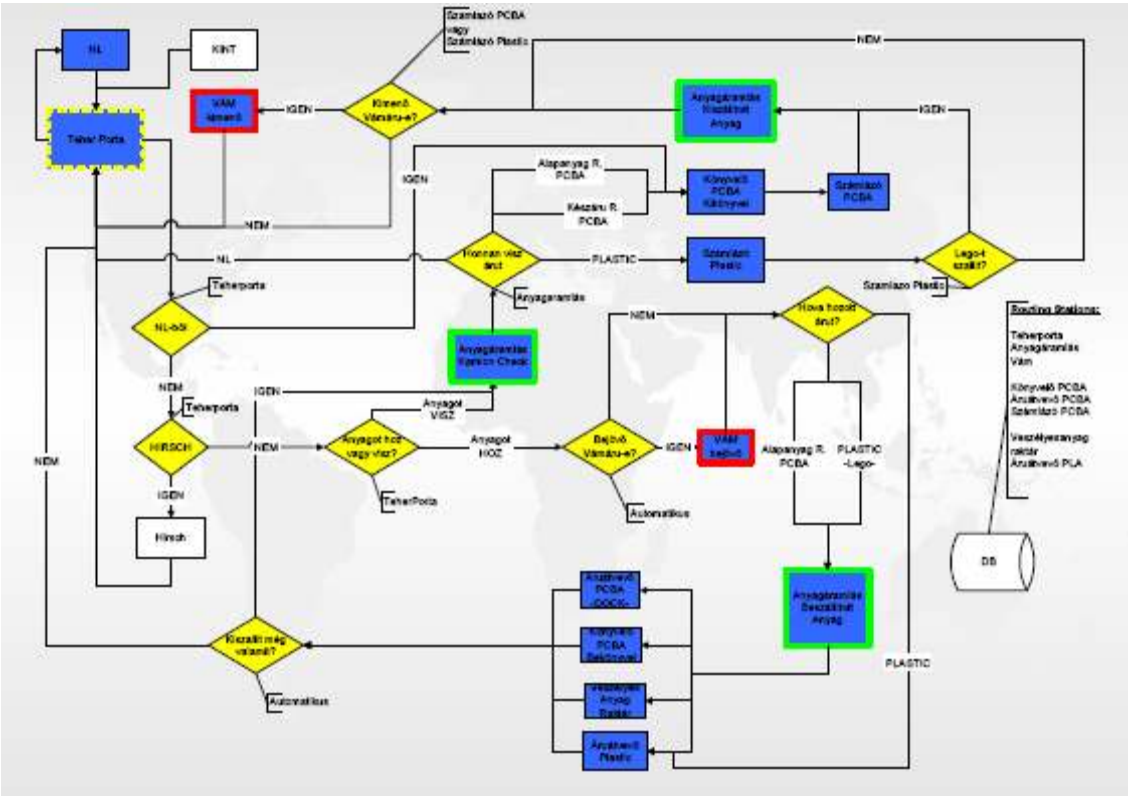
1.2 Régi rendszer



A fenti ábrán a nyíregyházi telephely vázlatos rajza látható. Minden nemű, nem direkt személyforgalom a teherportán keresztül folyik. Itt történik a minden érintett jármű ki - és beléptetése. Az ábrán ezt a rész jelzi. Ebbe a csoportba tartozik minden teherforgalmat megvalósító kamion, kishaszon-gépjármű, továbbá futárszolgálat és italautomatás szerviz kocsi, stb. Itt kerülnek rögzítésre a járművek információi, és egyéb információk, amelyek a későbbiekben részletezve lesznek. A teherportán belül 3 fő gyáregység van, és egy közös raktár az un. veszélyes anyagraktár. A teherportán kívül is van egy, logikailag a céghez tartozó külső raktár, amely az alkalmazás fejlesztésének elején még jelentős logisztikai szereppel rendelkezett. Jelentőssége az alkalmazás szempontjából tervezéskor meghatározó volt, ugyanis a megfelelő anyagmozgás ellenőrzése végett képessé kellett tenni a rendszert hasonló „kivételes” események lekezelésére is.

A tényleges fejlesztés megkezdése előtt az akkor használt folyamatok feltérképezése első ízben egy, a lentebb látható folyamatot vázolt fel. A rendszer

tulajdonképpeni működése a napi átlagot tekintve meg lehetősen egyszerűnek bizonyult, de a szóba jöhető kivételes események nagymértékben bonyolították a folyamatokat.

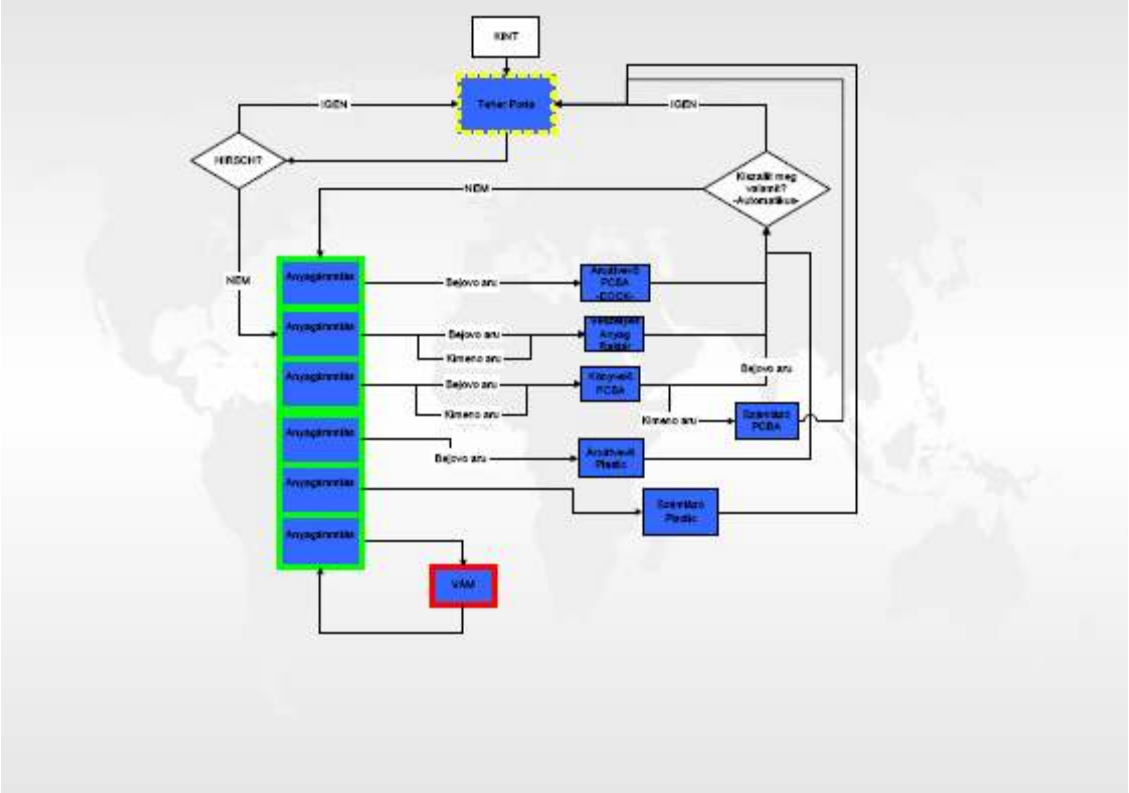


Az ábra tüzetes átvizsgálása nélkül is látható, hogy a rendszer sokparaméteres, sok elágazási lehetőséggel. Megfelelően sok kiindulási információt követelt volna meg egy erre épülő alkalmazás annak érdekében, hogy szemantikusan illeszkedjen a már meglévő folyamatokhoz. Az elsődleges okok, melyek amellett szóltak, hogy e rendszeren ne alapuljon fejlesztés, az volt, hogy amennyiben a rendszer strukturálisan változik, az generálisan az egész alkalmazás filozófiát érintené. A kék négyzetek jelölik azokat az állomásokat, amelyek fizikailag is egy helyhez, de legalább egy-egy személyhez kötődik, mint pl. vám, készáru raktár, stb. Ha változik az egyes „állomások” közti reláció, az, az egész ábra áttervezésével járhat.

Nyilván egy fejlesztés sem a jelenlegi rendszer analízisével kezdődik, hanem egy probléma specifikációval, ami ez esetben is természetesen az első lépés volt, viszont a pontos specifikáció megkövetelt egy 0. lépésben elvégzett felmérést. A kezdeti

specifikáció ugyan megvolt, de nem biztosított kellő információt ahhoz, hogy egy fejlesztést alapozni lehessen rá.

Következő lépésben a megfelelő területek vezetőinek és felelőseinek bevonásával elkészült a lentebb látható „normalizált” ábra, mely szemantikusan fedi az előbb felvázolt rendszer funkcionalitását, de már rendelkezik azzal az előnnyel, hogy egy megfelelően átgondolt alkalmazás, mely erre épül, képes teret adni esetleges későbbi változtatásoknak.



Az ábra alapján - amely megfelel a régi rendszernek, de illeszkedik az alkalmazás vázához – mikor belép egy jármű, azt a funkcionálisan jelentős információt kell csak specifikálni, hogy a jármű belépés után alá lesz-e rendelve az Anyagáramlási csoport irányításának és ellenőrzésének vagy sem. Amennyiben a jármű a HIRSCH nevű gyáregységhez kér belépést, az anyagáramlás nem ellenőrzi. Hasonló az eset minden szervizmunkára beengedett jármű esetében is. Ekkor belépést követően nem kerül rögzítésre semmilyen információ, csak majd a kilépéssel kapcsolatosak. Amennyiben az anyagáramláshoz kerül a jármű beléptetés után, úgy a további út információkat az anyagáramlás specifikálja. Ezzel biztosítva van a megfelelő dinamizmus járművenként az

állomások számára és sorrendiségére vonatkozóan. Ha az anyagáramlás által definiált útvonal teljesítve lett, a jármű kiléphet a teherportán.

1.3 Régi, „fizikai" rendszer

Biztonsági szempontból a folyamat lényeges pontjai a teherporta és a belső állomásokon történő eseményekre, és azok dokumentálására, követésére összpontosult. A régi megoldás mentes volt minden elektronikus alapú megoldástól, pusztán papír alapon történt minden nemű információ rögzítése. Ennek fő előnye mindössze a saját kezű aláírás hitelességére korlátozódott. Valójában viszont ezen érv mellett számos biztonsági rést nyitott a technológia. Az adatokat előírás szerint 5 évig szükséges megőrizni. Pusztán papír alapú megoldás esetén ennek napi 10 kamionos forgalom esetén is jelentős helyigénye van minden nemű kísérő lapok tárolásához. Nagyvállalati környezetben a hely persze másodlagos kérdés, viszont a mindenféle kísérő lapok rendezése, információk visszakereshetősége, statisztikák készítése már jelentős problémák forrása. A gyakorlat pedig azt mutatja, hogy az adatvesztés és az adatok hitelessége sem nyújtott kellő megbízhatóságot. A papíron tárolt információk hitelességére nem volt semmilyen jellegű biztosíték, illetve a papír megsérülése, esetleges eltűnése esetén az információvesztés nem volt visszaállítható.

A rendszer hatékonyságát rontotta továbbá az információk hozzáférhetőségének problémája is. Nem minden állomás számára álltak rendelkezésre a megfelelő információk tekintettel a jármű rendszámára, fuvarozójára, sofőrére, stb. Pl., ha egy kamion beérkezett idő előtt, míg engedélyt nem kapott, addig a parkolóban várakozott. Amint valamelyik állomás számára fontossá vált a kamion beérkezése, telefonáltak vagy a teherportára vagy az anyagáramlásnak, hogy bent van-e már a kamion, ha igen, hol, stb. Az információk csak közvetett módon, és időben rendszerint jelentős csúszással voltak elérhetőek. Nem egy esetben ez nem pusztán biztonsági, hanem termelés kritikus problémává is vált (termelés leállás, stb.). Hasonló jellegű probléma lépett fel mikor az illető jármű, kamion, távozni készült, de a teherportán nem álltak rendelkezésre

információk a kamion kiengedhetőségének feltételeiről. Nem volt elkerülhető az esemény, hogy egy kamion elhagyta a gyár területét, úgy hogy kihagyott egy vagy több szükséges állomást - hacsak nem járt utána a teherportán szolgálatban lévő őr telefonálással a jármű kiengedhetőségének feltételeinek. Ilyen és hasonló jellegű információáramlási kérdések maradtak nyitottan lévén, hogy nem volt megoldható az időben és állomások tekintetében- helyben homogén információ hozzáférés.

2 A rendszerfejlesztés folyamata

2.1 Áttekintés

A fentebb említettek alapján, és az megfogalmazott specifikáció alapján készült el először a Truck Tracer 1. 0 béta, majd Truck Tracer 1. 0 változat, mely megvalósította az elektronikus alapon történő jármű adminisztrációt, majd az 1. 1-es változat, mely már nem csak a szállítmányt fizikailag tartalmazó pótkocsi, hanem a pótkocsi és vontató párost is képes kezelni.

Az alkalmazás által elsősorban a következő megoldások kerültek megvalósításra:

- Biztonság fokozása mind a rögzítésre került információk védelmének területén, mind az információk rögzítésének folyamatában, és annak hitelességének területén.
- A rögzített információk visszanyerésének pontossága és szemantikus csoportosítása nagyobb hatékonysággal végezhető.
- Információtárolás gazdaságossá vált és az elektronikus archiválás által az információk adatvesztés nélkül, pontosan megőrizhetők.
- A napi szintű belső folyamatok átláthatóbbá, és gördülékenyebbé váltak. A megfelelő információk könnyebben és minden érintettnek egyaránt hozzáférhetőbbé váltak.

2.2 Architektúra

Az alkalmazás felépítését tekintve 3 rétegből áll. Legalsó szinten maga az információk tárolását megvalósító adatbázis áll. Második szinten a rendszer fő mechanizmusait – „üzleti logikát” – megvalósító működés áll, melyek tárolt eljárásai az adatbázisnak. Harmadik szinten maga az adatbázishoz közvetlenül csatlakozó kliens program áll.

2.3 Adatbázis-tervezés

Az adatbázis maga egy Microsoft 2000 SQL Serveren lévő adatbázis. Funkcionálisan ugyanúgy megfelelő lehetne, vagy lehetett volna akármely más adatbázis szerver, mely képes adatbázisban tárolt eljárások, függvények futtatására (mint pl. Oracle, stb.), de a rendelkezésre álló erőforrások tekintetében az MS SQL 2000 bizonyult a legkézenfekvőbb választásnak.

Az adatbázis struktúra - mint táblák szervezése, kolláció megválasztása, mentési stratégia tervezése, indexek szervezése – megtervezése a fizikai (kód szintű) implementáció első lépése volt, miután a specifikáció alapján a megfelelő folyamatábrák, és form tervek rendelkezésre álltak. Ezek alapján a rendszert leghatékonyabban kiszolgáló tábla struktúra a következőképp készült el:

2.3.1 Pótkocsi tábla

- Egyedi azonosító. Minden egyes jármű előfordulás egyedi azonosítót kap. Így el lehet különíteni, és különálló „eseményként” kezelni az ugyanazon pótkocsinak (is) adott

be és kilépése között történő információit. Bármilyen művelet esetén, (információmódosítás, visszakeresés), megkönnyíti az információk hozzáférhetőségét adatbázis oldalon, mind kezelhetőség és elérési idő tekintetében. Az azonosító egy szám, melyet az adatbázis generál futó sorszámként.

- Rendszám azonosító. Hogy ne legyen redundáns adattárolás, a rendszámok egy külön táblában vannak nyilvántartva, és egyedi azonosítóval vannak megkülönböztetve. Mivel egy adott rendszámú pótkocsi, vagy vontató többször is előfordulhat mind a telepen, mind az adatbázisban, így feleslegesen lenne letárolva ez az információ mindannyiszor, ahányszor az adott jármű belép.
- Fuvarozó azonosító. Hasonló megfontolásból egy külön, a fuvarozókat nyilvántartó táblának egy rekordjának egyedi azonosítója kerül csak regisztrálásra ebben a mezőben.
- Aktuális állomásazonosító. Egy, az *állomások* tábla valamely rekordjának azonosítója szerepel itt, jelezvén, hogy a jármű melyik állomáson áll aktuálisan. Egyfajta státusz információ.
- Aktuális vontató azonosító. Annak a vontatónak az egyedi azonosítója áll itt (ha van olyan), amelyhez jelenleg tartozik a vontató, amelyikkel jelenleg fizikailag össze van kapcsolva. Lehet *null* a mező értéke abban az esetben, ha a vontató és a pótkocsi szét lettek kapcsolva.
- Parkoláskor kapott vontató azonosító. Ha a szerelvény belépés előtt várakozott a parkolóban, akkor a parkoláskor rögzített vontató azonosítója szerepel itt, különben *null*. Előfordulhat az az eset, hogy a pótkocsi megérkezik a parkolóba, de pl. a sok várakozás miatt már a parkolóban vontatót cserél.
- Belépéskori vontató azonosító. A szerelvényhez belépéskor rögzített vontató azonosítója. Minden esetben szerepelnie kell, értelemszerűen.
- Kilépéskori vontató azonosító. A szerelvény kilépésekor rögzített vontató azonosító. Minden esetben kötelező.

2.3.2 Vontató tábla

- Egyedi azonosító. Minden egyes vontató előfordulás a pótkocsiéhoz hasonlóan új, egyedi azonosítót kap. Minden egyes vontató előfordulás egy újabb adatbázisbeli rekord, ennek megfelelően pedig különböző tulajdonságokkal rendelkezhet.
- Típusazonosító. Egy vontató típusokat nyilvántartó tábla egy rekordját hivatkozza. A vontató típusáról szóló információk vannak itt tárolva (pl. MAN, IVECO, stb.).
- Rendszám azonosító. A pótkocsi rendszám tárolására használt megoldás van itt is alkalmazva. Mind a pótkocsi, mind a vontató rendszámok rendszámai a *rendszám* táblában vannak letárolva.
- Sofőrazonosító. A *sofőr* tábla egy rekordját hivatkozza, mely a vontatót vezető sofőr információit tartalmazza. Nyilván egy sofőr többször is megfordulhat a telepen, és nem mindig ugyanazokkal a járművekkel, így célszerű ezt az információ halmazt is külön egységként kezelni és tárolni.
- Fuvarozó azonosító. A *fuvarozó* tábla egy rekordját hivatkozza, jelezvén, hogy melyik fuvarozó társasághoz tartozik az érintett jármű.
- Parkolás dátuma. Lévén, hogy a vontató, mint logikai egység nem jelentős a telepen belüli forgalom irányítás szempontjából (nem kell mindenféle állomásokat bejárnia, stb.), így a vontató élettörténetével kapcsolatban csak a parkolás, belépés és kilépési dátumot és a megjegyzéseket szükséges rögzítenünk.
- Belépési dátum.
- Kilépési dátum.
- Megjegyzés. Parkoláskor, belépéskor és kilépéskor a felvett megjegyzések ide íródnak be, a be illetve kiléptető őr nevével együtt. Maximum 3 állomásról van szó, az előfordulható megjegyzések mérete megbecsülhető, és így egy mező elegendő a rögzítésre.

2.3.3 Rendszám tábla

Tartalmaz egy egyedi azonosítót – futó sorszám, és magát a szövegesen tárol rendszámot. A rendszámok külön táblába gyűjtésének főként normalizációs okai vannak. (3NF-re hozás miatt). Bizonyos rendszámok többször is előfordulnak az adatbázisban, melyek szükségtelenül foglalnának annyi helyet, ami magának a szöveges rendszám tárolásához szükséges. Ehelyett a járműveknél csak 4 bájt hosszú rendszám azonosító van feltüntetve, ami hivatkozza a rendszám táblában a közel tetszőleges hosszú (de 4 bájtnál minden esetben hosszabb) karakteresen ábrázolt rendszámot. Rekordszám tekintetében nagyméretű *pótkocsi* vagy *vontató* tábla esetében ez viszonylag nagy mértékű helymegtakarítást jelent.

2.3.4 Vontató típus tábla

- Egyedi azonosító. Adatbázis által generált futó sorszám biztosítja a rekordok megkülönböztethetőségét. Ez a mező van hivatkozva a *vontató* tábla által.
- Típus. Maga a vontató típus név.

2.3.5 Sofőr tábla

- Egyedi azonosító. Ezt a mezőt hivatkozza a *vontató* tábla.
- Sofőr neve.
- Személyi igazolvány szám.
- Telefonszám.

2.3.6 Szállítólevél tábla

- Pótkocsi azonosító. Ez a tábla elsődleges kulcsa, mivel minden pótkocsihoz legfeljebb 1 szállítólevél szám tartozhat.
- Szállítólevél szám.

2.3.7 Plomba tábla

- Plombaazonosító. Egyedi azonosító. Az adatbázis által biztosított futó sorszám.
- Pótkocsi azonosító. Egy pótkocsihoz több plomba is tartozhat
- Plomba szám. A plomba száma. A gyakorlatban egyedi azonosítót jelent, de a számok mellett egyéb karaktereket is tartalmazhat, ami a rekord elérési idejét hosszabbá teheti, így nem ez a mező képezi az elsődleges kulcsot.

2.3.8 Felhasználó tábla

- Egyedi azonosító. Futó sorszám.
- Felhasználó név.
- Létrehozás dátuma. A felhasználó regisztrációjának dátuma.
- Tartományi felhasználó név. Alkalmazás oldalon ez az információ azonosítja be a felhasználót egyedien. Ezzel az információval vannak közvetve a felhasználóhoz rendelve a jogok, és minden egyéb információ.
- Archív-e? Ha a felhasználói hozzáférés törlésre kerül, akkor a fizikai törlésre csak abban az esetben kerül sor, ha a felhasználó nem volt semmilyen hatással az adatbázis tartalomra (nem hagyott jóvá egy kamiont sem, stb..). Viszont minden olyan esetben, mikor a felhasználói információk az adatbázis más helyein is hivatkozva van, úgy a felhasználó csak archiválásra kerül-e mező értékének átbillentésével.

2.3.9 Állomások tábla

- Egyedi azonosító. Ezt hivatkozza az *útvonalak*, a *történet*, a *jogosultság* tábla. Futó sorszám.
- Állomás név.
- Módosítható-e. Azon állomások, melyek funkcionálisan elengedhetetlenek a rendszer működéséhez, azok nem módosíthatónak vannak jelölve. Ezeket nem lehet alkalmazás oldalról semmilyen módon törölni. Ezek rendszerint a *parkoló, teherporta be, teherporta ki, forgalomirányító, nem ellenőrzött,* és *kiengedett* nevű állomások.
- Archív-e? Ha az állomás már szerepel valamely jármű útvonal listájában, így már van rá érvényes hivatkozás, úgy a törlésnél csak logikailag lesz törölve az állomás, e mező státuszának átbillentésével. Nem módosítható állomást nyilván archiválni sem lehet alkalmazás oldalról.

2.3.10 Jogosultság tábla

- Felhasználó azonosító
- Állomásazonosító

E tábla funkcióját tekintve az állomások, és felhasználók összerendelését valósítja meg, ennek megfelelően az elsődleges kulcs a két mező együtteséből adódik. Amennyiben létezik a keresett felhasználó és állomást reprezentáló azonosító relációja e táblában, úgy az adott felhasználó jogosult az állomáson álló járművekkel kapcsolatos operációra.

2.3.11 Útvonal tábla

- Egyedi azonosító. Futó sorszám. A tábla a gyakorlatban az útvonal információk „átmeneti” tárolására szolgál, így a tábla soha nem ér el nagy méretet, viszont minden, benne létrehozott útvonal bejegyzésnek egyedi azonosítóval kell rendelkeznie. Tekintve, hogy járművenként átlagosan (gyakorlati tapasztalat alapján) 6-9 útvonal bejegyzés kerül létrehozásra, a futó sorszámok határértékének kitolása szükséges, a sorszámok 8 bájt hosszan kerülnek ábrázolásra, (míg a többi futó sorszám 4 bájtos hosszal rendelkezik).
- Pótkocsi azonosító. Melyik pótkocsihoz tartozik az útvonal bejegyzés.
- Állomásazonosító. Az útvonal bejegyzés melyik állomást jelenti.
- Kötelező-e. Kötelező ezen az állomáson való továbbhaladáshoz elektronikus jóváhagyás.
- Sorszám. A vontatóhoz definiált útvonal terv hányadik sorszámú bejegyzése az illető rekord.
- Felhasználó azonosító. Mind addig, míg az útvonal bejegyzés nincs teljesítve, ennek a mezőnek értéke *null*. Amint valaki (arra jogosult) jóváhagyja, a *felhasználó* táblában lévő azonosítója beíródik ebbe a mezőbe.
- Jóváhagyás dátuma. Értéke ugyancsak *null,* míg nincs jóváhagyás. Jóváhagyáskor a felhasználói azonosítóval és megjegyzéssel egyszerre az aktuális (MS-SQL szerver szerinti) idő kerül bejegyzésre.
- Megjegyzés. Jóváhagyáskor a pótkocsihoz fűzött megjegyzések kerülnek itt rögzítésre. A megjegyzés felvevője ugyanaz, akinek az azonosítója a felhasználó azonosító mezőben szerepel.

2.3.12 Történet tábla

Strukturálisan teljesen megegyezik az *útvonal* tábla felépítésével. Különbség csak a tábla funkciójánál fogva a tábla méretében és ebből következően az index struktúra

kialakításában van. Minden, a telepen jelenleg bent álló (vagy parkoló) jármű útvonal információi az *útvonal* táblában vannak nyilvántartva. Az *útvonal* táblában az útvonal létrehozásakor kap egyedi azonosítót minden útvonal bejegyzés, és minden útvonal bejegyzéssel kapcsolatos művelet és lekérdezés ebben a táblában kerül végrehajtásra. Lévén, hogy csak az aktuálisan bent álló jármű információk szerepelnek a táblában, a tábla által tartalmazott információk mennyisége nem rontja a rendszer teljesítményét az idők folyamán (az adatbázisba került járművek számának növekedése következtében). A tábla állandó, kis mérete lévén nincs szükség index újraszervezésére – mint ahogy nincs is index szervezve a táblára -, stb. az adatok gyorsan hozzáférhetőek. A rendszer egészét tekintve is, az útvonal tábla a legfrekventáltabban használt tábla.
Amint a jármű kilép a telepről, az útvonal információi archiválásra kerülnek azáltal, hogy átíródik minden, az *útvonal* táblában lévő rekordja a *történet* táblába. A pótkocsi aktuális állomását jelző mező a *pótkocsi* táblában átállítódik a "*kiengedve"* nevű (fiktív) állomás azonosítójára, ezzel jelezve, hogy a pótkocsi már archív, az információi a *történet* táblában érhetők el. A tábla a gyakori keresési igényeknek megfelelően 3 fürtözött indexet tartalmaz, melyek hetente egy éjszakai „job” keretein belül - mint karbantartási terv (Maintenance plan)- újra vannak szervezve a megfelelő performancia mutatók megtartása végett.

2.3.13 Fuvarozók tábla

A *rendszám* táblához hasonló megfontolásból a Pótkocsi és Vontató táblán kívül vannak letárolva a tényleges fuvarozó nevek is. A helymegtakarításon kívül az esetleges információk javítása (fuvarozó átnevezése) is csak egy rekord, egy mező értékének érintését jelenti, míg, ha rekordonként minden járműhöz explicit módon rögzítve lenne maga a fuvarozó szöveges neve, akkor egy érték frissítés meglehetősen sok rekordot érintene. Keresés esetén pedig indexelés nélkül meglehetősen sokáig tartana a bejárás, indexek használatával viszont szükségtelenül nőne a tábla mérete, és az indexek újraszervezése is időnként szükséges lenne.

2.3.14 Hiba tábla

- Hiba kód
- Hiba leírás

Minden, az adatbázisban elvégzett művelet tárolt eljárásokkal van megvalósítva. A műveletek elvégzése során a tárolt eljárásokban különböző események fordulnak elő, rendszerint valamely nem várt esemény. Sok esetben ezek a kívánt művelet végre nem hajtását eredményezik, melyekről szükségszerűen informálni kell a felhasználót. Például, ha be szeretné jelenteni a szolgálatban lévő őr a parkoló nevű állomáson a már előzőleg valaki által beléptetett kamiont, a bejelentés sikertelen lesz, mivel egy hasonló rendszámú jármű már bent áll a telepen. Ez esetben a felhasználó szöveges üzenetet kap a bejelenteni kívánt jármű jelenlegi státuszának megfelelően. Minden ilyen, és ehhez hasonló helyzetben a megfelelő hiba táblára hivatkozó hiba kód kerül kiválasztásra a tárolt eljárásban, melynek megfelelő üzenet lesz átadva az eljárást hívó alkalmazásnak. A tábla tartalma állandó, a rendszer telepítésekor lett feltöltve. Esetleges idegen nyelvre való áttérésnél adatbázis oldalon így elkerült szituáció, hogy minden eljárást át kelljen kutatni az esetleges kimeneti üzenetek lefordítása végett. Minden, az alkalmazás oldalnak átadott üzenet itt van tárolva, így elég pusztán itt módosítani.

2.3.15 Sablon tábla

- Sablonazonosító. Futó sorszám.
- Sablon neve
- Sablon leírása, rövid (maximum300 karakter) terjedelemben.

Lehetőség van előre definiált útvonal sablonok létrehozására, amely egy sorrendileg definiált állomás lista. E táblában csak a sablon neve, azonosítója és leírása van

nyilvántartva. A sablon azonosító segítségével van összeállítva a sablonhoz tartozó lista a *sablon tartalom* táblában.

2.3.16 Sablon tartalom tábla

- Sablonazonosító. A *sablon* tábla sablon azonosítóját hivatkozza.
- Állomásazonosító. Az *állomás* tábla állomás azonosítóját hivatkozza.
- Sorszám. A sablonban betöltött sorszám.
- Kötelező-e. Megmondja, hogy a sablon ezen eleme az útvonal listába kötelező vagy nem kötelező jóváhagyás szerint kerüljön be.

A sablonazonosítóval azonosított sablonhoz tartozó állomások azonosítói vannak alapvetően tárolva, amelyek a többi kiegészítő információval arra szolgálnak, hogy egy tárolt eljárás segítségével az itt, a sablon listájában szereplő állomások felvételre kerüljenek a megfelelő jármű aktuális útvonal tervéhez.

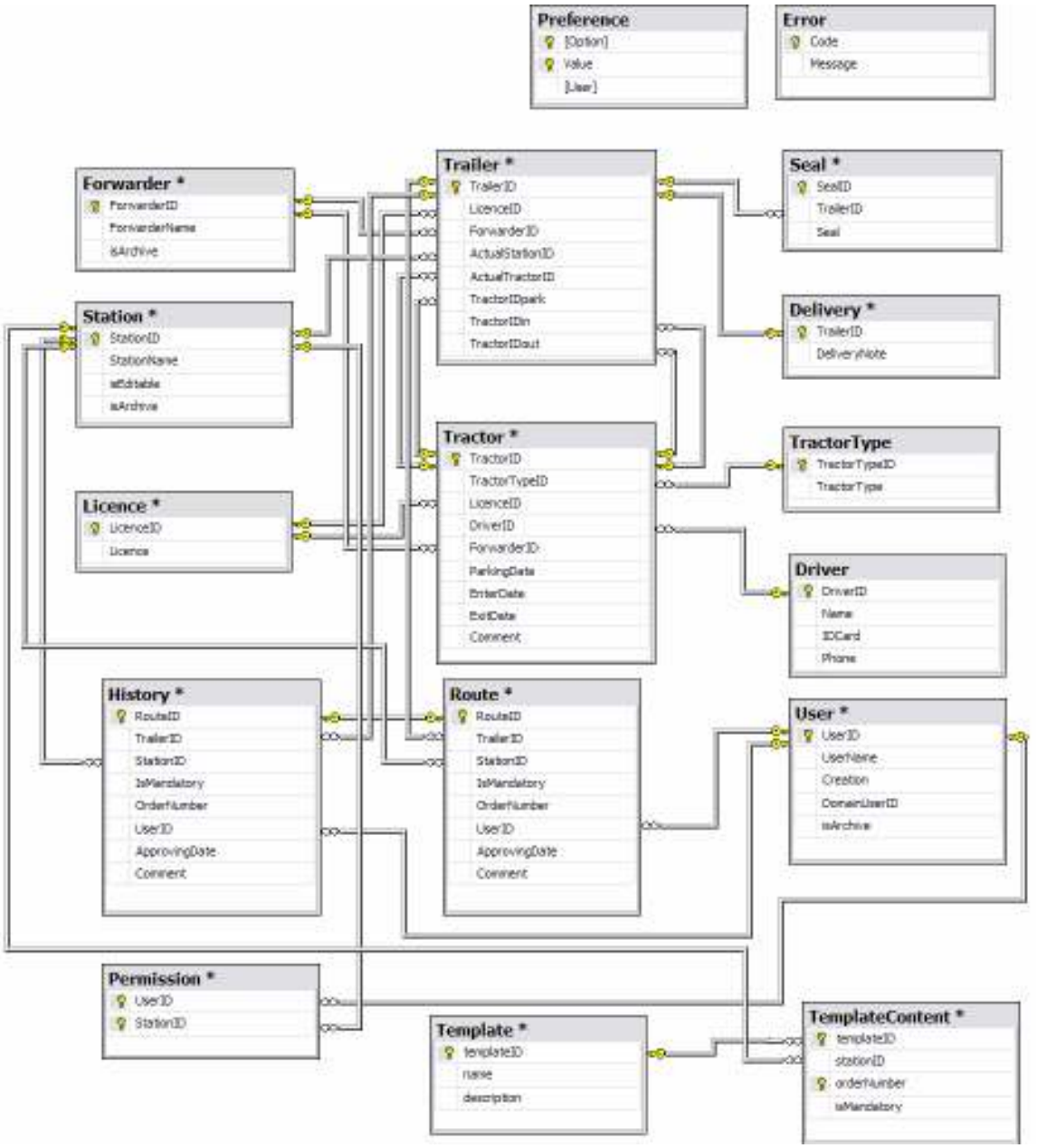
2.3.17 Beállítások tábla

- Kulcs, egy szöveges azonosító.
- Érték, egy szöveges érték. A kulcs és érték képezi a tábla elsődleges kulcsát.

Olyan információk tárolása történik e táblában melyek kliens futásához szükséges beállításokat jelent, de lokális tárolására nincs lehetőség, illetve nincs értelme. Pl. (Azoknak a felhasználó azonosítója, akik ADMIN jogosultsággal rendelkeznek. A *teherporta ki* állomásnál az LPT port konfigurációja.)

2.3.18 Adatbázisstruktúra-diagramm

Ezek alapján az elkészült adatbázis-struktúra diagrammja a következőképpen néz ki (az ábrán, mint ahogy a valóságban is az adatbázisban használt mezők nevei angolul vannak bizonyos vállalati konvenciók miatt):



2.4 Tárolt eljárások

A dolgozat jelen terjedelmi korlátai nem adnak lehetőséget, hogy akár csak vázlatosan is ismertetésre kerüljön az elkészült 77 tárolt eljárás, 14 nézet és 11 felhasználói függvény, így csak az alapvető fejlesztési koncepciók kerülnek tárgyalásra.
Ahogy említve is volt, a rendszer műveletei, adatbázisbeli változásai a rendszer üzleti logikáját implementáló tárolt eljárások segítségével vannak megoldva.
E megoldás mellett való döntés főbb okai közt az első az volt, hogy a kliensalkalmazástól lehetőleg legyen függetlenné téve az adatbázisbeli operációk implementációja. Sok alkalmazás oldali változás esetén az alkalmazásban kódolt SQL, illetve T-SQL parancsok meg lehetősen áttekinthetetlenné tennék az alkalmazást, ami leginkább a fejlesztést és az esetleges módosításokat nehezítenék meg.
Másik okként jelentkezik az a szükségtelenül nagy mennyiségű hálózati adatforgalom, amit a kliensalkalmazás oldalon implementált üzleti logika kívánna meg az alkalmazás és az adatbázis között. Gondoljunk bele, hogy egy viszonylag kicsi eredmény halmaz átvitele x mennyiségű hálózati adatforgalmat generál, míg ha az eredményhalmaz nagyméretű táblák értékhalmazainak bizonyos szemantikai szűréséből áll elő, ami nem minden esetben oldható meg egy-két soros SQL parancsokkal, akkor indokolatlanul nagy mennyiségű adat kerülne átvitelre. Ugyanakkor meglehet, hogy a forgalmazott adatok csak töredékére lesz szükség.

Figyelembe kell vennünk azt a paramétert is, hogy míg szerver oldalon egy meglehetősen erős hardver áll rendelkezésünkre mindenféle művelet elvégzésére, addig kliens oldalon néha elavult hardveren fut a JVM (*Java Virtual Machine*), ami miatt a felhasználói élmény jelentősen romlik, míg a szerver erőforrásai nagyon kis mértékben kerülnek kihasználásra. Ebből a megfontolásból lettek az információ transzformációs műveletek az adatbázis szerver erőforrásaira bízva. A gyakorlati tapasztalat szerint az elkészült alkalmazás szerver oldalon 2% - 10%-os átlag erőforrás kihasználtságot jelent a processzor és a memória számára. Eltekintve az egyes *történetben* táblából való lekérdezésektől, ez elég kis terhelést jelent a szerver számára, amellett, hogy a kliensek nincsenek így leterhelve, az erőforrásaiknak csak kis része van lekötve az alkalmazás által. Így a rendszert teljesítmény hangoltság tekintetében optimálisnak nevezhetjük, és

összevetve az előző szemponttal, tekinthetjük ez utóbbi döntéseket a teljesítményhangolási lépésnek.

Ezeket mérlegelve tehát, kerültek megfejlesztésre a tárolt eljárások, és az azokat kiszolgáló nézetek, függvények. Az általánosan követett koncepció szerint az adatbázisbeli operációkat megvalósító tárolt eljárások tranzakció alapúak. A tárolt eljárás elején egy új tranzakciót nyitunk, majd a sikeres operáció végén véglegesítjük a valós adatbázis tartalomban a tárolt eljárás által generált változásokat. Legfontosabb oka ennek a koncepciónak az, hogy biztosítsuk minden esetben a szemantikusan is helyes adatbázis tartalmat. Ritka eset, de előfordulhat, hogy valamely szerver oldalon bekövetkezett vagy a szerverre kiható probléma miatt az eljárásunk megreked, megakad. Ha nincs helyes tranzakció-kezelés, úgy az implicit tranzakciók elve miatt minden elvégzett T-SQL parancs végrehajtásra kerül, a nem elért SQL parancsok pedig nem. Ez meglehetősen nagy problémákhoz vezethet az adatbázis tartalmát érintően. Valós környezetben gyakrabban előfordul, hogy a kliens oldalon, vagy hálózati probléma miatt a kliens és a szerver kommunikációja megszűnik egy tárolt eljárás futása közben, és az említettel hasonló jellegű probléma állhat elő. Míg tranzakció elvű feldolgozásnál, egy kivételes külső esemény, vagy akár belső esemény kiváltódásakor a le nem zárt tranzakciók nem kerülnek jóváírásra. A felhasználói művelet sikertelen lesz, de megőriztük az adatbázis szemantikus konzisztenciáját. Meglehet, ezen a viselkedésen egy adatbázis beállítással MS-SQL környezetben változtathatunk, de alapbeállítás szerint ez a viselkedés rendelődik minden adatbázishoz.

Programozás technológiai előnyt is szerezhetünk tranzakciók használatával. (Explicit) tranzakciók használata nélkül egyes adatbázisban előállt felhasználói kivételek (szemantikus értelemben) lekezelése meglehetősen sok programozói munkával jár, mindenféle vizsgálatok, lekérdezések, és elágazások használatát megkövetelve sok esetben. Tranzakciók használatával már az első SQL utasítással elvégezhetünk módosításokat, beszúrásokat, melyeket egy általános esetben nem várt esemény, vagy állapot előfordulásával visszagörgethetünk. Ez a filozófia jelentősen könnyítheti a programozás folyamatát, de nagy körültekintést igényel. Tekintve egy olyan esetet, mikor egy elvégzendő update művelet rekordok nagy mennyiségét érintené, egy visszagörgetés

(*rollback*), illetve a változások a tranzakció végéig történő nem jóváírása meg lehetősen nagy erőforrás lefoglaltságot jelenthet, főleg memóriakezelési oldalon (sok lapozás, és az ehhez tartozó CPU műveletek). Amennyiben ilyen helyzetek jósolhatók, és ragaszkodunk a tranzakciók használatához, úgy minden ponton, ahol nagy mennyiségű adatok már teljes biztonsággal jóváírhatók, zárjuk le a tranzakciót, és kezdjünk újat. Sajnos az MS-SQL által használt T-SQL programnyelv nem biztosít az Oracle-ben használt köztes mentési pontokhoz hasonló eszközöket, így csak egész tranzakciókban gondolkozhatunk. Fontos még megjegyezni, hogy MS-SQL környezetben, ha egy tranzakció alatt meghívunk egy másik tranzakció elvű tárolt eljárást, akkor a hívó tranzakciónkat a meghívott tárolt eljárás befejezése után nem görgethetjük vissza. Külön tranzakciós egységet képeznek az egyes tárolt eljárások, melyek ilyen módon nem ágyazhatók egymásba.

Az adatbázis műveletek teljesítmény hangolása érdekében igyekeztem minden elvégzendő műveletet úgy implementálni az eljárások által, hogy a tárolt eljárás megkapja az alkalmazástól a lekérdezés vagy a módosítás által érintett rekordok elsődleges kulcsait, így redukálva a tárolt eljárásban elvégzendő keresések számát. Megvalósítása ennek a megoldásnak nem igényel igazán plusz programozói munkát, csak a fejlesztés megkezdése előtti körültekintést. Rendelkezésére kell bocsátani az alkalmazásnak a legtöbb lekérdezéskor az elsődleges kulcsokat, vagy a megfelelő indexekben szereplő mezők értékeit, hogy az alkalmazás oldalon született döntés alapján majd minél gyorsabb meg lehessen fogni a kívánt rekordokat. Programozás biztonsági okokból, annyiból jelent ez előny, hogy bonyolult keresési feltételek megfogalmazása tárolt eljárás oldalon is nagyobb gondot jelenthet, és erőforrás igénye is sokkal nagyobb, illetve egyedi azonosítók használatával kisebb a valószínűsége a pontatlan lekérdezésekből adódó rossz eredményhalmaznak. Ilyen és hasonló T-SQL problémák optimalizálási teendőinek elvégzésében jelent segítséget a Microsoft által biztosított Query Analizer alkalmazásának Display Estimated Execution Plan nevű modulja. Ebben lehetőségünk van egy kiválasztott lekérdezés, tárolt eljárás, vagy egyéb SQL művelet végrehajtási tervét megtekintenünk, feliratozva a becsült erőforrás lefoglaltsággal, és művelet igény hányadosokkal lépésenként. A modul használatával segítséget kapunk,

mely főleg az index struktúra, lekérdezések és egyéb optimalizációs témákban segít a fejlesztésben. Rendelkezésünkre áll még ugyan egy erre épülő Index Tunning Wizard nevű modul is, de az komolyabb, és főleg a gyakorlatban megvalósuló használattal nem képes reálisan számolni, ezért nem minden esetben javasolt teljesen megbízni az ajánlásában.

2.5 Adatbázis rendelkezésre állás

Mind az adatbázisra épülő alkalmazás, mind az adatbázis felépítésénél lényeges terület a rendelkezésre állás biztosítása. Maga a rendszer nem lenne önmagában egy kritikus, és nagy rendelkezésre állást igénylő rendszer, viszont ha megköveteljük, hogy minden jármű csak a megfelelő elektronikus adminisztráció után léphessen tovább az útvonal listában, vagy léphessen be a telepre akkor egy esetleges várakozás a telep működésére is kihathat, ami már kritikusnak mondható. Adatbázis oldalon ennek a kiküszöbölésére első sorban azzal léphetünk, hogy egy esetleges adatbázis „katasztrófa" esetén biztosítjuk az adatbázis gyors helyreállásának módját, mint például, ha a szerver hardveresen meghibásodik. Habár nem teljesen az adatbázishoz tartozó téma, de a mai adatbázisok futtatására dedikált szervereket RAID technológiával szervezett merevlemez csoportokba szervezik. Az általános ajánlás szerint az adatbázis szerverek merevlemezeit célszerű RAID 1 technológiával felfűzni, amit hívnak „tükrözésnek" is. Ennek során több lemez tartalmazza ugyanazt a fizikai információ struktúrát és tartalmat. Egymásnak „tükörképei". Hardveres RAID megoldásnál – szervereknél kizárólag hardveres jöhet szóba – a RAID kártya biztosítja, hogy a rendszer számára transzparens módon működjön a merevlemez hozzáférés, mind idő, mind egyéb tekintetben. Ha valamely merevlemez meghibásodik, működésképtelenné válik, a szerver futása közben lehetőség van a meghibásodott merevlemezt eltávolítani, kicserélni. Az újonnan behelyezett merevlemezre rövid időn belül „rátükröződik" a többi merevlemezen lévő tartalom, így nem történik effektív szolgáltatás kiesés már a logikai adatbázis számára sem. Számolni kell azonban olyan esetekkel, amikor nem helyreállítható „ilyen egyszerűen" a

keletkezett probléma. Az MS-SQL lehetőséget ad különféle mentési metódusok elvégzésére, amelynek helyes megválasztása egy újabb 9-essel növeli a rendelkezésre állási mutatót (99.9...%). A Truck Tracer adatbázisának becsült méretbeli növekedése a tábla definíciók (rekordok mezőinek típusa, mérete, indexek mérete stb.) és a járműforgalom figyelembevételével éves szinten 20-40 Mb. Ez kisméretű adatbázisnak számít és számít majd még évek multával is. Egy 300Mb méretű adatbázis mentése sem tart tovább 5 -10 percnél legszélsőségesebb esetben sem. Ezt figyelembe véve a mentési stratégia úgy lett megállapítva, hogy minden nap teljes mentés készül az adatbázisról, melyek szalagos mentési egységen más napi mentésekkel együtt elszeparáltan egy másik épületben helyezkednek el. Ez utóbbi kritérium már nem csak ajánlás, hanem az utóbbi időben bevezetett ISO szabvány is egyben.
Nagy adatbázisok (Gb és a fölötti) esetén a terhelés, performancia kritériumoktól függően célszerű un. differenciált – különbségi mentéseket készíteni , vagy a tranzakciós log-ról készíteni mentést esetleg a nap több szakában is akár, és csak ritkán teljes mentést. Nagy adatbázisok esetén célszerű főleg a teljes mentéseket olyan időpontokra időzíteni, mikor a rendszer rendelkezésre állását a mentés meglehetősen erőforrás igényes művelete a legkisebb mértékben befolyásolja.

A Truck Tracer esetén a rendszer rendelkezésre állásának fokozásában a megfelelő mentési stratégia kiválasztása mellett egy másik megoldásra is felkészítettem a rendszert, melynek alapja az MS-SQL által megvalósított un. replikáció. Olyan esetekben, amikor kettő vagy több adatbázis szerver áll a rendelkezésünkre, lehetőség van előállítani egyik adatbázis szerveren egy másik adatbázis szerver valamely adatbázisának replikált másolatát. Több célból is használható ez a megoldás, és különböző módon lehet implementálni, a célok függvényében. Az esetben, ha az éles adatbázis (például termelési adatgyűjtést kiszolgáló adatbázis) nagy terhelésnek van kitéve, vagy csak az esetleges lekérdezések, információ visszanyerések jelentenének nagy terhelést, úgy az éles adatbázis példány replikálásra kerül és a riportok, kimutatások, stb. lekérdezések így a replikált, nem éles adatbázisból dolgoznak, amely esetén nem kritikus a rendelkezésre állás, és egyéb performancia mutatók szinten tartása. A replikáció beállításaitól függően be lehet állítani, hogy milyen gyakran történjenek frissítések a replikált adatbázison.

Lehetnek napok, órák, percek, de lehet akár „on demand” is (azonnali). Ha un. „merge” (összefésülés) típusú replikációt választunk, akkor mindegyik adatbázis példányon életbe léptetett változtatás a megadott intervallum elteltével lefrissül mindegyik merge-el replikált adatbázison is. Az intervallum itt is lehet persze „on demand”. Ekkor elérhető, hogy több alkalmazás több adatbázisba dolgozva ugyanazt az információ halmazt használja. A Truck Tracer ezt a lehetőséget úgy használja ki, hogy alkalmazás oldalon az adatbázis kapcsolat felállásakor minden esetben egy „elsődleges” adatbázissal próbálja meg felépíteni a kapcsolatot. Ha ez nem sikerül, akkor megpróbálja a „másodlagos” adatbázissal is ugyanezt. Lehetőség lenne persze még harmadlagos, és további alternatívákra is, de az igény erre már csekély. Az elsődleges és másodlagos adatbázisok egymás replikáltjai, ilyen módon nem számít, hogy az alkalmazás melyik adatbázist használja. A felhasználó számára a működés transzparens, nem érzékelhető semmilyen különbség. Viszont ha adatbázis karbantartásra van szükség, vagy egyéb probléma lép fel akár az adott hálózati szegmensben, akár magával valamely adatbázissal, az alkalmazás csatlakozik a másodlagos adatbázishoz, és dolgozik tovább. Amint felállt az éles adatbázis ismét, a replikáció tovább folytatódik, az éles adatbázisban érvényesülnek a leállása óta a másodlagosban bekövetkezett változások, és a feladatokat a kliensalkalmazás példányok (újra) elindulásával visszakapja az éles adatbázis.

Általában az alkalmazások, ha nincsenek is felkészítve ilyen működésre, a merge, de egyáltalán más egyéb típusú replikáció is praktikus megoldás lehet, tekintve, hogy ha például, 5 percenként készítek egy adatbázisról (illetve annak változásairól) másolatot egy távoli szerveren, akkor ezzel előállt egy olyan adatbázisom, amely egy éles adatbázis másolata, ha úgy tekintjük, mentett példánya, melyen 5 percenként frissülnek az információk. Némely esetekben ez rendkívül jó mentési stratégiaként szolgálhat.

3 Alkalmazás réteg

3.1 Áttekintés

Mint korábban említésre került, a rendszer legfelső szintjén a kliens rétegen a Java nyelven fejlesztett alkalmazás áll. A programnyelv megválasztása nem csupán a fejlesztő jelen képességeinek figyelembevételével történt, hanem a programnyelv és maga a programnyelv által biztosított technológia által biztosított szolgáltatások mérlegelésével is. Vállalati környezetben egy biztonsági rendszer üzemeltetése során a rendszerrel szemben támasztott követelmények az állandó rendelkezésre állást, és a megbízhatóságot helyezik előtérbe. E mellett például a felhasznált erőforrások, és egyéb, költség oldalon amúgy jelentősebb paraméterek viszonylag kisebb prioritást élveznek.

3.2 Követelmények

A rendszer alkalmazás rétegével szemben támasztott követelmények közül első és legfontosabb a rendelkezésre állás biztosítása. A gyakorlatban előforduló esetek egy részében a szolgáltatás szünetelésének oka, a hálózati problémákból és karbantartásokból adódik. Az alkalmazás fejlesztésekor sem az alkalmazásnak, sem a fejlesztőnek nagy befolyása nincs a már meglévő hálózati infrastruktúra felépítésére, bár az alkalmazás rendelkezésre állásának nagy fajsúlyú területe ez. Ideális esetben, tekintve a vállalati környezetet, feltételezhetünk egy jól szervezett, megfelelő kvalitású és nagy áteresztőképességű informatikai hálózati infrastruktúrát. Ez a rendszer szempontjából az alkalmazást futtató kliens számítógép és az adatbázis működtető szerver számítógép közötti kapcsolatot jelenti. A kliens és szerver közti informatikai kapcsolatban ideális esetben 100 Mbit/sec –os hálózati sebesség él, és a szükséges köztes hop-ok lehetőleg jó

minőségű switch-ek segítségével vannak megoldva. A hálózat rendelkezésre állásának biztosítása ilyen esetben technológiai oldalról a rendszerünk működésének szempontjából biztosított. Ezen felül minden, a rendszer működését befolyásoló hálózati esemény felügyelete az informatikai osztály felelőssége.

A gyakorlatban már a fejlesztés menetének tervezésekor figyelembe kell venni az alkalmazás több helyen történő hozzáférhetőségének biztosítását is. Az esetben, ha a rendszer kliens oldala a funkcionalitások függvényében külön alkalmazásokra van bontva, akkor az alkalmazás hozzáférhetősége már a telepítésnél is, de minden további esetben is jelentős plusz munkát jelent a rendszert üzemeltető informatikai osztály számára. Például, ha minden állomás kezeléséhez a megfelelő alkalmazásra van szüksége a felhasználónak, akkor egy olyan esetben, mikor jogosultságot kap egy másik állomás kezeléséhez is, egy informatikus kolléga néhány perces munkájára lehet szükség. Nem beszélve olyan esetekről, mikor a jogosultság elvételére kerülne sor, vagy az illető felhasználó nem fér hozzá a saját számítógépéhez, és szeretne egy másik kollégájának számítógépével dolgozni egy rövid időre. Ilyen és ezekhez hasonlóan számos kérdés és probléma merülhet fel, mely jelentős plusz munkát okoz, és a hozzáférhetőséget rontja. Ezen követelmények figyelembe vételével az alkalmazásnak rendelkeznie kell egy megfelelő funkcionális integráltsággal, ami biztosítja a megfelelő alkalmazás funkciók hozzáférhetőségét, illetve elrejtését egy azon alkalmazáson belül a jogosultságok függvényében. Továbbá az alkalmazásnak hozzáférhetőséget kell biztosítania különböző fizikai kliensekről úgy, hogy a funkcionalitások egyforma mértékben legyenek elérhetők bármely kliens számítógépről, minél gyorsabban – lehetőleg ne legyen szükség informatikus kolléga segítségére, ami sok esetben időveszteség.

A gyakorlati használat során az alkalmazásnak kellően intuitívnak és informatívnak kell lennie. Az alkalmazást használó felhasználók száma változó, és nem minden esetben van lehetőség a felhasználó oktatására. Legjelentősebb oka ennek főleg az, hogy a legtöbb felhasználó számára az alkalmazás nem kapcsolódik szorosan a munkájához, leginkább csak plusz munkát jelent számára adminisztrálni a járműveket a rendszerünkben, mind a mellett a sok teendő mellett, amit feltétlenül el kell végezzen. Például egy, a vámon dolgozó kolléga miután a rengeteg papír munkát elvégezte, ha csak 2 kattintás erejéig is, de plusz munka a járművek adminisztrálása, tekintve, hogy a

munkáját e nélkül is „elvégezte”. Emellett nincs is minden esetben kapacitás arra, hogy valaki állandóan oktassa a megfelelő kollégákat, ha meg van a lehetőség az oktatás elhagyására is. Meglehet, a Truck Tracer kliensalkalmazásból is elérhető az aktuális felhasználói dokumentáció, ami teljes körű leírást tartalmaz, de a valóságban a felhasználók túlnyomó többsége tartózkodik minden nemű dokumentációval való ismerkedéstől. Így lehetőség szerint a felhasználói felületnek (*GUI*), megfelelően intuitívnak, beszédesnek, és informatívnak kell lennie, hogy minden előképzettség nélkül is az alapvető dolgát képes legyen elvégezni a felhasználó nagy biztonsággal már az első alkalommal is.

Számos esetben előfordul, hogy a fejlesztés megkezdésekor specifikált igények megváltoznak, esetlegesen változtatni kell valamit, vagy ami jellemzőbb, valamilyen fejlesztést kell implementálni. Ennek leggyakoribb oka nem az elkészült alkalmazás gyengeségeiből adódik jellemző esetben, hanem a fizikai folyamatok valamilyen fokú megváltozása, ami szükségszerűen kihat az alkalmazás felépítésére is. Ez már egy telephelyen belül is jelentkezhet, viszont ha felmerülhet az igény esetleg más telephelyen való adoptálásra is, akkor ez igényli, hogy az alkalmazás maga jól paraméterezhető legyen, ezáltal némely opcionális funkcióval is fel legyen készítve több jósolható igény kielégítésére is. Ez természetesen nem csak az alkalmazás réteget illeti, ha a folyamat szinten valami változás történik, de ha az adatbázis struktúra – ami inkább kritikusabb – fel is van készítve előre bizonyos változásokra, az alkalmazáshoz való fejlesztés maga is időt vesz igénybe, nem beszélve az átállásról, bevezetésről. Már a rendszer specifikációjakor lehetett prognosztizálni olyan fejlesztési igényeket, amelyekre fel lehetett készíteni a rendszert egyfokú paraméterezhetőség által. Jelen esetben ilyen az állomások nevének, illetve számának konfigurálhatósága, mely által lehetőség van az éles rendszer futása közben is ha kell megváltoztatni szinte a teljes állomás lista szerkezetét, beleértve az állomások és a felhasználók közti relációk azonnali karbantartását. Hasonló szempontok szerint lehessen a fuvarozók listáját módosítani, felhasználók listáját karbantartani, illetve egyéb apróbb beállításokat elvégezni.

Néha azonban beérkeznek olyan fejlesztési igények is, amelyek már konkrét, esetleg kód szintű módosítást kívánnak meg, amit nem lehet, vagy nincs értelme paraméterezni. Ez a helyzet egy újabb problémát vet fel, miszerint milyen módon, és

mekkora átfutással lehet átállni a legújabb verziójú alkalmazásra? Milyen hosszú szolgáltatás kiesést eredményez (ami a rendelkezésre állást befolyásolja, illetve rontja)? Milyen módon lehet biztosítani a verziókövetést? És mi a garancia arra, hogy nem maradt valahol egy futó példány, vagy egy régi verzió még működőképesen? Ez utóbbi különösen fontos kérdés, tekintve, hogy egy verziólépés nem feltétlenül csak az alkalmazás oldalra hat, hanem az adatbázis tartalom szemantikus tartalmát is befolyásolja. Természetesen, egy, az adatbázis szerkezetét is érintő változásnál a szolgáltatás kiesés nem elkerülhető, de amennyiben azt nem érinti, lehetőséget kell biztosítani a szolgáltatás kiesés mentes átállásra.

Nem utolsó szempont a rendszer biztonsága, hozzáférhetőségének szabályozása. Követelmény hogy az alkalmazáshoz csak az arra jogosított személyek és csak az arra megfelelő módon férhessenek hozzá. Fontos terület a vállalati hálózaton belüli szabályozás is, de a legveszélyesebb területnek a potenciálisan ártalmas világháló minősül. Ilyen és hasonló rendszer biztonsági kérdésekre is választ kell, hogy tudjon biztosítani a rendszer.

3.3 Futtató környezet, telepítési technológia

Az utóbb említett problémára kínál a Java Web Start nevű technológiája megfelelően átfogó, és nagy biztonsággal alkalmazható megoldást főleg a telepítési illetve verziókövetési problémákra.

Nagy vonalakban, a technológia legáltalánosabban használt funkciója, a mindössze http protokoll alapon történő telepítési mechanizmus, amely a verziókövetést is biztosítja. A Truck Tracer által megvalósított telepítési mechanizmus a Web Start által kínált megoldást alkalmazza. A kliensalkalmazás egy Java csomagolt fájlként áll elő. jar kiterjesztéssel (*Java ARchive*). A csomagolási eljárás a már jól ismert ZIP eljárást rejti magában, melynek segítségével a csomagban tárolunk főként Java osztályokat (class files), egyéb, az alkalmazás által használt fájlokat (képek, ikonok, egyéb), illetve egyéb, Java specifikus un. meta adatokat (*metadata*), mint például manifesztációs információkat,

és egyéb, a fejlesztő környezet (*IDE*) által bejegyzett jövedékes információkat tárolunk. A manifesztációs információk általában a *META-INF/MANIFEST.MF* néven érhetők el a csomag gyökeréből. Tipikusan a manifesztációs fájlban van meghatározva explicit módon az alkalmazás belépési pontjául szolgáló osztály elérési útvonala a csomagon belül. Egy jar fájlba csomagolt alkalmazás rendelkezhet több belépési ponttal is, ami hasznos lehet az esetben, ha különböző alkalmazás funkciókat szeretnénk elérhetővé tenni az alkalmazás indításakor paraméterezéssel. Az elkészült jar fájlokat általános forma szerint a *java –jar alkalmazas.jar* formában indíthatjuk. Egy jar file indításához szükséges és elégséges egy telepített Java Runtime Enviroment-tel rendelkezni (*JRE*) ami tartalmazza az alkalmazást futtató Java Virtual Machine-t (*JVM*). Amennyiben a jar fájlunkat szeretnénk módosítani, illetve szeretnénk új jar fájlt létrehozni, úgy a Java Development Kit-ben (*JDK*) lévő *jar* nevű program paraméterezésével megtehetjük azt. Az előbb említett *java –jar alkalmazas.jar* formában az alkalmazás manifesztációs információiban megjelölt osztály *main* metódusa lesz meghívva, mint alapértelmezett belépési pont. Ha nincs meghatározva alapértelmezett belépési pont, akkor az említett futtatási forma hibát fog dobni. Ez esetben, és minden olyan esetben mikor el kívánunk térni az alapműködéstől, meg kell adnunk explicit módon az indítani kívánt osztály csomag szintű elérési útvonalát a jar fájl után.

A Web Start-os működtetéshez szükség lesz még létrehozni egy un. JNLP (*Java Network Launching Protocol*) fájlt .jnlp kiterjesztéssel. A jnlp fájl egy XML felépítésű fájl, ami nagyon lesarkítva egy távoli parancsikonként funkcionál. A következő, főbb un. tag-ek lettek felhasználva az alkalmazást indító jnlp fájlban:
<?xml version="1.0" encoding="utf-8"?> - kötelező, XML paraméterek
<jnlp spec="1.0+" codebase="http://valami.com/TruckTracer/" href="TruckTracer.jnlp">
- jnlp specifikáció verziószáma, a jar fájl elérhetősége (aminek meg kell egyezni a jnlp fájl mappa szintű elérhetőségével), illetve a jnlp file neve.

Az információs szekcióban definiálva vannak a következő információk:
Title – az alkalmazás neve. Ezzel a névvel fog szerepelni a telepített programok listájában, illetve, ha telepíttetünk a WS-on keresztül parancsikonokat is, akkor a parancsikonok nevei is ezt a nevet kapjak.

Vendor – Az alkalmazás fejlesztője. Megjelenik a telepített programok listájában az alkalmazás neve mellett.

Homepage – Az alkalmazások listájában az alkalmazás neve mellett megjelenik link-ként. Praktikus itt elhelyezni a fejlesztő honlapját, az alkalmazás dokumentációját, stb.

Description – Az alkalmazás magyarázó neve, vagy rövid leírása. Azt asztalon elhelyezett parancsikon "tool tip"-jeként is megjelenik.

Description kind="short" – A programok listájában megjelenő rövid leírás az alkalmazásról.

Icon – Az alkalmazás ikonja, mellyel megjelenik az asztalon, Start menüben (Windows XP – Vista esetén) és a telepített programok listájában.

Icon kind="splash" – Ha meg van adva, akkor indításkor nem a Java Web Start logója jelenik meg, hanem egy általam meghatározott kép.

Ha a *shortcut online ="true"* kapcsoló jelölve van, akkor az alkalmazás telepítésekor a Start menüben (Windows XP - Vista) elhelyezi az alkalmazás megfelelő ikonnal ellátott parancsikonját. Ha a *menu submenu* kapcsoló fel van tüntetve, akkor a Start menüben egy, a megadott nevű csoport alá fogja besorolni a parancsikont. Ha a *desktop* kapcsoló fel van tüntetve, úgy telepítéskor az asztalon is elhelyezésre kerül egy parancsikon. Kliens oldali Java konfigurációtól függően ezt a szolgáltatást ki lehet kapcsolni, vagy be lehet állítani, hogy a WS rákérdezzen a parancsikon asztalon történő elhelyezésére.

Egy példa részlet:

```
<information>
    <title>Az alkalmazás neve</title>
    <vendor>Az alkalmazás fejlesztője</vendor>
    <homepage href="http://aHonlapCime.com/" />
    <description>Parancsikon Tool Tip</description>
    <description kind="short">Rovid leiras az alkalmazásról</description>
    <icon href="http:// aHonlapCime /TruckTracer/desktopLogo.jpg"/>
    <icon href="http:// aHonlapCime /TruckTracer/Logo.jpg" width="300"
height="427" kind="splash" />
<shortcut online="true">
            <desktop/>
```

```
        <menu submenu="Fejlesztő cég neve"/>
    </shortcut>
</information>
```

A *resources* csoport alatt definiálni lehet, hogy milyen verziójú Java platform meglétét követeljük meg a kliens számítógépen. Ez telepítéskor illetve minden futtatáskor információt közöl a felhasználóval, hogy az alkalmazás működése nem biztosított ez alatt a verzió alatt. Ha meg van adva a *href=http://java.sun.com/products/autodl/j2se* link, akkor az esetben, ha a kliens számítógép nem rendelkezik a megfelelő Java JRE verzióval, úgy a specifikált weboldalról a JWS letölti az igényelt JRE változatot, majd telepíti is automatikusan. A gyakorlati tapasztalatok szerint habár a JWS kizárólag http protokollt igényel, ami az internet hozzáférés miatt általában minden vállalati környezetben a proxy szervereken ki van engedve, ez a típusú frissítés nem minden esetben működik, függően a proxy, és egyéb hálózat védelmi beállításoktól. Pl. *<j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>* A jelenleg elérhető Java platformok: (J2SE version) 1.3, 1.4, 1.5, 1.6.

Definiálni lehet továbbá a jnlp fájlban egyéb beállításokat, mint pl. milyen java proxy beállítások legyenek érvényesek a kliensen futó alkalmazásra, milyen jogosultsággal férhessen hozzá az alkalmazás a kliens számítógép erőforrásaihoz. Jellemzően ezeket a jogosultságokat csak szűkíteni lehet. (Ha az illető felhasználó jogosultságai nagymértékben korlátozottak a kliens számítógépen a Microsoft környezetben használt pl. group policy beállítások miatt, akkor az alkalmazás, mely ennek a policy-nak alá van vetve, nem lesz képes jogosultságokat adni önmaga és így a felhasználó számára a meglévőkön kívül.)

Miután elkészült a konfigurált jnlp fájl, a hozzátartozó jar fájllal együtt el kell helyezni egy web szerveren. Ez értelemszerűen lehet egy Intranetes web szerver is, mint ahogy a Truck Tracer esetében is van. A web szerveren lévő fájlok közül egyedül csak a jnlp fájl kell elérhetővé tenni, publikálni, lévén, hogy a jar fájl-hoz a jnlp-n keresztül férhetünk hozzá. Az elkészült jar fájlunkat hogy a Java Web Start publikálni tudja,

szükséges digitálisan aláírnunk (*certificate*). A Java Web Start-ot alapvetően Interneten keresztüli alkalmazásterjesztésre tervezték, így a biztonság lényeges szempont. Intranetes környezetben ez talán nem olyan számottevő, de a JWS megköveteli, hogy digitális aláírással lássuk el a terjeszteni kívánt alkalmazásunkat. Ezt a certificate –et a JDK-ban lévő *keytool* nevű eszközzel lehet előállítani. Általános formája:
keytool -genkey -validity < érvényesség időtartama napokban> -keystore <aláírás fájl neve> -alias <Fájl Megjelenési neve> -storepass <az aláírás jelszava> -keypass <a fájl saját jelszava> -dname "cn=<kibocsátó személy>, ou=<kibocsátó cég részlege>, o=<kibocsátó cég>, c=<ország>"

Második lépésben a certificate fájlt magát is levédjük:
keytool -selfcert -validity < érvényesség időtartama napokban> -alias <Fájl Megjelenési neve> -keystore <aláírás fájl neve> -storepass <az aláírás jelszava> -keypass <a fájl saját jelszava>

Az elkészült certificate-tel aláírjuk az alkalmazást az ugyancsak a JDK-ban található *jarsigner* nevű eszközzel:
jarsigner –keystore <aláírás fájl neve> <Jar file neve> <Fájl Megjelenési neve> -storepass <az aláírás jelszava> -keypass <a fájl saját jelszava>

Majd ellenőrizzük, hogy megfelelően sikerült –e az aláírás:
jarsigner -verify <Jar file neve>

Az így digitálisan aláírt jar file http protokollon keresztül JWS-tal történő átvitele is biztonságos(abb)á van téve. Ettől kezdve a web szerveren elhelyezett jnlp és jar file JWS technológiájú telepítést tesz lehetővé.
Felmerülhet a kérdés, hogy vállalati, Intranetes környezetben miért nem elég az alkalmazást egy mindenki által hozzáférhető közös, megosztott tárhelyen elhelyezni, hogy aztán onnan indítható legyen.
Első szempontként szerepel, hogy az alkalmazás letöltése nem igényel file share-t, csak http protokollt, mint ahogy az már említve volt. Ez ugye nem lenne probléma, tekintve,

hogy ipari környezetben legalább 10 de inkább 100Mbit/sec –os sávszélesség biztosított a hálózati végpontok között, és a biztonság, ami az Interneten kritikus, itt szinte teljesen biztosított (jól szeparált alhálózat van kiépítve). A különbség abban rejlik, hogy míg file share esetén az alkalmazás egy kvázi távoli helyről van futtatva minden esetben, addig JWS esetén az alkalmazás lokálisan fut. Távolról indítva az alkalmazás minden esetben jelentős hálózati forgalmat generálna, ami abban az esetben lehet kritikus, ha az adott felhasználó külső hálózatból bejelentkezve a vállalati hálózatba próbálja futtatni az alkalmazást. A Cisco Systems által jól bevált és elterjedt VPN technológia (beállítástól függő) konfigurációja szerint 200 Kbit/sec maximális sebesség létesíthető a felépült VPN csatornán. Ez esetben minden indítási alkalommal fix hálózati forgalomként kellene számolnunk az alkalmazás által foglalt ~ 1.5 Mb –os hálózati forgalomra + az alkalmazás által forgalmazott egyéb információk. Ezzel szemben a JWS működése kifinomultabb. A jnlp fájl által első induláskor a JWS ellenőrzi, hogy a kliens számítógépen az adott felhasználó számára telepítve volt-e már az alkalmazás. Ha nem, akkor http protokollon keresztül az alkalmazás letöltődik a kliens számítógépre, és a megfelelő alkalmazás regisztrációs beállítások elvégződnek az operációs rendszerben, mint például parancsikonok létrehozása, programok listájában történő regisztráció, és a kliens gépen lévő Java számára is bejegyzésre kerül az alkalmazás. Gyakorlatilag egy beavatkozás mentes telepítés zajlódik le, nem igényel informatikai területen szerzett előképzettséget, laikusok által is nagy biztonsággal elvégezhető egy ilyen telepítés, tekintve, hogy egy kattintás az elvégzendő feladat. Ha kész a telepítés, az alkalmazás automatikusan indul. Amennyiben már telepítésre került az alkalmazás, akkor a kliens számítógépen elhelyezett parancsikon és a web-en vagy intraneten elérhető jnlp fájl indítása funkcionálisan megegyezik. Bármely indításakor a JWS ellenőrzi a web szerveren elhelyezett jar fájl verzió számát és összeveti a kliensen éppen aktuális verziószámmal. Amennyiben ezek megegyeznek, nem történik frissítés, a kliensen lévő alkalmazás példány elindul. Ha eltérnek a verziók, a JWS letölti a szerveren lévő példányt, telepíti, és futtatja. Ebből a felhasználó csak annyit érzékel, hogy egy Java letöltési ablak jelenik meg, majd eltűnik a letöltés befejeztével. A verziókövetést a JWS belsőleg megoldja, nincs szükség saját magunknak biztosítani erre külön megoldást. Ezzel biztosítva van minden kliensen az alkalmazás legfrissebb példányának birtoklása, nincs szükség minden

kliensen külön manuális telepítés elvégzésére. Elég a megfelelő aláírással ellátott jar fájlt kicserélni a web szerveren, és következő induláskor a felhasználók már az új verziót használják. Ezzel szemben, ha egy közös fájlt futtat minden kliens, akkor egy verzióváltás nem hajtható végre mindaddig, míg a fájlt legalább egy felhasználó is használja. A fájl nem cserélhető. Ezzel szemben a JWS csak akkor fordul az eredeti jar-hoz, mikor valamely kliensen telepítés előtti ellenőrzés folyik, ám ez esetben is lehetséges az eredeti jar fájlt egy újra cserélni gond nélkül.

Fel kell készülni olyan esetekre is, mikor a JWS nem képes kommunikálni a web szerverrel, például egy esetleges karbantartás miatt. Ekkor a jnlp fájlban meg lehet határozni egy <*offline*/> kapcsolóval, hogy az alkalmazás offline üzemmódban futtatható e vagy sem. Esetünkben, ha előfordulhatna az az eset, hogy egyes kliensek az adatbázishoz hozzáférhetnek, de valamilyen oknál fogva némelyek a web szervert elérik, mások meg nem, akkor, ha az offline mód meg van engedve, előfordulhatna olyan helyzet, hogy egy esetleges frissítéskor egyes klienseken eltérő verziójú alkalmazás példányok lennének használatban. Ilyen vagy hasonló esetekben lehet értelme az offline mód kikapcsolásának. A Truck Tracer esetén feltételezzük, hogy a web szerver homogén módon érhető el minden kliensről, így a fentebb említett speciális probléma nem fordulhat elő. Nem képes viszont az alkalmazás kommunikálni a web szerverrel, sem ha nincs hálózati kapcsolat. Ez esetben viszont az adatbázist sem képes elérni. Ebből a megfontolásból az alkalmazás úgy lett felépítve, az offline mód engedélyezve van – nem teszem függővé a rendszer stabilitását a web szerver rendelkezésre állásától - hogy sikertelen adatbázis kapcsolat esetén az alkalmazás elindul, viszont mindössze egy kellően informatív hibaüzenetet ad a probléma minőségére és a javasolt megoldással kapcsolatban.

Az alkalmazás eltávolítására Windows rendszerekben a Programok hozzáadása/törlése menüpont alatt van lehetőség, lévén, hogy a JWS regisztrálja az alkalmazást, mint telepített program.

3.4 Hozzáférhetőség

Rendszer hozzáférhetőségének szabályozása mind hálózati, mind alkalmazáson belüli szinten fontos terület. Egyrészről a rögzített adatok hitelességének megőrzése végett meg kell védeni az alkalmazást illetéktelenek hozzáférésétől, másrészről információk láthatóságának szabályozására is megoldást kell biztosítani. A Truck Tracer sebezhetőségének mértéke a fontosabb területeken redukálva van a lehetőségeknek legjobban megfelelően. A rendszer, beleértve az adatbázist és a kliensalkalmazást, a vállalati hálózaton kívül nem használható, nem hozzáférhető. Az egész hálózat jól szervezett, jó minőségű hálózati eszközök mögött helyezkedik el a külvilág számára. Külső hálózatból csak VPN kapcsolaton keresztül lehet bejelentkezni, amihez nem pusztán a kliens számítógépen lévő VPN kliens program megléte és helyes konfigurációja szükséges, hanem a felhasználót jogosítani kell a külső hálózatból való hozzáféréshez explicit módon. Maga a VPN technológia biztosítja a külső hálózatból belső hálózat irányába és fordítva történő kommunikáció védettségét.
Ráadásul az alkalmazás telepítése a JWS által is külön titkosítva történik az előállított digitális aláírás segítségével.
Hálózati részről gyenge pontnak mondható az alkalmazás és az adatbázis titkosítás nélküli TCP/IP alapú JDBC kapcsolata, viszont mivel maga a hálózat van védve, így a biztonsági kockázat e ponton csökkentett.

Alkalmazás oldalon a hozzáférhetőség felhasználónként van szabályozva. Habár lehetőség lenne, hogy maga a telepítés is csak a megfelelő tartományi felhasználók által legyen elvégezhető a jnlp fájlon elvégzett jogosultság karbantartásával, ez esetünkben fölösleges plusz munkát jelentene. Az alkalmazás un. Windows integrált hitelesítést támogat, minek során a System.property() metódussal induláskor meghatározásra kerül a felhasználó Windows-ban regisztrált azonosítója, mely felhasználónként egyedi. Amennyiben előzetesen fel lett véve a felhasználó az alkalmazás, és annak adminisztrátora által a *felhasználók* táblába a felhasználói azonosítójával és valós nevével, úgy a felhasználó jogosult az alkalmazás elindítására és információk lekérdezésére. Ha valamely állomáshoz szeretne hozzáférni, tehát valamilyen módon operálni szeretne a rendszerben, úgy a megfelelő funkciókat egyenként hozzá kell

rendelni a felhasználóhoz az alkalmazásban. Adatbázis oldalon minden egyes jogosultság, pontosabban, szerep beállítása egy rekord felvételét jelenti a Jogosultságok táblában. A rekord tartalmazza a felhasználó adatbázisban kapott azonosítóját, és a megfelelő állomás azonosítóját.
Előfordulhat szélsőséges esetekben, hogy a rendszer használata közben szükséges egy felhasználó jogosultságait módosítani. Új szerep hozzáadása esetén a hozzáadott jogok az alkalmazás újraindításával lépnek életbe. Amennyiben egy jog megvonásra kerül, az alkalmazás nem érzékeli a változást. Tervezéskor a valós igényeket figyelembe véve az a következtetés született, hogy felesleges erőforrás pazarlás lenne időnként ilyen irányú lekérdezéseket végezni, majd ennek megfelelően reagálni az alkalmazásnak, tekintve, hogy az előfordulása az ilyen eseteknek igen csekély. Szofisztikáltabb megoldás és ugyan olyan hatékony, hogy minden jelentősebb művelet elvégzésekor az üzleti logikát megvalósító tárolt eljárások gondoskodnak az aktuális művelethez tartozó jogok ellenőrzéséről. Amennyiben a felhasználó nem rendelkezik a megfelelő jogosultsággal a művelet megkezdésének időpontjában, a kért művelet nem hajtódik végre, a rendszer ilyen irányú védelme biztosított. A Truck Tracer ezt a megoldást valósítja meg.

3.5 Modularitás, funkcionális integráltság, skálázhatóság

Gyenge implementációs megoldás, és csak kevés esetben indokolt, mikor a kliens oldali alkalmazás funkciók fizikai (fájl) szinten szét vannak bontva. Jellemzően, ezekben az esetekben nincsen felhasználó függő funkció hozzárendelés. A rendszer funkciók a kliens számítógépekhez vannak ilyen esetekben rendelve. Ha kliensenként több felhasználó fér hozzá a számítógéphez, és felhasználónként szeretnénk szétválasztani a hozzáférhető funkciókat, arra lehetőség van, de az csak felhasználónként való lokális telepítéssel lehet megoldani, és a felhasználó számára csak az adott kliens állomásról lesz hozzáférhető a megfelelő alkalmazás. Ilyen és hasonló problémákra jelent megoldást az alkalmazásban a felhasználók - szükségszerű- megkülönböztetése, és felhasználónként történő funkció hozzárendelés a jogosultságok függvényében. A Truck Tracer esetén ez

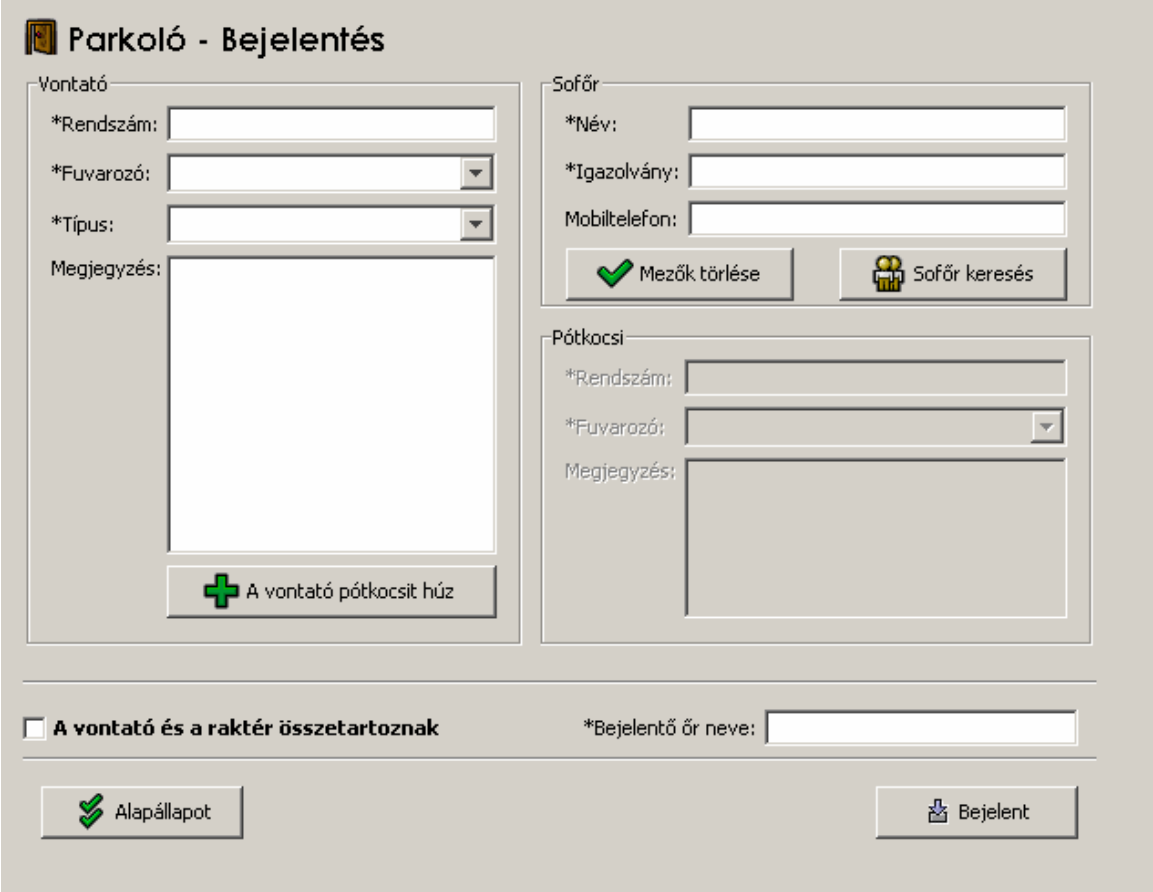
azt jelenti, hogy minden egyes fizikai alkalmazás példány magában hordozza az elérhető összes alkalmazás funkciót, viszont csak a szerepeknek megfelelő modulokat teszi elérhetővé az adott felhasználó számára. Egyrészről nem válik az alkalmazás indokolatlanul bonyolulttá a felhasználó számára, akinek csak egy funkcióhoz - állomáshoz - kell hozzáférnie, de ugyanazon alkalmazás példány által képes pl. a rendszer adminisztrátora elvégezni minden műveletet, ami meg lett fejlesztve az alkalmazásban, az által hogy a saját felhasználó nevében futtatja az alkalmazást. Számos telepítési, frissítési, verziókövetési, és jogosultság változásból adódó problémát előzhetünk meg, ha megfelelő módon megvalósított funkcionális integráltságot implementálunk, mint ahogy ez történt a Truck Tracer estén is. Ennek elsődleges eszközei a már említett JWS, és a felhasználók megkülönböztetése az alkalmazás oldalán is.

Az esetek egy részében a meglévő és bekonfigurált rendszer lehetőségeit kell korlátozni pl. felhasználókként, mint ahogy az imént kifejtésre került. Egyéb esetekben, például, a fizikai folyamatok csekély mértékű változásakor a rendszerbővítésre, fejlesztésre szorul, aminek a rendszer funkcionális skálázhatósága szab határt. A folyamatok „csekély” mértékű változására a rendszert egyfajta paraméterezhetőségével fel lehet készíteni. Esetünkben ilyen „csekély” változásnak számít az alkalmazás és így a rendszer képességeit tekintve, ha pl. az állomások listája változik, egy vagy több állomás kerül bevezetésre, vagy megszüntetésre. Vagy ilyen eset lehet a rendszer adoptálása más telephelyen, némileg eltérő folyamat specifikációval – eltérő állomás nevek, állomás struktúra, jogosultság kombinációk, stb.. A Truck Tracerben alkalmazott megoldás erre a problémára úgy történik, hogy vannak, un. fix állomások, amelyek a rendszer működéséhez elengedhetetlenül szükségesek. Ezek a fix állomások minden telephelyen megvannak, de meg is kell, hogy legyenek. Ilyen állomások a *parkoló* állomás, *teherporta be, teherporta ki*, és néhány további fiktív állomás, amelyek a valóságban inkább szerepekként tekinthetők, mint a “k*iengedve*” állomás, *nem ellenőrzött* állomás, és a *forgalomirányító*. A többi állomás illetve szerep listája bővíthető, redukálható, tetszőlegesen lehet felhasználókat hozzárendelni az adott szerepekhez, stb. Ha más telephelyen kell bevezetni a rendszert, akkor ezek az opcionális állomások a rendszer éles futása közben is konfigurálhatók teljes mértékben, amennyiben pedig a fix állomások

neveinek módosítására van szükség, ez is elvégezhető, bár ez utóbbi esetben a szolgáltatás átmeneti, rövid ideig tartó szünetelésével jár. A fix állomások neveinek módosítását kivéve minden műveletet el tud végezni a rendszer adminisztrátora a rendszer működése közben is. Fix állomások módosításához fejlesztői beavatkozásra van szükség, de ilyen eset csak rendszer implementációkor fordul elő, tekintve, hogy az ilyen eset a fizikai folyamatok nagyfokú átszervezéséből adódik, ami a fizikai folyamatok leállásával is jár. Ilyen módon a rendszerünk rendelkezésre állását egyik művelet sem befolyásolja – negatív irányba.

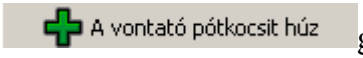
3.6 Felhasználói felület tervezése

3.6.1 Parkoló állomás



A fenti képen, a teherportán az ott szolgálatot teljesítő őrök által látható képernyőkép látható. Az űrlap értelemszerű kitöltésével az adott jármű információi rögzítésre kerülnek a rendszerben és minden egyéb állomás által hozzáférhetővé válnak az információi. A jármű lehet egy pótkocsival rendelkező vagy nem rendelkező kamion, illetve egy pl. egyterű kishaszon-gépjármű. Ez a funkció arra az esetre került bevezetésre, mikor a jármű megérkezett a telephelyez, de még nem kapott engedélyt a belépésre. Ekkor az információi csak lejelentésre kerülnek, és az érintett állomások által azonnal

látható. Ez a megoldás azt az esetet hivatott kiküszöbölni, mikor pl. egy kamion, amely alapanyagot szállít, beáll a parkolóba, de a bent érintett állomások nem tudnak a kamionról, csak ha időközönként telefonálgatnak a Teherportára, hogy megjött-e a kamion vagy sem. Legrosszabb esetben ez termelés leállást is eredményezhet, ha a kellő időben nem áll rendelkezésre a kamion által szállított alapanyag. Másrészt, ha a fuvarozó kötbérrel kívánjuk terhelni az esetleges késés miatt, úgy a késés ténye bizonyítható.

Mint látható az űrlapon a megfelelő információ beviteli mezők jól el vannak különítve egymástól csoportba szervezve, így kellően intuitív a felhasználói felület. A 

gomb megnyomásával a Pótkocsi szekció aktívvá válik, jelezve, hogy nem csak vontató, hanem pótkocsi is bejelentésre kerül. A „Vontató és a raktér összetartoznak” választómezőt abban az esetben kell kitölteni, ha egy egyterű kishaszon-gépjárműről van szó, pl. Ford Transit. Ekkor a rendszerben megjelenik egy vontató és egy pótkocsi, de ugyanazzal a rendszámmal, és a továbbiakban logikailag egy egységként kerülnek kezelésre. Az adatbevitelt gyorsító mechanizmus a sofőr adatainak feltöltését segítő programrész - 

gomb. Ha az illető sofőr adatai már előzőleg rögzítve lettek valamikor akkor az adatbázis információk alapján felkutathatók azok, és automatikusan kitöltődnek a megfelelő mezők a sofőr kiválasztásával.

3.6.2 Teherporta Be

A Teherporta Be állomás ugyancsak a teherportáról férhető hozzá, és hasonló felépítésű a parkoló állomáséhoz. Markáns különbség azonban egy „Parkoló kamionok” feliratú gomb. Ha a kamion előzőleg már be lett jelentve, akkor a gomb megnyomásával a parkoló kamionok listájából ki lehet választani a járművet, melynek információival feltöltődnek a megfelelő mezők, ezzel is gyorsítva a felhasználók munkáját. A Pótkocsi szekcióban a „Forgalomirányítás szükséges” feliratú választó mező látja el azt a funkciót, ami a folyamatábrában (lásd a dolgozat elején) látható belépés utáni elágazás irányát meghatározza. Olyan esetekben mikor a jármű a cég területére belép, de a telephely

biztonsági szolgálata által nem áll közvetlen ellenőrzés alatt, úgy a kapcsolómező üresen marad. Ekkor egy üres útvonal terv határozódik meg a járműnek:

0. Parkoló – Függően attól hogy előzőleg várakozott-e a parkolóban vagy sem
1. Teherporta Be
2. Nem ellenőrzött
3.Teherporta Ki

Ekkor a "Nem ellenőrzött" állomás bejegyzés jóváhagyása nem kötelező, lévén, hogy ez egy fiktív állomás, mely az olyan eseteket jelöli mikor a jármű pl. egy kávéautomatás szerviz kocsi, vagy a büfébe hozott árut, vagy ahogy a folyamatábrán is látható, a HIRSCH nevű belső céghez kíván belépni. Ekkor az állomás nem lesz senki által sem jóváhagyva a rendszerben, viszont az ilyen esetek miatt a "Nem ellenőrzött" állomás megléte feltétlen szükséges így ez az állomás is un. fix állomás, a rendszerből nem eltávolítható.

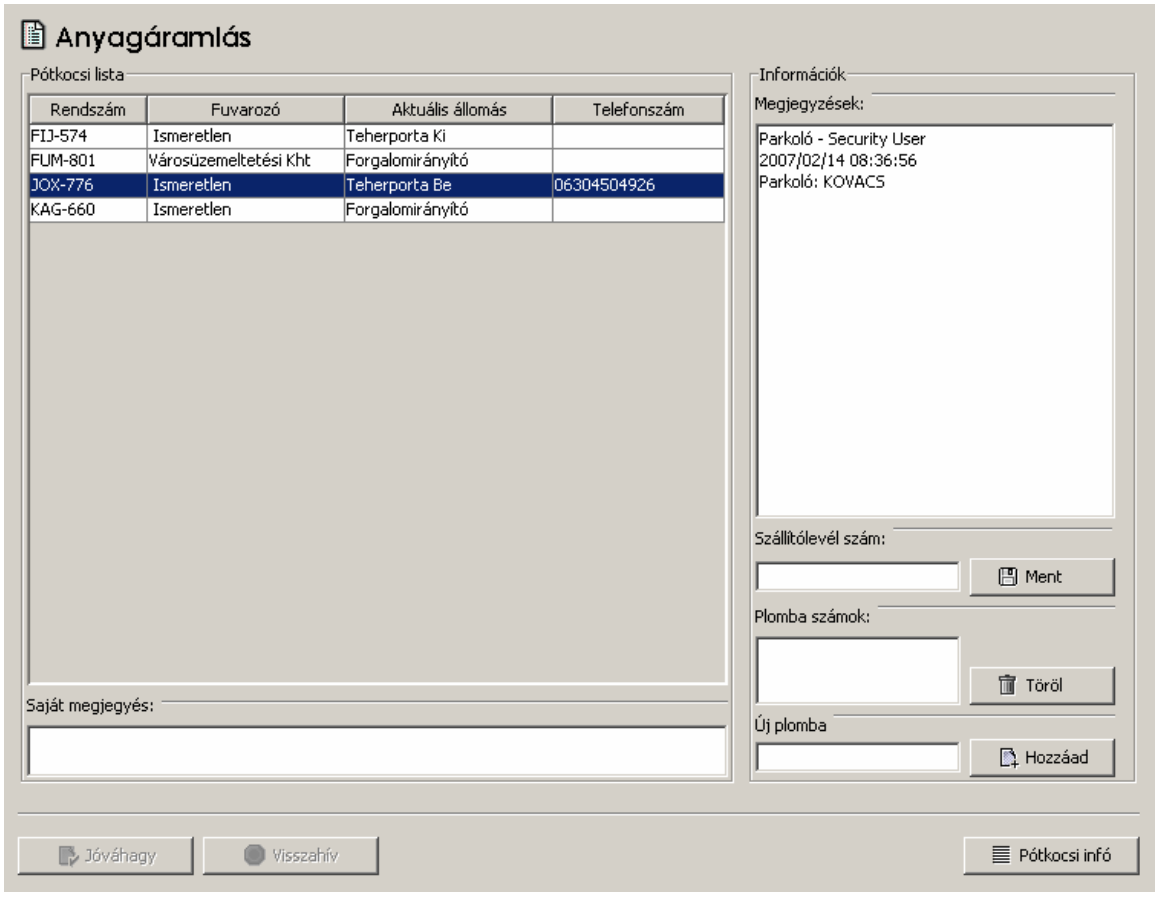
Ha a Forgalomirányító mező be van kapcsolva, akkor a fent említett útvonal tervben a 2. pozícióban a "Nem ellenőrzött" állomás helyett a Forgalomirányító nevű állomás szerepel majd, de kötelező jóváhagyást kívánva. Ekkor belépés után a Forgalomirányító állomás jóváhagyására vár a jármű, aktuális állomásként megjelölve azt. A Forgalomirányító állomás, mint ahogy említésre került már, nem feltétlenül létező állomás, inkább csak egy "szerep", melyhez jogot lehet adni az érintetteknek. Esetünkben ilyen szerepet töltenek be a Teherporta felhasználói és az Anyagáramlás felhasználói. Nem minden esetben kerül az Anyagáramlás elé a jármű, viszont még lehet szükség forgalomirányításra, ezért is volt szükség a teherporta felhasználóit is jogosítani forgalomirányításra.

Előfordulnak olyan esetek, mikor sok járműnek mindig ugyanazt az útvonalat kell bejárnia, de legalábbis nagyban hasonlót. Erre az igényre épült be az Útvonal Sablon nevű modul. A sablonokat a rendszer adminisztrátora tudja karbantartani az állomások karbantartási szekciójában. Az adminisztrátor által definiált útvonal sablonokból lehet választani a jármű belépésekor. A belépést követően az útvonal sablonban definiált állomások a sorrendiségnek megfelelően egyből rátöltődnek a jármű útvonal tervére, ezzel megspórolva a forgalomirányításkor elvégzendő munkát, ha a megfelelő információk beléptetéskor rendelkezésre állnak. Az elkészült útvonal lista természetesen

ugyanúgy módosítható később mintha kézzel történt volna meg az útvonal terv meghatározás.

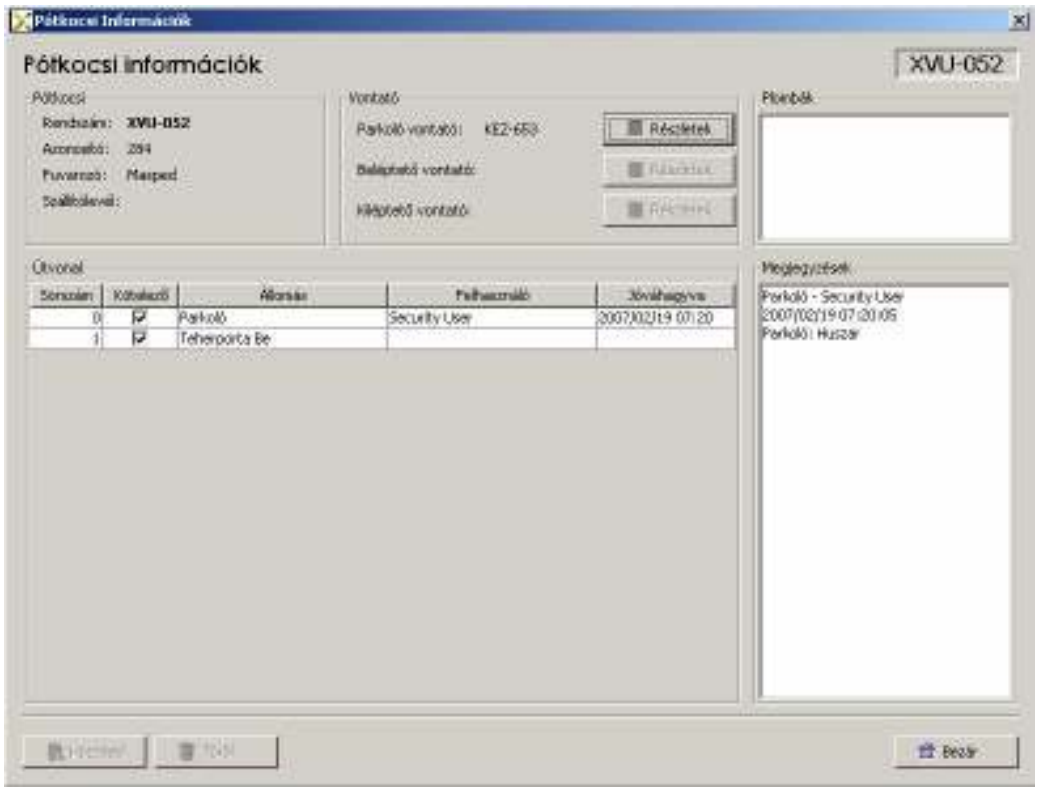
Beléptetéskor a sikeres adatbázis művelet után nyomtatásra kerül egy vagy kettő, un. Kamionkísérő lap, amely papír alapon is reprezentálja a vontató és pótkocsi információkat. Ennek jelentőssége a felhasználói jóváhagyások hitelesítésében van. Ha egy felhasználó jóváhagy egy kamiont elektronikusan, akkor az elektronikus aláírása felkerül a pótkocsihoz tartozó útinformációk megfelelő rekordjába. E mellett viszont a sajátkezű aláírásával is hitelesítenie kell tevékenységét a kamionkísérő lapon vagy lapokon, amit a sofőr a teherportán való kiléptetéskor ad le. Ilyen módon 2 pontos is hitelesítve vannak az elvégzett műveletek, amelyek visszakereshetők.

3.6.3 Belső állomások

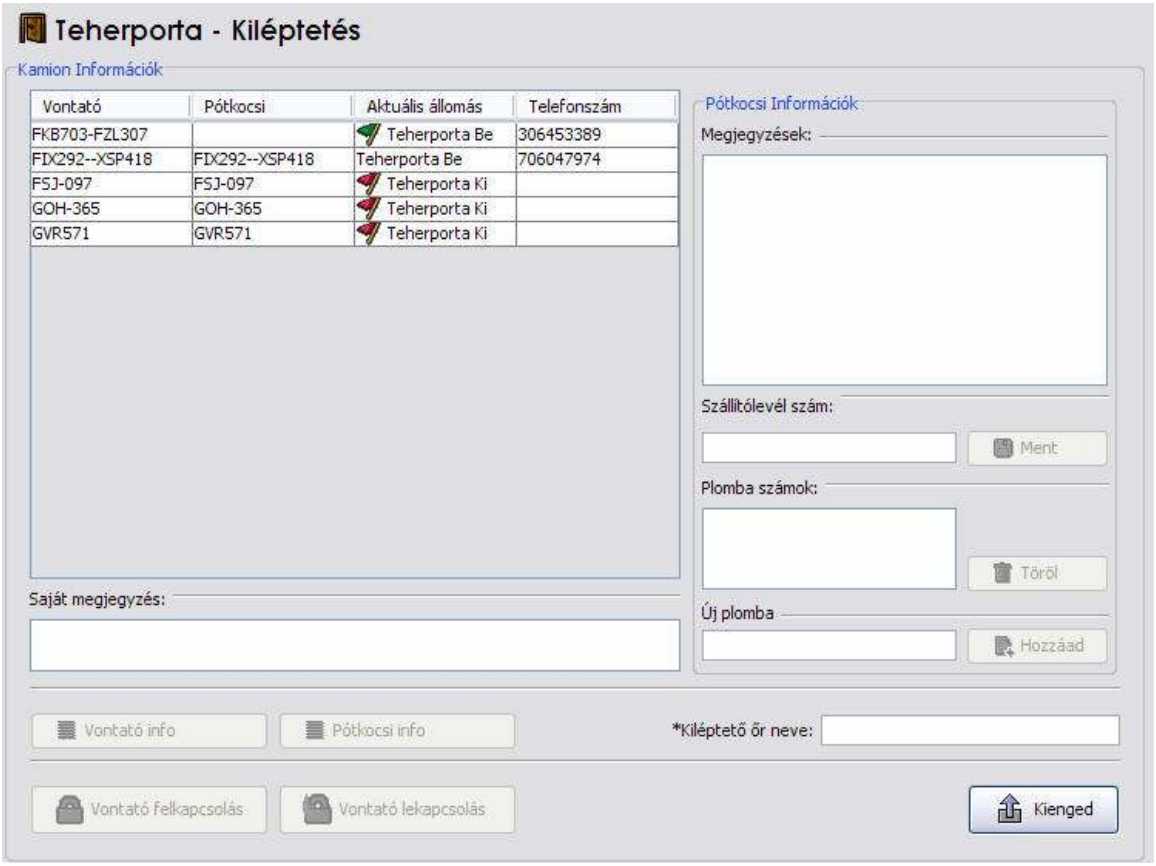


A képen egy átlagos “benti” állomás kezelő felülete látható. Értelemszerűen az “Anyagáramlás” név helyén a megfelelő állomás neve szerepel. A rendszer funkcionális skálázhatósága a gyakorlatban ennek az alkalmazás résznek a paraméterezésével történik. Ha az adott felhasználónak joga van pl. az Anyagáramlás állomás, vagy a Vám állomás kezeléséhez, akkor ugyanaz a Java osztály kerül példányosításra annyi példányban ahány belső állomáshoz van joga a felhasználónak, de az osztály példányai paraméterezésben különbözni fognak (Állomás neve, állomás adatbázisbeli azonosítója), így biztosítva a bővíthetőséget.

A Pótkocsi lista szekcióban a bejelentett vagy már bent álló pótkocsik információit láthatjuk. A benti állomások számára indifferens a pótkocsi vontatójának információja. Kizárólag pótkocsi rendszámok, és egyéb pótkocsi információkra van szükségük a felhasználóknak a munkájuk elvégzéséhez, függetlenül a vontató rendszámától, vagy, hogy egyáltalán tartozik e jelenleg a pótkocsihoz vontató vagy sem. Ha szükséges elérhetők a vontató információk is közvetve a “Pótkocsi info” gomb által biztosított információs felületről, mint ahogy azt a következő ábrán is láthatjuk.



3.6.4 Teherporta Ki



A Teherpota Ki állomás ugyancsak a teherportáról érhető el, de lehetőség van, arra is, hogy a Teherporta Be és Ki állomások fizikailag külön helyen helyezkedjenek el. Ugyanúgy lehetőség van itt is még kilépés előtt pótkocsi információkat módosítani, mint bármely más belső állomáson (szállítólevél, plomba szám). A lényeges különbség egyéb belső állomásokkal szemben, hogy itt a jóváhagyást a Kienged gomb megnyomása biztosítja, melynek hatására a jármű út információi az Útvonal táblából átkerülnek a Történet táblába. Ezáltal archiválásra kerülnek a jármű információi, amelyet a Kamion Történetből lehet elérni. E mellett az útinformációkhoz még archiválás előtt felvételre

kerül egy “Kiengedve” nevű, ugyancsak fiktív állomás, amely Aktuális állomásként jelenik meg a Pótkocsi nevű táblában a jármű rekordbejegyzésénél.
Másik különbség, hogy itt egy ablakban jelennek meg a vontató és pótkocsi alap információk, amely a köztük lévő aktuális relációt is jelzi. Ilyen módon van itt lehetőség le –illetve felkapcsolni egy vontatót a pótkocsira/ról. Szükséges ez olyan esetben például, ha egy pótkocsit nem ugyanaz a vontató húz ki a telephelyről, mint amelyikkel bejött.

4 Összegzés

Az ilyen módon elkészül, és ismertetett rendszer a gyakorlati tapasztalatok alapján megfelelően kielégíti a meglévő folyamatok által támasztott követelményeket, elsősorban a rögzített információk minőségét és azok visszakereshetőségének lehetőségét érintő területeken. Elsősorban az adatbázis struktúra lehetőségeiből adódóan lehetőség van igény szerinti további riportokat készíteni, legegyszerűbben web alapon például a Microsoft SQL Server család 2005-ös verziójában továbbfejlesztett Reporting Service nevű szolgáltatással is. Esetleges további fejlesztések elsősorban ezen a területen a legérdemreméltóbbak, tekintve, hogy az alkalmazás funkciók, és a gyűjtött információk strukturáltsága mára már teljesen kielégíti az eddig felmerült követelményeket. Az eddig bevezetett telephelyek nagy részén előfordult igényeket pedig mindeddig túl is teljesíti több területen is.

5 Irodalomjegyzék

Java Standard Edition Documentation, version 1.6: http://java.sun.com/javase/6/docs/api/

Java Development Kit 1.6 - http://java.sun.com/javase/downloads/index.jsp

Netbeans IDE - http://www.netbeans.org/

Microsoft SQL JDBC driver, documentation: http://www.microsoft.com/downloads/details.aspx?familyid=4f8f2f01-1ed7-4c4d-8f7b-3d47969e66ae&displaylang=en

Microsoft SQL Server Reporting Services - http://www.microsoft.com/sql/technologies/reporting/default.mspx

Microsoft SQL 2000 Server - http://www.microsoft.com/sql/default.mspx

Microsoft SQL 2000 Server Documentation- http://msdn2.microsoft.com/en-us/sql/aa336367.aspx

Microsoft Press - Course 2072A: Administering a Microsoft SQL Server 2000 Database